Compiled Labelled Deductive Systems for Access Control

Krysia Broda and Alessandra Russo
Department of Computing
Imperial College London

ABSTRACT. This paper proposes a Compiled Labelled Deductive System, called AC_{CLDS}, for reasoning about role-based access control in distributed systems, which builds upon Massacci's tableau system for role-based access control. The AC_{CLDS} system overcomes some of the limitations of Massaci's approach by combining its multi-modal propositional language with a labelling algebra that allows reasoning explicitly about *dynamic properties* of the accessibility relations. This combined feature, which is typical of the Compiled Labelled Deductive framework, facilitates a sound and complete, and more natural AC_{CLDS} reasoning mechanism than Massacci's sound and only partially complete tableau system. Limitations of the usefulness of Massacci's multi-modal logic in formalising access control systems are also discussed, showing that they relate to the initial formulation of Abadi's calculus for access control. Solutions for overcoming these limitations are briefly proposed within the context of the AC_{CLDS} system.

1 Introduction

Labelled Deductive Systems (LDS) is a methodology initially proposed by Gabbay in [18] for both the theoretical study of logics and the development of logical systems suitable for the needs of specific applications. In the LDS approach, a basic unit of information is not just a formula but a *labelled formula*, where the label belongs to a given labelling algebra. Additional information, for example regarding the overall structure of the data or the underlying semantic properties of the logic, can be explicitly embodied, via the labels, in the object language of the system. Derivation rules act on both the labels and the formulae, according to certain fixed rules of propagation. The addition of this separate "object-level" dimension (i.e. labels and labelling algebra) enables the development of a *uniform* system

for different logics within the same family, in that every deduction rule can be applied to each of these logics and their differences captured entirely by the labelling algebra. The general proposals in [18] acted initially as a manifesto or starting point for a wide programme of research. Detailed investigations were, for example, undertaken into the benefits of using the LDS methodology to reformulate intuitionistic modal logics [28] and substructural logics [8, 15, 19]. Specialised frameworks based on LDS where also proposed [3, 6, 10, 26]. Among these the *Compiled Labelled Deductive Systems* (CLDS) approach demonstrated how LDS techniques facilitates the reformulation and generalisation of a large class of modal logics and conditional logics [9, 26]. This chapter contributes to this programme of research by showing how the CLDS approach can be used to develop a *sound* and *complete* logical system for role-based access control, called AC_{CLDS} system, which enables the formalisation of and reasoning about access control in distributed systems.

Various logical formalisms for (role-based) access control have been proposed in the literature [1, 13, 22, 24, 29]. These all attempt to model and reason about the role-based hierarchical relationships between principals and between groups, where privileges can be inherited along the hierarchical structure. The approach in [22] is based on a first-order logic representation of access control policies expressed in programming-style languages (such as Ponder [16] and Tower [20]), the $\mathcal{PDL}$ approach in [2, 29] uses an *event-condition-action* formalisation and relative implementation in logic programming, and [13] deployes a deontic-based language, also translated into first-order logic for reasoning purposes. Abadiand Massacci, on the other hand, have proposed a modal logic formalisation of access control systems, referred in this chapter as the *logic for access control*. In particular, Abadi work in [1] provides a first axiomatisation of basic notions of access control systems, such as principals, privileges, authentication and delegation ([27]). This is based on multi-modal logic, where modalities are assumed to be abstractions of roles, and constraints over the different accessibility relations are used to define various delegation principles. The inference capability of this axiomatisation was shown by Massacci to be limited [24]; for instance the so-called "hands-off axiom" $\neg(A \texttt{ says } \bot) \rightarrow (A \texttt{ controls } (P \Longrightarrow A))$, used to express that a principal A hands-over its privileges to another principal P, is valid in Abadi's calculus, but not provable. This is partly due to the fact that universal formulae of the form $P \Longrightarrow A$, although expressable in the languange, are not axiomatisable within a Hilbert System [5] and therefore not axiomatisable in Abadi's calculus. The labelled tableau system, proposed subsequently by Massacci in [24], overcomes this limitation. This system enables explicit reasoning about formulas of the form $P \Longrightarrow A$,

so facilitating, in turn, dynamic inference of properties of the accessibility relations. For instance, the inference of $P \Longrightarrow P|P$ forces *dynamically* the accessibility relation for the principal P to be transitive. However, as stated in [24], Massacci's labelled tableaux system is only partially complete. This is because the tableaux rules do not allow these universal formula to be unfolded over compound principals.

This chapter aims to address the limitations of both Abadi's and Massacci's logic-based approaches. It builds upon Massacci's work to propose a CLDS system, called AC_{CLDS} system, where explicit reasoning about accessibility relations of both primitive and compound principals can be performed, so allowing the development of a sound and complete proof system for role-based access control in distributed system. In the CLDS approach a theory combines a logical theory written in an object *language* and a *labelling algebra*, written in a first-order *labelling language*. The latter is used to axiomatise semantic and/or proof theoretic properties that uniquely identify the underlying object logic. In the AC_{CLDS} system, the labelling algebra is defined as a binary first-order theory axiomatising the properties of atomic and compound principals, whereas the object language is the multi-modal language given in [24]. The two languages are combined via the notion of a *declarative unit*, written as $\alpha : \lambda$, which expresses that the formula α is *true* at the label (i.e. point) λ. The inference rules include *basic* rules for the classical connectives, which appear in the formula part of the declarative units, rules, driven by the labelling algebra, for reasoning explicitly about the accessibility relations, as well as additional *specialised* rules for reasoning about modal connectives and their interaction with primitive and compound principals. The modularity of the CLDS approach is also reflected by the AC_{CLDS} system. The set of basic rules is in fact the same as the set of basic rules that any logical system would require when formalised within a CLDS framework [11]. It is the set of specialised rules that provides the AC_{CLDS} system with the specific characteristic of a logical system for access control. It is this specialised set of rules that allows explicit unfolding of modal properties through compound principals so enabling the AC_{CLDS} system to be complete.

The chapter is organised as follows. Section 2 gives a brief overview of the language features and semantics of a CLDS approach. The general natural deduction style proof system for a CLDS system is described together with a general model–theoretic semantics, based on a translation method into classical logic and related notion of semantic entailment. Section 3 introduces basic concepts of access control for distributed systems and the features that a logic for this type of application should have. The specific AC_{CLDS} system is described in Section 4, and proofs of soundness and

completeness are given in Section 5. The chapter ends in Section 6 with a general discussion on how to further improve the logic for access control so to capture more realistic behaviours of access control systems.

Notation Throughout the chapter predicate symbols, terms and meta-variable denoting principals begin with an upper-case letter, whereas other constants, variables and function symbols begin with a lower-case letter. Greek letters meta-variables are used to refer to terms different from the principals, and expressions in the system. Larger entities such as structures, sets, theories and languages are symbolised in calligraphic font, $\mathcal{A}, \mathcal{B}, \mathcal{C}$, etc.. The power set of a given set $\mathcal{A}$ will be denoted by $PW(\mathcal{A})$.

2 Overview of CLDS

This section introduces the CLDS approach within the context of modal logic, by giving a brief description of its basic language features, syntax and semantics[1]. A CLDS is a logical framework in which explicit reference can be made to specific possible worlds and to relationships between possible worlds, whilst retaining the conventional syntax of modal logic. In this approach, a logical theory, called a *configuration*, is a generalisation of standard modal theories, as it is defined as a structure of arbitrary modal theories, the proof-theoretical presentation is uniform across the different logics of the same family and retains the conciseness of standard modal logic proof systems. The semantics is defined through a translation into first-order logic, whereby the notions of model, satisfiability of a configuration and semantic entailment are given in terms of classical semantics.

2.1 Languages and Syntax

A CLDS language is, in general, defined as an ordered pair $\langle \mathcal{L}_P, \mathcal{L}_L \rangle$, where $\mathcal{L}_P$ is a given *object* language and $\mathcal{L}_L$ is a *labelling language* defined as a binary fragment of a first-order language. In the propositional case, the object language is composed of a countable set of propositional letters, $\{p, q, r, \ldots\}$ and boolean connectives $\{\neg, \wedge, \rightarrow\}$. The labelling language $\mathcal{L}_L$ includes, instead, a countable set of constant symbols $\{s_0, s_1, s_2, \ldots\}$, a countable set of variables $\{x, y, z, \ldots\}$, a set of binary predicate symbols $\{R_1, R_2, \ldots\}$, called *R-predicates*, the set of logical connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ and the quantifiers $\forall$ and $\exists$.

For proof-theoretic purposes, the labelling language $\mathcal{L}_L$ is, in general, extended with additional sets of Skolem terms, which are used only in the definition of some labelled natural deduction rules of the system. The extended labelling language is then called a *semi-extended labelling language* and is defined as follows.

[1] For a more general presentation of a CLDS, the reader is refererd to [11].

DEFINITION 1.1. Let $\mathcal{L}_P$ be a propositional language and $\{\alpha_1, \alpha_2, \ldots\}$ be the set of all wffs of $\mathcal{L}_P$. The *semi-extended labelling language* $Func(\mathcal{L}_P, \mathcal{L}_L)$ is defined as the language $\mathcal{L}_L$ extended with a set of *skolem* function symbols $\{sk^n_{\alpha_1}, sk^n_{\alpha_2}, \ldots\}$ where $n \geq 0$.

Ground terms of $Func(\mathcal{L}_P, \mathcal{L}_L)$ are called *labels*. The specific definition of the skolem symbols $sk^n_{\alpha_i}$ and their interpretation depend on the specific CLDS system. Similarly for the R-predicates. In the case of modal logics labels refer to possible worlds and R-predicates to the accessibility relation. For access control logics, R-predicates denote the accessibility relation for the principals and they provide the direction to which privileges can be propagated amongst various roles.

Syntax. The CLDS language facilitates the formalisation of two types of information, (i) what holds at particular points, expressed by a syntactic entity called *declarative unit*, and (ii) which points are in relation with each other and which are not, expressed by the syntactic entity called an *R-literal*. A declarative unit is defined as a pair *"formula:label"*, where the label component is a ground term of the semi-extended labelling language $Func(\mathcal{L}_P, \mathcal{L}_L)$ and the formula is a wff of the language $\mathcal{L}_P$. An R-literal is any ground literal in the semi-extended labelling language involving a binary R-predicate, usually of the form $R_i(\lambda_1, \lambda_2)$ and $\neg R_i(\lambda_1, \lambda_2)$, where λ_1 and λ_2 are labels, expressing that λ_2 is or is not related to λ_1 by the accessibility relation R_i. For each R-literal Δ, the *conjugate* of Δ, written $\overline{\Delta}$, is the opposite in sign of Δ.

This combined aspect of the CLDS syntax yields a definition of a CLDS theory more general than the traditional definition of a logical theory. Informally, a CLDS theory, called a *configuration*, is composed of two sets, a set of R-literals and a set of declarative units. For example the pair of sets $\{R_Q(w_0, w_1))\}$ and $\{P \texttt{ says } r : w_0, P \Longrightarrow Q : w_2, r : w_1\}$ is a consistent AC_{CLDS} configuration, which states that the principal P requests r (P says $r : w_0$) and this request is granted at the P accessible world w_1 ($r : w_1$) since w_1 is Q accessible ($R_Q(w_0, w_1)$)) and P inherits privileges from the principal Q ($P \Longrightarrow Q : w_2$). The formal definition of a configuration is given below.

DEFINITION 1.2. Given a CLDS language, a configuration is a tuple $\langle \mathcal{D}, \mathcal{F} \rangle$ where $\mathcal{D}$, called a *diagram*, is a finite set of R-literals and $\mathcal{F}$ is a function from the set of ground terms of $Func(\mathcal{L}_P, \mathcal{L}_L)$ to the set PW(wff($\mathcal{L}_P$)) of sets of wffs of $\mathcal{L}_P$.

To capture different classes of logics within the CLDS approach, an appropriate first-order theory $\mathcal{A}$, written in the language $\mathcal{L}_L$ and called a *labelling algebra*, needs to be defined. This is used by the proof system to

infer properties about the diagram part of a configuration. Inference rules and the notion of a derivability relation are, in fact, defined between configurations. A set $\mathcal{R}$ of such inference rules, together with a CLDS language $\langle \mathcal{L}_P, \mathcal{L}_L \rangle$ and a labelling algebra $\mathcal{A}$, uniquely define a CLDS system (i.e. for any CLDS system $\mathcal{S}$, $\mathcal{S} = \langle \langle \mathcal{L}_P, \mathcal{L}_L \rangle, \mathcal{A}_S, \mathcal{R}_S \rangle$).

2.2 A "basic" natural deduction system

The "structural" aspect of a CLDS theory stimulated the idea of defining deductive processes that describe how configurations "evolve" by reasoning within and between the local theories associated with each point in a configuration or by reasoning about the diagram of a configuration. An inference rule of a CLDS is defined between configurations as follows.

DEFINITION 1.3. An inference rule $\mathcal{I}$ is a set of pairs of configurations, where each such pair is written as $\mathcal{C}/\mathcal{C}'$. If $\mathcal{C}/\mathcal{C}' \in \mathcal{I}$ then $\mathcal{C}$ is called an *antecedent* configuration of $\mathcal{I}$, and $\mathcal{C}'$ an *inferred* (or *consequence*) configuration of $\mathcal{I}$ with respect to $\mathcal{C}$.

All the rules except one have the effect of expanding the antecedent configuration. These rules can extend an antecedent configuration $\mathcal{C}$ with either a declarative unit, or with an R-literal, or with both. However, configurations equal or smaller than the antecedent one can also be inferred. This is facilitated by an inference rule called the $\mathcal{C}$-Reduction (C-R) rule. Only a graphical representation of the inference rules will be used throughout the chapter and the reader is referred to [11, 26] for a complete formal definition of a CLDS proof system. Tables 1.1 and 1.2 show, respectively, the classical rules for the $\rightarrow$, $\neg$ and $\wedge$ connectives[2] and the basic rules for R-literals, which are all common to any CLDS system. In this graphical representation, $\mathcal{C}\langle \alpha : \lambda \rangle$ (respectively $\mathcal{C}\langle \Delta \rangle$) denotes that $\mathcal{C}$ includes a declarative unit $\alpha : \lambda$ (respectively R–literal Δ). Declarative units and R-literals contained in square brackets are assumptions introduced within a derivation that are subsequently discharged. $\mathcal{C}'\langle \pi \rangle$, where π is a declarative unit or an R-literal, represents that the inferred configuration $\mathcal{C}'$ is $\mathcal{C}$ extended with π. $\tilde{\mathcal{C}}$ are the configurations derived in subderivations after adding to the antecedent configuration $\mathcal{C}$ temporary assumptions.

In most CLDSs, the labels occurring in the classical rule are the same, except for the $(\neg\mathcal{I})$ rule. The notion of contradiction (or inconsistency) in the CLDS approach strictly depends on the type of logic. Modal logics, and therefore the logic for access control, include a classical notion of inconsistency, for which the symbol $\perp$ used in the $(\neg\mathcal{I})$ rule is a short-hand for any $\mathcal{L}_P$ wff of the form $\alpha \wedge \neg\alpha$. In CLDSs where the notion of classical

[2] Introduction and elimination rules for $\vee$ can be derived using the equivalence $\alpha \vee \beta \equiv \neg(\neg\alpha \wedge \neg\beta)$.

Table 1.1. Classical CLDS Rules.

$$\frac{\mathcal{C}\langle \alpha \rightarrow \beta : \lambda, \alpha : \lambda \rangle}{\mathcal{C}'\langle \beta : \lambda \rangle}\ (\rightarrow\mathcal{E}) \qquad \frac{\begin{array}{c}\mathcal{C}\langle [\alpha : \lambda] \rangle \\ \vdots \\ \tilde{\mathcal{C}}\langle \beta : \lambda \rangle\end{array}}{\mathcal{C}'\langle \alpha \rightarrow \beta : \lambda \rangle}\ (\rightarrow\mathcal{I})$$

$$\frac{\mathcal{C}\langle \alpha \wedge \beta : \lambda \rangle}{\mathcal{C}'\langle \alpha : \lambda, \beta : \lambda \rangle}\ (\wedge\mathcal{E}) \qquad \frac{\mathcal{C}\langle \alpha : \lambda, \beta : \lambda \rangle}{\mathcal{C}'\langle \alpha \wedge \beta : \lambda \rangle}\ (\wedge\mathcal{I})$$

$$\frac{\mathcal{C}\langle \neg\neg \alpha : \lambda \rangle}{\mathcal{C}'\langle \alpha : \lambda \rangle}\ (\neg\neg) \qquad \frac{\begin{array}{c}\mathcal{C}\langle [\alpha : \lambda] \rangle \\ \vdots \\ \tilde{\mathcal{C}}\langle \bot : \lambda' \rangle\end{array}}{\mathcal{C}'\langle \neg \alpha : \lambda \rangle}\ (\neg\mathcal{I})$$

inconsistency applies the particular labels λ and λ' in the $(\neg\mathcal{I})$ rule are not required to be the same – a contradiction in some part of a configuration introduces an inconsistency to the configuration as a whole.

The R-literals rules, given in Table 1.2, facilitate reasoning about the diagram of a configuration, using the particular labelling algebra $\mathcal{A}$ under consideration, and infering R-literals and declarative units which would not be inferred using only the logical connectives. For logics of the same family (*e.g.* different modal logics), the (R-A) rule captures *entirely* the difference between one CLDS system and another, allowing all other inference rules to be equally applicable to any CLDS system. In the AC_{CLDS} system, the (R-A) rule allows to reason about properties of principals explicitly within the derivation process. The rules $(\bot\mathcal{E})$ and $(R\mathcal{I})$ express additional forms of interactions between the R-literals and the declarative units. The $(\bot\mathcal{E})$ rule allows the inference of falsity (i.e. $\bot : \lambda$) whenever R-literals and its negations are present in a configuration. This is necessary because since no compound classical formulae with R-literals can be inferred in a configuration, inconsistency of this form would not otherwise be captured. The $(R\mathcal{I})$ rule enables instead the derivation of R-literals in the presence of a logical inconsistency.

DEFINITION 1.4. Given a CLDS system $\mathcal{S} = \langle \langle \mathcal{L}_P, \mathcal{L}_L \rangle, \mathcal{A}_S, \mathcal{R}_S \rangle$, a *proof* is a pair $\langle \mathcal{P}, m \rangle$, where $\mathcal{P}$ is a sequence of configurations $\{\mathcal{C}_0, \ldots, \mathcal{C}_n\}$, with $n > 0$, and m is a mapping from the set $\{0, \ldots, n-1\}$ to $\mathcal{R}_S$ such that for each i, $0 \leq i < n$, $\mathcal{C}_i / \mathcal{C}_{i+1} \in m(i)$.

Table 1.2. Rules for R-literals

$\dfrac{\mathcal{C}\langle\Delta, \overline{\Delta}\rangle}{\mathcal{C}'\langle\alpha:\lambda\rangle}$ $(\perp\mathcal{E})$	$\dfrac{\begin{array}{c}\mathcal{C}\langle[\ \overline{\Delta}\ ]\rangle \\ \vdots \\ \tilde{\mathcal{C}}\langle\perp:\lambda\rangle\end{array}}{\mathcal{C}'\langle\Delta\rangle}$ $(R\mathcal{I})$
$\dfrac{\mathcal{C}}{\mathcal{C}'}$ $(C\text{-R})$ where $\mathcal{C}' \subseteq \mathcal{C}$	$\dfrac{\mathcal{C}}{\mathcal{C}'\langle\Delta\rangle}$ $(R\text{-A})$ if $\mathcal{A} \cup \mathcal{D} \vdash_{FOL} \Delta$

DEFINITION 1.5. Given a CLDS system $\mathcal{S}$, and two configurations $\mathcal{C}$ and $\mathcal{C}'$, $\mathcal{C}'$ is *derivable* from $\mathcal{C}$ in $\mathcal{S}$, written $\mathcal{C} \vdash_{\mathrm{S}} \mathcal{C}'$, if there exists a proof $\langle\{\mathcal{C}, \ldots, \mathcal{C}'\}, m\rangle$.

The above definition of derivability can be reformulated as a relation between a configuration and single declarative unit or R-literal. Given, for instance a CLDS $\mathcal{S}$, a configuration $\mathcal{C} = \langle\mathcal{D}, \mathcal{F}\rangle$ and a declarative unit or R-literal π, then $\mathcal{C} \vdash_{\mathrm{S}} \pi$ if there exists a configuration $\mathcal{C}'$ such that $\mathcal{C} \vdash_{\mathrm{S}} \mathcal{C}'$ and $\pi \in \mathcal{C}'$.

2.3 Semantics

A CLDS can be seen as a "semi-translated" approach to a given object logic, because of its feature of syntactically representing in the proof system semantic notions of the underlying logic. Its own model-theoretic semantics is therefore naturally defined in terms of a first-order semantics using a specific "compiled" translation method into first-order logic. This translation method is defined here and the notions of a CLDS model, satisfiability of a configuration and semantic entailment are given also in terms of classical semantics.

As mentioned before, a declarative unit $\alpha:\lambda$ represents that the formula α is verified (or holds) at the point λ, whose interpretation is strictly related to the type of underlying logic. In the CLDS approach these notions are expressed in terms of first-order statements of the form $[\alpha]^*(\lambda)$, where $[\alpha]^*$ is a predicate symbol. The *extended labelling language* $Mon(\mathcal{L}_P, \mathcal{L}_L)$ is an extension of the language $Func(\mathcal{L}_P, \mathcal{L}_L)$ given by adding a monadic predicate symbol $[\alpha]^*$ for each wff α of $\mathcal{L}_P$.

DEFINITION 1.6. Let $Func(\mathcal{L}_P, \mathcal{L}_L)$ be a semi-extended labelling language. Let $\alpha_1, \ldots, \alpha_n, \ldots$, be the ordered set of wffs of $\mathcal{L}_P$. The *extended labelling language* $Mon(\mathcal{L}_P, \mathcal{L}_L)$ is defined as the language $Func(\mathcal{L}_P, \mathcal{L}_L)$ extended with the set of unary predicate symbols $\{[\alpha_1]^*, \ldots, [\alpha_n]^*, \ldots\}$.

The relationships between the monadic predicates $[\alpha]^*$ in $Mon(\mathcal{L}_P, \mathcal{L}_L)$ are constrained by a set of first-order axiom schemas which capture the satisfiability conditions of each type of formula α. These schemas strictly depend on the underlying logic. For example, as shown in Section 4, the extended labelling algebra $\mathcal{A}^+$ of the AC_{CLDS} system includes the axiom schema $\forall x([A \Longrightarrow B]^*(x) \rightarrow \forall y, z(R_B(y, z) \rightarrow R_A(y, z)))$ to capture the semantic meaning of the $\Longrightarrow$ operator.

The translation method adopted by the CLDS approach associates syntactical expressions of the CLDS language with sentences of the language $Mon(\mathcal{L}_P, \mathcal{L}_L)$ and CLDS configurations with first-order theories in $Mon(\mathcal{L}_P, \mathcal{L}_L)$. Each declarative unit $\alpha : \lambda$ is associated with the sentence $[\alpha]^*(\lambda)$, and each R-literal is associated with itself. The first-order translation of a configuration is a first-order theory defined as follows.

DEFINITION 1.7. Let $\mathcal{C} = \langle \mathcal{D}, \mathcal{F} \rangle$ be a configuration. The *first-order translation* of $\mathcal{C}$, written $FOT(\mathcal{C})$, is a theory written in $Mon(\mathcal{L}_P, \mathcal{L}_L)$ and defined by the expression $FOT(\mathcal{C}) = \mathcal{D} \cup \mathcal{DU}$, where $\mathcal{DU} = \{[\alpha]^*(\lambda) \mid \alpha \in \mathcal{F}(\lambda),\ \lambda$ is a ground term of $Func(\mathcal{L}_P, \mathcal{L}_L)\}$.

Since labels in a configuration can only be ground terms of the language $Func(\mathcal{L}_P, \mathcal{L}_L)$, the first-order translation of a configuration is a set of *ground literals* of the language $Mon(\mathcal{L}_P, \mathcal{L}_L)$. The notions of model, satisfiability and semantic entailment are given in terms of classical semantics (where "$\mathcal{M} \Vdash_{FOL} \psi$" signifies that the classical formula ψ is true in the classical model $\mathcal{M}$, according to the standard classical logic definition).

DEFINITION 1.8. Given a CLDS system $\mathcal{S}$, the associated extended algebra $\mathcal{A}_S^+$, a declarative unit $\alpha : \lambda$ and an R–literal Δ,

(1.1) $\mathcal{M}$ is a CLDS model of $S \quad \Leftrightarrow_{\text{def}} \quad \mathcal{M}$ is a model of $\mathcal{A}_S^+$

(1.2) $\mathcal{M} \Vdash_S \alpha : \lambda \quad \Leftrightarrow_{\text{def}} \quad \mathcal{M} \Vdash_{FOL} [\alpha]^*(\lambda)$

(1.3) $\mathcal{M} \Vdash_S \Delta \quad \Leftrightarrow_{\text{def}} \quad \mathcal{M} \Vdash_{FOL} \Delta$

In the above definition, the statement (1.1) defines the class of models of a CLDS system $\mathcal{S}$ in terms of models of the extended algebra $\mathcal{A}_S^+$ associated with $\mathcal{S}$. Statements (1.2) and (1.3) define, respectively, the satisfiability of declarative units and R–literals in terms of classical satisfiability of their associated first–order translations. A CLDS model $\mathcal{M}$ satisfies a configuration

$\mathcal{C}$, written $\mathcal{M} \parallel\!\!\sim_S \mathcal{C}$, if and only if for each $\pi \in \mathcal{C}$ (where π may be a declarative unit or an R–literal), $\mathcal{M} \parallel\!\!\sim_S \pi$. The notion of semantic entailment in a CLDS system is given here as a relation between configurations.

DEFINITION 1.9. Let $\mathcal{S} = \langle\langle \mathcal{L}_P, \mathcal{L}_L, \rangle, \mathcal{A}_S, \mathcal{R}_S\rangle$ be a CLDS and let $\mathcal{A}^+$ be the extended algebra of $\mathcal{S}$. Let $\mathcal{C} = \langle \mathcal{D}, \mathcal{F}\rangle$ and $\mathcal{C}^{'} = \langle \mathcal{D}^{'}, \mathcal{F}^{'}\rangle$ be two configurations of $\mathcal{S}$ and $FOT(\mathcal{C}) = \mathcal{D} \cup \mathcal{DU}$ and $FOT(\mathcal{C}^{'}) = \mathcal{D}^{'} \cup \mathcal{DU}^{'}$ be their respective first-order translations. The configuration $\mathcal{C}$ semantically entails $\mathcal{C}^{'}$, written $\mathcal{C} \models_S \mathcal{C}^{'}$, iff for each $\Delta \in \mathcal{D}^{'}$, $\mathcal{A}^+ \cup FOT(\mathcal{C}) \models_{FOL} \Delta$, and for each $[\alpha]^*(\lambda) \in \mathcal{DU}^{'}$, $\mathcal{A}^+ \cup FOT(\mathcal{C}) \models_{FOL} [\alpha]^*(\lambda)$.

Proving soundness. Given that the semantics is based on a first-order translation method, the proof of the soundness property of the $\vdash_S$ for a CLDS system $\mathcal{S}$ takes advantage of the soundness property of the first-order classical derivability relation $\vdash_{FOL}$. A diagrammatic representation of the soundness theorem of a CLDS system $\mathcal{S}$ is given in Figure 1.1. The

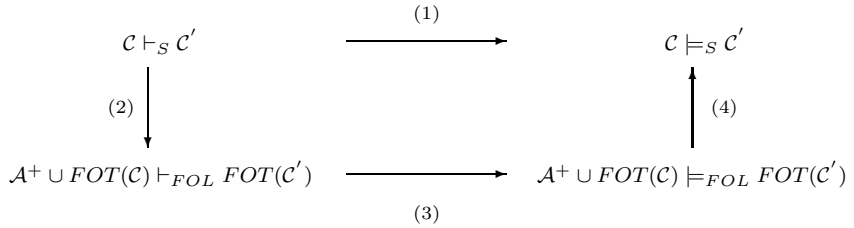

Figure 1.1. Proof of the soundness property of a CLDS system $\mathcal{S}$.

soundness statement, which corresponds to the arrow labelled with (1), is proved by the composition of three main steps, arrows (2), (3) and (4) respectively. The first step (arrow (2)) proves that the hypothesis, $\mathcal{C} \vdash_S \mathcal{C}^{'}$, for a CLDS system $\mathcal{S}$, implies that $\mathcal{A}^+ \cup FOT(\mathcal{C}) \vdash_{FOL} FOT(\mathcal{C}^{'})$. This trivially implies (by soundness of first-order logic) that $\mathcal{A}^+ \cup FOT(\mathcal{C}) \models_{FOL} FOT(\mathcal{C}^{'})$, which gives the second step of the proof (arrow (3)). Arrow (4) is given by the definition of the semantic entailment between configurations given in Definition 1.9. Note that this methodology is generally applicable to any CLDS system. The first step is the only one that needs to be proved for each specific CLDS system.

Proving completeness. The completeness property of a CLDS system with respect to the semantics described in Section 2.3 can be proved using standard Henkin-style methodology [21]. The theorem states that, given a

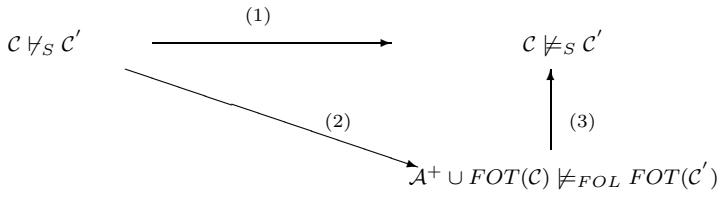

Figure 1.2. Proof of the completeness property of a CLDS system $\mathcal{S}$.

CLDS system and two configurations $\mathcal{C}$ and $\mathcal{C}'$ such that $\mathcal{C}' - \mathcal{C}$ is finite, if $\mathcal{C}'$ is semantically entailed from $\mathcal{C}$ then $\mathcal{C}'$ is also derived from $\mathcal{C}$, where $\mathcal{C}' - \mathcal{C}$ (formally defined in [26]) is the set of declarative units and R-literals in $\mathcal{C}'$ but not in $\mathcal{C}$. The methodology adopted to prove the completeness of a CLDS system is diagrammatically represented in Figure 1.2 and can be informally described as follows.

The proof considers the contrapositive statement (arrow (1)), which states that, given a CLDS system and two configurations $\mathcal{C}$ and $\mathcal{C}'$ such that $\mathcal{C}' - \mathcal{C}$ is a finite, if $\mathcal{C} \nvdash_{\mathrm{S}} \mathcal{C}'$ then $\mathcal{C} \not\models_{\mathrm{S}} \mathcal{C}'$. This is proved by the composition of two main steps, arrows (2) and (3). Arrow (3) is already given by Definition 1.9, while arrow (2) represents the main part of the theorem. The proof of arrow (2) is based on the statement *if $\mathcal{C}$ is a consistent configuration then $\mathcal{C}$ is satisfiable*, known as the "Model Existence Lemma". It consists of the following reasoning steps.

- The hypothesis that $\mathcal{C}'$ is not derivable from $\mathcal{C}$, $\mathcal{C} \nvdash_{\mathrm{S}} \mathcal{C}'$, implies that there exists a $\pi \in \mathcal{C}' - \mathcal{C}$ (where π is a declarative unit or an R-literal) such that $\mathcal{C} \nvdash_{\mathrm{S}} \pi$.

- The above step implies that the configuration $\mathcal{C}$ extended with $\neg\pi$ (written $\mathcal{C} + [\neg\pi]$) is a consistent configuration.

- The above step implies then that the configuration $\mathcal{C} + [\neg\pi]$ is satisfiable. Therefore, there exists a semantic structure $\mathcal{M}$ of the CLDS system $\mathcal{S}$ which satisfies $\mathcal{C}$ and that also satisfies $\neg\pi$. It is then shown that $\mathcal{M}$ does not satisfy π. Thus, since $\pi \in \mathcal{C}'$, by definition of satisfiability of a configuration, $\mathcal{M}$ does not satisfy $\mathcal{C}'$. Hence $\mathcal{A}^+ \cup FOT(\mathcal{C}) \not\models_{\mathrm{FOL}} FOT(\mathcal{C}')$.

The above methods for proving soundness and completeness of a CLDS will be used in Section 5 to show the soundness and compleetness of the access control system AC_{CLDS}.

3 Overview of a logic for Access Control

Access control logics are used to formalize access rights in distributed systems. A collection of such authorization rights is called an *Access Control Policy* and individual rules of the collection are sometimes called *Policy Norms* [13], or more often referred to as policies [22, 23]. A large volume of work is present in the literature on the specification of access control policies, especially in the area of *Role Based Access Control* (RBAC) [1, 27]. These include logic-based declarative descriptions like Abadi's Calculus for Access Control [1, 24] and the logic programming approaches proposed in [2, 4], as well as specialised programming language type descriptions, such as the PONDER [16] and Tower languages [20]. Each method has its benefits and limitations. Logic-based specifications of access control policies, although sometime difficult to understand for the software engineer, are precise, generally succinct and amenable to formal analysis for inconsistency detection [22]. On the other hand, the programming style approaches to RBAC can be quite verbose, but are more accessible to system managers and to those responsable for specifying such policies. The work described in this chapter falls into the first category of access control policy specification. It provides a new formalisation, called AC_{CLDS} system, of the logic for access control initially proposed by Abadi in [1] and subsequently enriched by Massacciwith a tableaux method [24], which overcomes some of its current limitations. Before describing the AC_{CLDS} system in detail, an informal introduction of the main features and expressiveness of this logic is given below, together with an illustrative example.

In RBAC, specifications of policies make use of the notions of a *principal* (or *role*) and an associated set of *privileges* that grant holders of that role access to data or to a system. A principal may also represent a group; members of the group have then the privileges granted to the group. Formulas include atoms, representing satisfied requests, compound expressions stating the making of a request by a principal, relationships between principals and any Boolean combinations of these. For example, the information that a belongs to the manager-role A can be expressed as $a \Longrightarrow A$, the statements that a is requesting to read a file f and B is a sub-manager role of A can be, respectively, formalised as a `says` $read(f)$ and $A \Longrightarrow B$. Primitive operators can be defined to form compound principals which often imply combined privileges of different roles. For instance, the information that a role combines two different roles B and C can be formalised using a binary operator &. This combination of roles is often used to express particular rights of holders of a role B that can be kept private and not inherited. The Access Control Logic includes also the binary operator | used to state that a principal is a member of more than one different group or role and can

make requests assuming any of these roles.

A simple example application drawn from a typical workplace organisation is given here to illustrate the formalisation of some of the basic features of a RBAC. This example consists of four different roles called $\{A, B, C, D\}$, where A could be seen as the role of charge-nurse, B and C as the role of staff nurse responsible for medications and patient welfare respectively, and D as the role of nurse. The privileges of a nurse are inherited by staff nurses, who also have some additional privileges, and the charge-nurse inherits all the privileges of staff nurses. This role dependency is expressed by the formulas $B \Longrightarrow D$, $C \Longrightarrow D$, $A \Longrightarrow C$ and $A \Longrightarrow B$, where $\Longrightarrow$ indicates the role hierarchy and is graphically represented in Figure 1.3(a). It is easy to show that $A \Longrightarrow B\&C$ follows from $A \Longrightarrow C$ and $A \Longrightarrow B$; that is, A has the union of the privileges of B and C.

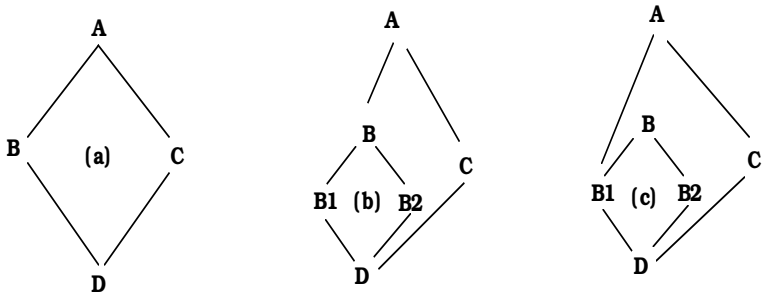

Figure 1.3. (a) Ward-nurse hierarchy, (b) privileges divided, (c) privileges restricted

In Abadi'slogic for access control, access authorization is given by a formula of the form $(A \texttt{ says } r) \rightarrow r$, meaning that if A makes a request r then the request will be granted. This is short-handed as $A \texttt{ controls } r$. The role structure given in Figure 1.3(a), together with the assumption $B \texttt{ controls } r$ would allow the inference of $A \texttt{ controls } r$. Similarly if $C \texttt{ controls } r$. The above scenario could be elaborated by splitting B into two sub-roles $B1$ and $B2$, for example, to encompass the fact that members of staff-nurse of role B may access records of two different kinds. This new dependency, illustrated in Figure 1.3(b), can be expressed by the formulas $B \Longrightarrow B1\&B2$, $B1 \Longrightarrow D$ and $B2 \Longrightarrow D$ instead of $B \Longrightarrow D$. If now $B2 \texttt{ controls } r$ then it can be shown that $A \texttt{ controls } r$ also. In fact, since all members of $B2$ have authorization to access r and A inherits all the privileges of $B2$ through B, all members of A have authorization to access

r as well. It may be, however, desirable for A to inherit only the privileges of role $B1$ (for example if the role $B2$ is for staff-nurse members who communicate only between themselves). In that case, the privilege inheritance, shown graphically in Figure 1.3(c), can be formalised by replacing $A \Longrightarrow B$ with $A \Longrightarrow B1$. Given this new statement, it is no longer possible to infer A `controls` r.

Another aspect of access control systems concerns *Separation of Duty* policies [20]. Such a policy is useful, for example, when a role allows a member two types of access privileges for accessing the same data, but not both. For instance, members of the staff nurse roles might be allowed to issue a medication, but only if the request has been certified, which must be done by some other staff nurse. This kind of access control policy can be captured by the following formalization, where m is a medication: B `controls` $issue(m)$, B `controls` $certify(m)$, $b \Longrightarrow B$ and b `says` $issue(m) \rightarrow \neg b$ `says` $certify(m)$)

4 AC_{CLDS} – A CLDS for Access Control

In this section a CLDS for access control, called AC_{CLDS}, is formally defined following the approach described in Section 2.

Language of AC_{CLDS} . The language of the AC_{CLDS} system is a pair $\langle \mathcal{L}_{\mathcal{P}}, \mathcal{L}_L \rangle$, where $\mathcal{L}_{\mathcal{P}}$ and $\mathcal{L}_L$ share a common set of terms, called *principals*. These include a countable set of constants $\{P, Q, \ldots\}$, called *primitive principals*, and compound terms, called *compound principals*, constructed from the primitive principals using the operators $\{|, \&\}$. In the remainder of this chapter, meta-variables A, B, C will be used to denote principals. The labelling language $\mathcal{L}_L$ is a first-order language constructed from a countable set of constant symbols $\{w_0, w_1, \ldots\}$, representing possible worlds, a countable set of variables $\{x, y, z, \ldots\}$, the binary functors q_B^A for every principal A and B, and binary predicates R_A, for every principal A. The functor q_B^A is a skolem function used in the labelling algebra to express a property of the binary relation $R_{A|B}$. The second component of the AC_{CLDS} language, $\mathcal{L}_{\mathcal{P}}$, is a propositional multi-modal language consisting of a countable set of proposition letters $\{r, s, t, \ldots\}$, representing *access requests*, the constant symbol $\bot$, the classical connectives $\{\neg, \wedge, \rightarrow\}$, the binary operator $\Longrightarrow$ and the modality operators A `says` for every principal A. Well-formed formulae of $\mathcal{L}_{\mathcal{P}}$ are defined as follows.

DEFINITION 1.10. A well-formed formula of $\mathcal{L}_{\mathcal{P}}$ is defined inductively as:

- a propositional letter
- $A \Longrightarrow B$, for any principals A and B

- A `says` α, for any principal A and wff α
- boolean conbinations of wffs

The language $\mathcal{L}_L$ is extended into $Func(\mathcal{L}_P, \mathcal{L}_L)$ by adding for each wff α of $\mathcal{L}_\mathcal{P}$ and principals A and B, the unary functor s^A_α and the constant symbols $sp^1_{A,B}$ and $sp^2_{A,B}$. The ground term $s^A_\alpha(\lambda)$ can be thought of as referring to any *arbitrary* world specifically associated with α and A, and used to express Kripke semantic notions of the form "for all possible worlds..." corresponding to a given formula A `says` α. The constant symbols $sp^1_{A,B}$ and $sp^2_{A,B}$ can also be thought of as *arbitrary* worlds and are used to capture the definition of the $\Longrightarrow$ operator within the proof rule.

The labelling algebra $\mathcal{A}_L$ is a set of first-order axioms expressing the properties of the binary relations R_A, for any principal A.

DEFINITION 1.11. The labelling algebra $\mathcal{A}_L$, written in $Func(\mathcal{L}_P, \mathcal{L}_L)$, is the first-order theory given by the following axioms:

$$\forall x, y(R_{A\&B}(x,y) \leftrightarrow (R_A(x,y) \vee R_B(x,y))) \qquad \text{(Union)}$$
$$\forall x, y(R_{A|B}(x,y) \rightarrow R_A(x, q^A_B(x,y)) \wedge R_B(q^A_B(x,y), y)) \qquad \text{(QuotingE)}$$
$$\forall x, y, z((R_A(x,z) \wedge R_B(z,y)) \rightarrow R_{A|B}(x,y)) \qquad \text{(QuotingI)}$$

An AC_{CLDS} system $\mathcal{L}$ is defined by the tuple $\mathcal{L} = \langle\langle \mathcal{L}_P, \mathcal{L}_L\rangle, \mathcal{A}_L, \mathcal{R}_L\rangle$ where the set $\mathcal{R}_L$ of inference rules is given in the following section.

Table 1.3. Additional Rules for the AC_{CLDS} system

$$\frac{\mathcal{C}\langle A \texttt{ says } \alpha : \lambda_1,\ R_A(\lambda_1, \lambda_2)\rangle}{\mathcal{C}'\langle \alpha : \lambda_2\rangle} \quad (\texttt{ says } \mathcal{E}) \qquad \frac{\begin{array}{c}\mathcal{C}\langle[R_A(\lambda, s^A_\alpha(\lambda))]\rangle \\ \vdots \\ \tilde{\mathcal{C}}\langle \alpha : s^A_\alpha(\lambda)\rangle\end{array}}{\mathcal{C}'\langle A \texttt{ says } \alpha : \lambda\rangle} \quad (\texttt{ says } \mathcal{I})$$

$$\frac{\mathcal{C}\langle B \Longrightarrow A : \lambda_1, R_A(\lambda_2, \lambda_3)\rangle}{\mathcal{C}'\langle R_B(\lambda_2, \lambda_3)\rangle} \quad (\Longrightarrow\mathcal{E}) \qquad \frac{\begin{array}{c}\mathcal{C}\langle[R_A(sp^1_{A,B}, sp^2_{A,B})]\rangle \\ \vdots \\ \tilde{\mathcal{C}}\langle R_B(sp^1_{A,B}, sp^2_{A,B})\rangle\end{array}}{\mathcal{C}'\langle B \Longrightarrow A : \lambda\rangle} \quad (\Longrightarrow\mathcal{I})$$

$$\frac{\begin{array}{c}\mathcal{C}\langle R_{A\&B}(\lambda_1, \lambda_2)\rangle \\ \mathcal{C}\langle[R_A(\lambda_1, \lambda_2)]\rangle \qquad \mathcal{C}\langle[R_B(\lambda_1, \lambda_2)]\rangle \\ \vdots \qquad\qquad\qquad \vdots \\ \mathcal{C}'\langle\pi\rangle \qquad\qquad \mathcal{C}'\langle\pi\rangle\end{array}}{\mathcal{C}'\langle\pi\rangle} \quad (\&\mathrm{R}\mathcal{E})$$

4.1 Proof Rules for AC_{CLDS}

There are three types of reasoning step that can occur in an AC_{CLDS} system. Those of the first type are "classical" and occur within a particular local theory included in $\mathcal{C}$ and respect standard notions of inference for classical connectives. These are the rules given in Table 1.1. The second type of reasoning step concerns reasoning about the diagram part of a configuration using the properties of the labelling algebra. These are the rules described in Table 1.2 among which the (R-A) rule plays a prominent role. It allows, in particular, the inference of relationships about compound principals. For instance, given a configuration that includes the R-literals $R_A(\lambda_1, \lambda_2)$ and $R_B(\lambda_2, \lambda_3)$, the ($R$-A) would allow the inference of the R-literal $R_{A|B}(\lambda_1, \lambda_3)$ using the axiom (QuotingI) of the labelling algebra.

The third type of reasoning step is about the interaction between different theories in a configuration, and uses the rules given in Table 1.3. In particular, the rules (`says` $\mathcal{I}$) and (`says` $\mathcal{E}$) are similar to standard introduction and elimination rules for the modal $\Box$ operator. The term $s^A_\alpha(\lambda)$ used in the (`says` $\mathcal{I}$) rule refers to an arbitrary possible world uniquely associated with the formula α and accessible from λ via the accessibility relation R_A. The rules ($\Longrightarrow\mathcal{I}$) and ($\Longrightarrow\mathcal{E}$) allow dynamic inference of properties of the diagram part of a given configuration. For instance, the transitive property of an R_P relation can be implicitly introduced in a derivation from $P \Longrightarrow P|P:\lambda_1$ whenever the two R-literals $R_P(\lambda_1, \lambda_2)$ and $R_P(\lambda_2, \lambda_3)$ are given or inferred, as shown in the AC_{CLDS} derivation in Figure 1.4. The rule (&R$\mathcal{E}$) allow the inference process to implictly reason about disjunctions of R-literals, as this type of formulae cannot be directly expressed in a configuration.

$$\dfrac{\dfrac{\mathcal{C}\langle P \Longrightarrow P|P:\lambda_1, R_P(\lambda_1,\lambda_2), R_P(\lambda_2,\lambda_3)\rangle}{\mathcal{C}_1\langle R_{P|P}(\lambda_1,\lambda_3)\rangle}\ (R\text{-A})}{\mathcal{C}_2\langle R_P(\lambda_1,\lambda_3)\rangle}\ (\Longrightarrow\mathcal{E})$$

Assumptions

Figure 1.4. Implicit inference of R_P's transitivity from $P \Longrightarrow P|P:\lambda_1$

Finally, the rules given in Table 1.4 are "short-hand" derived rules, which allow compound principal terms involved in `says` formulas to be reduced to primitive principal terms. The first four of these rules mirror the tableau rules given in [24]. The (*univ*) rule captures exactly the global property of the formula $A \Longrightarrow B$ and corresponds to Massacci's rule $[U_{gr}]$. The (*speaks*) rule is a generalisation of Massacci's rules $[L_{gr}]$ and $\langle R_{gr}\rangle$, in that neither principals A and B need to be restricted to be primitive whereas the ($\neg$*speaks*) rule corresponds to Massacci's $\langle U_{gr}\rangle$ rule but without requiring

the introduction of new arbitrary requests since explicit reference to the accessibility relations can be made in the AC_{CLDS} system. Finally, the (*cons*) rule is a new rule that states the consistency of a principal A. This rule is, in particular, used for the derivation of the so-called " hands-off" axiom [1] (see Figure 1.5), which states that from $\neg(Q \texttt{ says } \bot) : s_0$ can be derived $Q \texttt{ controls } P \Longrightarrow Q$ (i.e. $Q \texttt{ says } (P \Longrightarrow Q) \rightarrow P \Longrightarrow Q) : s_0$.

Table 1.4. A collection of useful derived rules in the AC_{CLDS} system

$\dfrac{\mathcal{C}\langle (A\vert B) \texttt{ says } \alpha : \lambda\rangle}{\mathcal{C}'\langle A \texttt{ says } B \texttt{ says } \alpha) : \lambda\rangle}$	$(quote\mathcal{E})$	$\dfrac{\mathcal{C}\langle A \texttt{ says } B \texttt{ says } \alpha : \lambda\rangle}{\mathcal{C}'\langle (A\vert B) \texttt{ says } \alpha : \lambda\rangle}$	$(quote\mathcal{I})$
$\dfrac{\mathcal{C}\langle (A\&B) \texttt{ says } \alpha : \lambda\rangle}{\mathcal{C}'\langle A \texttt{ says } \alpha : \lambda, B \texttt{ says } \alpha : \lambda\rangle}$	$(\&\mathcal{E})$	$\dfrac{\mathcal{C}\langle A \texttt{ says } \alpha : \lambda, B \texttt{ says } \alpha : \lambda\rangle}{\mathcal{C}'\langle (A\&B) \texttt{ says } \alpha : \lambda\rangle}$	$(\&\mathcal{I})$
$\dfrac{\mathcal{C}\langle A \Longrightarrow B : \lambda_1, A \texttt{ says } \alpha : \lambda_2\rangle}{\mathcal{C}'\langle B \texttt{ says } \alpha : \lambda_2\rangle}$	$(speaks)$	$\dfrac{\mathcal{C}\langle \neg(A \Longrightarrow B) : \lambda\rangle}{\mathcal{C}'\langle R_B(sp^1_{A,B}, sp^2_{A,B}), \neg R_A(sp^1_{A,B}, sp^2_{A,B})\rangle}$	$(\neg speaks)$
$\dfrac{\mathcal{C}\langle A \Longrightarrow B : \lambda_1\rangle}{\mathcal{C}'\langle A \Longrightarrow B : \lambda_2\rangle}$	$(univ)$	$\dfrac{\mathcal{C}\langle \neg(A \texttt{ says } \bot) : \lambda\rangle}{\mathcal{C}'\langle R_A(\lambda, s^A_\bot(\lambda))\rangle}$	$(cons)$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{C}_0\langle \neg(Q \texttt{ says } \bot) : w_0\rangle}\ (\text{assumption})}{\tilde{\mathcal{C}}_0\langle \neg(Q \texttt{ says } \bot) : w_0, [Q \texttt{ says } (P \Longrightarrow Q) : w_0]\rangle}\ (\text{assumption})}{\tilde{\mathcal{C}}_1\langle Q \texttt{ says } (P \Longrightarrow Q) : w_0, R_Q(w_0, s^Q_\bot(w_0))\rangle}\ (cons)}{\tilde{\mathcal{C}}_2\langle P \Longrightarrow Q : s^Q_\bot(w_0)\rangle}\ (\texttt{ says } \mathcal{E})}{\tilde{\mathcal{C}}_3\langle P \Longrightarrow Q : w_0\rangle}\ (univ)$$

$$\mathcal{C}'\langle Q \texttt{ says } (P \Longrightarrow Q) \rightarrow P \Longrightarrow Q : w_0\rangle \quad (\rightarrow\mathcal{I})$$

Figure 1.5. Derivation of $Q \texttt{ controls } P \Longrightarrow Q$

Two example derivations are given in Figures 1.5 and 1.6 showing, respectively, a proof of the "hands-off" axiom and a derivation of the (*cons*). These

$\mathcal{C}_0\langle\neg(A \texttt{ says } \bot):\lambda\rangle$ (Assumption)

$\tilde{\mathcal{C}}_0\langle[\neg R_A(\lambda, s^A_\bot(\lambda))]\rangle$ (Assumption)

$\tilde{\mathcal{C}}_1\langle\neg R_A(\lambda, s^A_\bot(\lambda)), [R_A(\lambda, s^A_\bot(\lambda))]\rangle$ (Assumption)

$\tilde{\mathcal{C}}_2\langle\bot : s^A_\bot(\lambda)\rangle$ $(\bot\mathcal{E})$

$\tilde{\mathcal{C}}_3\langle\neg(A \texttt{ says } \bot):\lambda, A \texttt{ says } \bot : \lambda\rangle$ (says $\mathcal{I}$)

$\tilde{\mathcal{C}}_4\langle\bot:\lambda\rangle$ $(\wedge\mathcal{I})$

$\mathcal{C}'\langle R_A(\lambda, s^A_\bot(\lambda))\rangle$ $(R\mathcal{I})$

Figure 1.6. Derivation of rule *cons*

example derivations make use of the following convenient notation. Declarative units introduced in a proof as temporary assumptions, mainly by introduction rules, are written in square brackets, as they represent temporary assumptions that need to be discharged once the rule has been applied. In the example derivation in Figure 1.5, for instance, the given configuration ($\mathcal{C}_0$) is extended to include the temporary assumption Q says $(P \Longrightarrow Q):w_0$. The configurations $\tilde{\mathcal{C}}_1$, $\tilde{\mathcal{C}}_2$ and $\tilde{\mathcal{C}}_3$ are then infered from $\tilde{\mathcal{C}}_0$ by applying (*cons*), (says $\mathcal{E}$) and (*univ*) respectively. In the last step of the derivation, the configuration $C^{'}$ is derived by the initial configuration C_0 using the $(\neg\mathcal{I})$ rule.

4.2 Semantics for AC_{CLDS}

The semantics of the AC_{CLDS} system is based on the model theoretic semantics defined in Section 2.3 for a general CLDS system. The *extended labelling algebra* $\mathcal{A}^+_{\mathrm{Ac}}$ of the AC_{CLDS} system is formally given below.

DEFINITION 1.12.

Given the extended labelling language $Mon(\mathcal{L}_P, \mathcal{L}_L)$ and the labelling algebra $\mathcal{A}_{\mathrm{Ac}}$ of the AC_{CLDS} system, the *extended algebra* $\mathcal{A}^+_{\mathrm{Ac}}$ is the theory in $Mon(\mathcal{L}_P, \mathcal{L}_L)$ given by the labelling algebra extended with the following axiom schemas (Ax1)-(Ax7). For any wffs α and β of $\mathcal{L}_P$ and principals A and B:

$$\forall x([\alpha \wedge \beta]^*(x) \leftrightarrow ([\alpha]^*(x) \wedge [\beta]^*(x))) \quad \text{(Ax1)}$$

$$\forall x([\neg\alpha]^*(x) \leftrightarrow \neg[\alpha]^*(x)) \quad \text{(Ax2)}$$

$$\forall x([\alpha \rightarrow \beta]^*(x) \leftrightarrow ([\alpha]^*(x) \rightarrow [\beta]^*(x))) \quad \text{(Ax3)}$$

$$\forall x((R_A(x, s^A_\alpha(x)) \rightarrow [\alpha]^*(s^A_\alpha(x))) \rightarrow [A \texttt{ says } \alpha]^*(x)) \quad \text{(Ax4)}$$

$$\forall x([A \texttt{ says } \alpha]^*(x) \rightarrow (\forall y(R_A(x, y) \rightarrow [\alpha]^*(y))) \quad \text{(Ax5)}$$

$$\forall x((R_B(sp^1_{A,B}, sp^2_{A,B}) \rightarrow R_A(sp^1_{A,B}, sp^2_{A,B})) \rightarrow [A \Longrightarrow B]^*(x)) \quad \text{(Ax6)}$$

$$\forall x([A \Longrightarrow B]^*(x) \rightarrow (\forall y, z(R_B(y, z) \rightarrow R_A(y, z))) \quad \text{(Ax7)}$$

The first three axiom schemas express the distributive properties of the logical connectives among the monadic predicates of $Mon(\mathcal{L}_P, \mathcal{L}_L)$. The axiom schemas (Ax4) and (Ax5) reflect the traditional Kripke semantic definition of satisfiability of modal $\Box$. The axiom schemas (Ax6) and (Ax7) together express the *specific* semantic meaning of the operator $\Longrightarrow$ for the logic for access control. These axioms facilitate the inference of dynamic properties of the accessibility relations (*i.e.* by simply using the semantic definition of the formula $P \Longrightarrow P|P$), as illustrated in the first-order proof derivation given in Figure 1.7. This derivation clearly shows this main feature of the AC_{CLDS} system.

$$\dfrac{\dfrac{\dfrac{[P \Longrightarrow P|P]^*(\lambda_1)}{\forall x, y(R_{P|P}(x, y) \rightarrow R_P(x, y))}\ \text{(Ax7)}}{\forall x, y, z(R_P(x, y) \wedge R_P(y, z) \rightarrow R_{P|P}(x, z))}\ \text{(QuotingI)}}{\forall x, y, z(R_P(x, y) \wedge R_P(y, z) \rightarrow R_P(x, z))}\ (\rightarrow\mathcal{I})$$

(Assumption)

Figure 1.7. First-order derivation of transitivity of R_P from $P \Longrightarrow P|P : \lambda_1$

The notions of satisfiability and semantic entailment of the AC_{CLDS} system, denoted with $\models_{\text{Ac}}$, are as specified in Definitions 1.8 and 1.9, but based on the extended algebra $\mathcal{A}^+_{\text{Ac}}$.

5 Soundness and Completeness Properties

The soundness and completeness proofs given in this section are based respectively on the two methodologies described in Section 2.3.

5.1 Soundness of AC_{CLDS}

As described in Section 2.3 it is sufficient to show Lemma 1.13, called "Soundness Lemma". The proof of this lemma is by induction on the size of

the derivation of $\mathcal{C}'$ from $\mathcal{C}$, and makes use of the following property. Given any first-order theory T and formula ϕ:

(1.4) If $T \vdash \phi$ then $T \vdash T \cup \{\phi\}$.

Each derivation rule has an associated *size*. The rule (C-R) has size equal to 0. Rules such as (`says` $\mathcal{E}$) or ($\neg\neg$), in which no assumptions are made, have size equal to 1. Rules in which a single assumption is made, such as ($\rightarrow\mathcal{I}$) or ($\Longrightarrow\mathcal{I}$), have size equal to 1 plus the size of the smallest possible derivation that could be made for the antecedent sub-proof. The rule (&R$\mathcal{E}$) has size equal to 1 plus the maximum of the sizes of the smallest possible derivations that could be made for the two antecedent sub-proofs.

LEMMA 1.13. *Let $\mathcal{C}$ and $\mathcal{C}'$ be two configurations. If $\mathcal{C} \vdash_{AC} \mathcal{C}'$ then $\mathcal{A}^{+}_{Ac} \cup FOT(\mathcal{C}) \vdash FOT(\mathcal{C}')$,*

Proof. Let $\langle \mathcal{C}_0, \ldots, \mathcal{C}_n = \mathcal{C}' \rangle$ be a derivation of $\mathcal{C}'$ from $\mathcal{C}$ of size k, $k \geq 0$. The proof is by induction on the length of the derivation.
Base Case: The length of the derivation is $k = 0$. Then the derivation consists only of (C-R) steps. For each i, $0 < i \leq n$, $FOT(\mathcal{C}_i) \subseteq FOT(\mathcal{C}_{i-1})$ and hence, by the transitivity of $\subseteq$, $FOT(\mathcal{C}_n) \subseteq FOT(\mathcal{C}_0)$ and hence by first-order logic $FOT(\mathcal{C}_0) \vdash FOT(\mathcal{C}_n)$.
Inductive Hypothesis: Assume that if $\mathcal{C} \vdash_{AC} \mathcal{C}''$ has a derivation of size less than k then $\mathcal{A}^{+}_{\mathrm{Ac}} \cup FOT(\mathcal{C}) \vdash FOT(\mathcal{C}'')$.
Inductive Step: The length of the derivation is $k > 0$. Then the proof is by cases on the last derivation step between $\mathcal{C}_{n-1}$ and $\mathcal{C}_n$. For each case it is shown that $FOT(\mathcal{C}_{n-1}) \vdash FOT(\mathcal{C}_n)$. Then by the induction hypothesis, since the size of the derivation $\mathcal{C}_0 \vdash_{\mathrm{AC}} \mathcal{C}_{n-1}$ is $< k$, $FOT(\mathcal{C}_0) \vdash FOT(\mathcal{C}_{n-1})$ and hence, by the transitivity of $\vdash$, $FOT(\mathcal{C}_0) \vdash FOT(\mathcal{C}_n)$.
Case ($\wedge\mathcal{E}$): By assumption $[\alpha \wedge \beta]^*(\lambda) \in FOT(\mathcal{C}_{n-1})$. By the ($\forall\mathcal{E}$) and ($\wedge\mathcal{E}$) natural deduction rules of first-order logic $[\alpha]^*(\lambda)$ and $[\beta]^*(\lambda)$ are derivable from Axiom (Ax1). Therefore, by property (1.4), $FOT(\mathcal{C}_{n-1}) \vdash FOT(\mathcal{C}_n)$.
Case ($\Longrightarrow\mathcal{E}$): By assumption $[A \Longrightarrow B]^*(\lambda) \in FOT(\mathcal{C}_{n-1})$ and $R_B(\lambda', \lambda'') \in FOT(\mathcal{C}_{n-1})$. By first-order natural deduction $R_A(\lambda', \lambda'')$ is derivable using Axiom (Ax7). Therefore, $FOT(\mathcal{C}_{n-1}) \vdash FOT(\mathcal{C}_n)$.

The cases for the rules of ($\rightarrow\mathcal{E}$), ($\wedge\mathcal{I}$), ($\neg\neg$), ($\perp\mathcal{E}$), (R-A) and (`says` $\mathcal{E}$) are similar to the above cases.
Case ($\rightarrow\mathcal{I}$): By assumption there is a sub-derivation of $\tilde{\mathcal{C}}\langle \beta : \lambda \rangle$ from $\mathcal{C}\langle [\alpha : \lambda] \rangle$ with size $< k$. Therefore, by the induction hypothesis there is a derivation of $FOT(\mathcal{C}) \cup \{[\beta]^*(\lambda)\}$ from $FOT(\mathcal{C}) \cup \{[\alpha]^*(\lambda)\}$ and hence by the ($\rightarrow\mathcal{I}$) natural deduction rule of first-order logic and property (1.4), there is a derivation of $FOT(\mathcal{C}) \cup \{[\alpha \rightarrow \beta]^*(\lambda)\}$.

Case ($\texttt{says}\,\mathcal{I}$*)*: By assumption there is a sub-derivation of $\tilde{\mathcal{C}}\langle\alpha : s^A_\alpha(\lambda)\rangle$ from $\mathcal{C}\langle[R_A(\lambda, s^A_\alpha(\lambda))]\rangle$ with size $< k$. Therefore, by the induction hypothesis there is a derivation of $FOT(\mathcal{C}) \cup \{[\alpha]^*(s^A_\alpha(\lambda))\}$ from $FOT(\mathcal{C}) \cup \{R_A(\lambda, s^A_\alpha(\lambda))\}$ and hence by the $(\rightarrow\mathcal{I})$ natural deduction rule of first-order logic and property (1.4) there is a derivation of $FOT(\mathcal{C}) \cup \{[A\ \texttt{says}\ \alpha]^*(\lambda)\}$. *Case (*$\&\mathrm{R}\mathcal{E}$*)*: By assumption there is a sub-derivation of $\tilde{\mathcal{C}}\langle\pi\rangle$ from each of $\mathcal{C}\langle[R_A(\lambda_1, \lambda_2)]\rangle$ and $\mathcal{C}\langle[R_B(\lambda_1, \lambda_2)]\rangle$ both with size $< k$. Therefore, by the induction hypothesis there are two derivations of $FOT(\mathcal{C}) \cup \{FOT(\pi)\}$ from $FOT(\mathcal{C}) \cup \{R_A(\lambda_1, \lambda_2)\}$ and from $FOT(\mathcal{C}) \cup \{R_B(\lambda_1, \lambda_2)\}$. Hence by the $(\rightarrow\mathcal{I})$ and $(\vee\mathcal{E})$ natural deduction rules of first-order logic and property (1.4) there is a derivation of $FOT(\mathcal{C}) \cup \{FOT(\Psi)\}$.

The cases for the rules of $(\neg\mathcal{I})$, $(\Longrightarrow\mathcal{I})$ and $(R\mathcal{I})$ are similar to the above cases. ■

Using the method described in Section 2.3, the Soundness Lemma allows to prove the soundness of AC_{CLDS}:

THEOREM 1.14. *Let $\mathcal{C}$ and $\mathcal{C}'$ be two configurations of the AC_{CLDS} system. If $\mathcal{C} \vdash_{AC} \mathcal{C}'$, then $\mathcal{C} \models_{AC} \mathcal{C}'$*

5.2 Completeness of AC_{CLDS}

The Completeness proof of the AC_{CLDS} system follows the steps of the completeness proof for a general CLDS outlined in Section 2.3. It makes use of a main lemma, called the *Model Existence Lemma* whose proof uses the results given in Propositions 1.15, 1.16 and 1.17. All the following proofs make use of an additional notation. Let $\mathcal{C} = \langle\mathcal{D}, \mathcal{F}\rangle$ be a configuration and π be a declarative unit or a R-literal. If π is of the form $\alpha : \lambda$, $\mathcal{C} + [\alpha : \lambda]$ is the configuration $\langle\mathcal{D}, \mathcal{F}'\rangle$, such that $\mathcal{F}'(\lambda) = \mathcal{F}(\lambda) \cup \{\alpha\}$ and for any λ' different from λ, $\mathcal{F}'(\lambda') = \mathcal{F}(\lambda')$. If π is an R-literal Δ, then $\mathcal{C} + [\Delta]$ is the configuration $\langle\mathcal{D}', \mathcal{F}\rangle$ such that $\mathcal{D}' = \mathcal{D} \cup \{\Delta\}$.

PROPOSITION 1.15 (Consistency). *Let $\mathcal{C}$ be an AC_{CLDS} configuration, $\alpha : \lambda$ be an arbitrary declarative unit and Δ be an arbitrary R-literal. If $\mathcal{C} \nvdash_{AC} \alpha : \lambda$ then $\mathcal{C} + [\neg\alpha : \lambda] \nvdash_{AC} \bot : \lambda'$. Similarly, if $\mathcal{C} \nvdash_{AC} \Delta$ then $\mathcal{C} + [\overline{\Delta}] \nvdash_{AC} \bot : \lambda'$.*

Proof. Suppose by contradiction that $\mathcal{C} + [\neg\alpha : \lambda] \vdash_{\mathrm{AC}} \bot : \lambda'$. Then by rule $(\neg\mathcal{I})$ $\mathcal{C} \vdash_{\mathrm{AC}} \alpha : \lambda$, which contradicts the hypothesis. Similarly, assume that $\mathcal{C} + [\overline{\Delta}] \vdash_{\mathrm{AC}} \bot : \lambda'$. Then by rule $(R\mathcal{I})$ $\mathcal{C} \vdash_{\mathrm{AC}} \Delta$, which contradicts the hypothesis. ■

A consistent configuration $\mathcal{C}$ can be extended into a *maximal consistent configuration* $\mathcal{M}_{\mathrm{ccc}}$ by enumerating every possible formula of AC_{CLDS} and, starting from $\mathcal{C}$ add each formula in the enumeration, depending whether

it is consistent or not. It is easy to prove that by construction a $\mathcal{M}_{ccc}$ is a consistent configuration (*i.e.* $\mathcal{M}_{ccc} \nvdash_{\mathrm{AC}} \bot : \lambda$ for any label λ).

The following two Propositions 1.16 and 1.17 prove that an $\mathcal{M}_{ccc}$ is maximal and satisfies various semantic properties on the accessibility relation and logical operators.

PROPOSITION 1.16 (Maximality of $\mathcal{M}_{ccc}$). *Let $\mathcal{M}_{ccc}$ be a maximal consistent configuration. For any declarative unit $\alpha : \lambda$ and R-literal Δ:*

(1.5) *If $\alpha : \lambda \in \mathcal{M}_{ccc}$ then $\neg\alpha : \lambda \notin \mathcal{M}_{ccc}$, and if $\Delta \in \mathcal{M}_{ccc}$ then $\overline{\Delta} \notin \mathcal{M}_{ccc}$.*

(1.6) *Either $\alpha : \lambda \in \mathcal{M}_{ccc}$ or $\neg\alpha : \lambda \in \mathcal{M}_{ccc}$, and either $\Delta \in \mathcal{M}_{ccc}$ or $\overline{\Delta} \in \mathcal{M}_{ccc}$.*

Proof.
Property (1.5): Suppose by contradiction that $\neg\alpha : \lambda \in \mathcal{M}_{ccc}$. Then $\mathcal{M}_{ccc} \vdash_{\mathrm{AC}} \bot : \lambda$ by $(\wedge\mathcal{I})$, which contradicts the fact that $\mathcal{M}_{ccc}$ is consistent. Similarly for any R-literal Δ and its conjugate.
Property (1.6): Suppose by contradiction that $\alpha : \lambda \notin \mathcal{M}_{ccc}$ and $\neg\alpha : \lambda \notin \mathcal{M}_{ccc}$. By construction, there exist configurations $\mathcal{C}' \subseteq \mathcal{M}_{ccc}$ and $\mathcal{C}'' \subseteq \mathcal{M}_{ccc}$ such that $\mathcal{C}' + [\alpha : \lambda] \vdash_{\mathrm{AC}} \bot : \lambda$ and $\mathcal{C}'' + [\neg\alpha : \lambda] \vdash_{\mathrm{AC}} \bot : \lambda$. By monotonicity, $\mathcal{M}_{ccc} + [\alpha : \lambda] \vdash_{\mathrm{AC}} \bot : \lambda$ and $\mathcal{M}_{ccc} + [\neg\alpha : \lambda] \vdash_{\mathrm{AC}} \bot : \lambda$. Therefore, by $(\neg\mathcal{I})$, $\mathcal{M}_{ccc} \vdash_{\mathrm{AC}} \alpha : \lambda$ and $\mathcal{M}_{ccc} \vdash_{\mathrm{AC}} \neg\alpha : \lambda$ and hence $\mathcal{M}_{ccc} \vdash_{\mathrm{AC}} \bot : \lambda$, which contradicts the fact that $\mathcal{M}_{ccc}$ is consistent. Similarly for Δ and its conjugate. ■

PROPOSITION 1.17 (Properties of $\mathcal{M}_{ccc}$). *Let $\mathcal{M}_{ccc}$ be a maximal consistent configuration. The following properties are satisfied, for any labels λ, λ', λ'', wff α and principals A and B:*

(1.7) *$R_{A\&B}(\lambda, \lambda') \in \mathcal{M}_{ccc}$ if, and only if, $R_A(\lambda, \lambda') \in \mathcal{M}_{ccc}$ or $R_B(\lambda, \lambda') \in \mathcal{M}_{ccc}$*

(1.8) *if $R_A(\lambda, \lambda'') \in \mathcal{M}_{ccc}$ and $R_B(\lambda'', \lambda') \in \mathcal{M}_{ccc}$ then $R_{A|B}(\lambda, \lambda') \in \mathcal{M}_{ccc}$*

(1.9) *if $R_{A|B}(\lambda, \lambda') \in \mathcal{M}_{ccc}$ then $R_A(\lambda, q_B^A(\lambda, \lambda')) \in \mathcal{M}_{ccc}$ and $R_B(q_B^A(\lambda, \lambda'), \lambda') \in \mathcal{M}_{ccc}$*

(1.10) *if A `says` $\alpha : \lambda \in \mathcal{M}_{ccc}$ and $R_A(\lambda, \lambda') \in \mathcal{M}_{ccc}$ then $\alpha : \lambda' \in \mathcal{M}_{ccc}$*

(1.11) *if $\neg(A$ `says` $\alpha) : \lambda \in \mathcal{M}_{ccc}$ then $R_A(\lambda, s_\alpha^A(\lambda)) \in \mathcal{M}_{ccc}$ and $\neg\alpha : s_\alpha^A(\lambda) \in \mathcal{M}_{ccc}$*

(1.12) *if* $A \Longrightarrow B{:}\lambda \in \mathcal{M}_{ccc}$ *and* $R_B(\lambda^{'}, \lambda^{''}) \in \mathcal{M}_{ccc}$ *then* $R_A((\lambda^{'}, \lambda^{''}) \in \mathcal{M}_{ccc}$

(1.13) *if* $\neg(A \Longrightarrow B){:}\lambda \in \mathcal{M}_{ccc}$ *then* $R_B(sp^1_{A,B}, sp^2_{A,B}) \in \mathcal{M}_{ccc}$ *and* $\neg R_A(sp^1_{A,B}, sp^2_{A,B}) \in \mathcal{M}_{ccc}$

(1.14) $\alpha \wedge \beta{:}\lambda \in \mathcal{M}_{ccc}$ *if, and only if,* $\alpha{:}\lambda \in \mathcal{M}_{ccc}$ *and* $\beta{:}\lambda \in \mathcal{M}_{ccc}$

(1.15) $\alpha \rightarrow \beta{:}\lambda \in \mathcal{M}_{ccc}$ *if, and only if, either* $\neg\alpha{:}\lambda \in \mathcal{M}_{ccc}$ *or* $\beta{:}\lambda \in \mathcal{M}_{ccc}$

Proof.
Property (1.7): (*Only if:*) Suppose by contradiction that $R_{A\&B}(\lambda, \lambda^{'}) \in \mathcal{M}_{\text{ccc}}$, $R_A(\lambda, \lambda^{'}) \notin \mathcal{M}_{\text{ccc}}$ and $R_B(\lambda, \lambda^{'}) \notin \mathcal{M}_{\text{ccc}}$. By Property (1.6) the literals $\neg R_A(\lambda, \lambda^{'}) \in \mathcal{M}_{\text{ccc}}$ and $\neg R_B(\lambda, \lambda^{'}) \in \mathcal{M}_{\text{ccc}}$. By ($R$-A) rule $\mathcal{M}_{\text{ccc}} \vdash_{\text{AC}} \neg R_{A\&B}(\lambda, \lambda^{'})$ which together with the assumption gives a contradiction. (*If:*) A similar argument can be applied in this case, using the assumptions either $R_A(\lambda, \lambda^{'}) \in \mathcal{M}_{\text{ccc}}$ or $R_B(\lambda, \lambda^{'}) \in \mathcal{M}_{\text{ccc}}$, and $R_{A\&B}(\lambda, \lambda^{'}) \notin \mathcal{M}_{\text{ccc}}$.

The proofs for Properties (1.8), (1.9), (1.10), (1.12) and (1.14) are similar to the proof for Property (1.7).
Property (1.11): Suppose by contradiction that $\neg(A \texttt{ says } \alpha){:}\lambda \in \mathcal{M}_{\text{ccc}}$ and $R_A(\lambda, s^A_\alpha(\lambda)) \notin \mathcal{M}_{\text{ccc}}$. Therefore, by Property (1.6) $\neg R_A(\lambda, s^A_\alpha(\lambda)) \in \mathcal{M}_{\text{ccc}}$. Using the rules ($\texttt{says}\,\mathcal{I}$), ($\bot\mathcal{E}$) and ($\wedge\mathcal{I}$) it can be shown that $\mathcal{M}_{\text{ccc}} \vdash_{\text{AC}} \bot{:}\lambda$. On the other hand, suppose for contradiction that $\neg(A \texttt{ says } \alpha){:}\lambda \in \mathcal{M}_{\text{ccc}}$ and $\neg\alpha{:}s^A_\alpha(\lambda) \notin \mathcal{M}_{\text{ccc}}$. Therefore, by Property (1.6) $\alpha{:}s^A_\alpha(\lambda) \in \mathcal{M}_{\text{ccc}}$. Using the rules ($\texttt{says}\,\mathcal{I}$) and ($\wedge\mathcal{I}$), it can be shown that $\mathcal{M}_{\text{ccc}} \vdash_{\text{AC}} \bot{:}\lambda$.
Property (1.13): Suppose by contradiction that $\neg(A \Longrightarrow B) : \lambda \in \mathcal{M}_{\text{ccc}}$ and $R_B(sp^1_{A,B}, sp^2_{A,B}) \notin \mathcal{M}_{\text{ccc}}$. Therefore, by Property (1.6) the literal $\neg R_B(sp^1_{A,B}, sp^2_{A,B}) \in \mathcal{M}_{\text{ccc}}$. Using the derived rule ($\neg speaks$) it can be shown that $\mathcal{M}_{\text{ccc}} \vdash_{\text{AC}} R_B(sp^1_{A,B}, sp^2_{A,B})$. The case for $\neg R_A(sp^1_{A,B}, sp^2_{A,B})$ is similar.
Property (1.15): (*Only If:*) Suppose by contradiction that $\alpha \rightarrow \beta{:}\lambda \in \mathcal{M}_{\text{ccc}}$, $\neg\alpha : \lambda \notin \mathcal{M}_{\text{ccc}}$ and $\beta : \lambda \notin \mathcal{M}_{\text{ccc}}$. By Property (1.6) $\alpha : \lambda \in \mathcal{M}_{\text{ccc}}$ and $\neg\beta{:}\lambda \in \mathcal{M}_{\text{ccc}}$. By the ($\rightarrow\mathcal{E}$) and ($\wedge\mathcal{I}$) rules $\mathcal{M}_{\text{ccc}} \vdash_{\text{AC}} \bot{:}\lambda$. (*If:*) Suppose for contradiction that $\neg\alpha{:}\lambda \in \mathcal{M}_{\text{ccc}}$ and $(\alpha \rightarrow \beta){:}\lambda \notin \mathcal{M}_{\text{ccc}}$. By Property 1.6 $neg(\alpha \rightarrow \beta) : \lambda \in \mathcal{M}_{\text{ccc}}$. Using the rules ($\rightarrow\mathcal{I}$), ($\neg\mathcal{I}$) and ($\neg\neg$) it can be shown that $\mathcal{M}_{\text{ccc}} \vdash_{\text{AC}} \bot{:}\lambda$. The case for $\beta{:}\lambda$ is similar. ■

The following Model Existence Lemma shows how to construct a AC_{CLDS} model for a maximal consistent configuration $\mathcal{M}_{\text{ccc}}$ obtained from a initial consistent configuration $\mathcal{C}$.

LEMMA 1.18 (Model Existence Lemma). *Let $\mathcal{M}_{ccc}$ be an AC_{CLDS} maximal consistent configuration. There exists an AC_{CLDS} model M that satisfies $\mathcal{M}_{ccc}$.*

Proof. According to Definition 1.8 an AC_{CLDS} model is a first-order interpretation that satisfies the labelling algebra of AC_{CLDS} and the semantic axioms of AC_{CLDS}, and it satisfies the $\mathcal{M}_{\mathrm{ccc}}$ if and only if (by definition) it satisfies the $FOT(\mathcal{M}_{\mathrm{ccc}})$. Such an interpretation M can be constructed as follows.

- The domain of M is the set of ground labels in $Func(\mathcal{L}_P, \mathcal{L}_L)$.
- For each predicate $[\alpha]^*$in the extended labelling language $Mon(\mathcal{L}_P, \mathcal{L}_L)$, the interpretation $M([\alpha]^*)$ is given by the set $\{\lambda | \alpha : \lambda \in \mathcal{M}_{\mathrm{ccc}}\}$.
- For each R-literal R_A in $Mon(\mathcal{L}_P, \mathcal{L}_L)$, the interpretation $M(R_A)$ is given by the set $\{(\lambda, \lambda^{'}) | R_A(\lambda, \lambda^{'}) \in \mathcal{M}_{\mathrm{ccc}}\}$.

It is shown below that the interpretation M satisfies the three axioms of the labelling algebra and the seven semantic axioms (Ax1) - (Ax7) and it is therefore an AC_{CLDS} model. It can then be easily shown that, by construction, M satisfies the $FOT(\mathcal{M}_{\mathrm{ccc}})$.
Case (Union): Let λ and λ' be elements of $Func(\mathcal{L}_P, \mathcal{L}_L)$, the domain of M. (*If*:) Assume that $(\lambda, \lambda') \in M(R_A)$ or $(\lambda, \lambda') \in M(R_B)$. Then by construction either $R_A(\lambda, \lambda') \in \mathcal{M}_{\mathrm{ccc}}$ or $R_B(\lambda, \lambda')) \in \mathcal{M}_{\mathrm{ccc}}$. In the first case, by Property (1.7) of Lemma 1.17, $R_{A\&B}(\lambda, \lambda') \in \mathcal{M}_{\mathrm{ccc}}$ and hence $(\lambda, \lambda') \in M(R_{A\&B})$ Similarly for the second case. (*Only if*:) Assume that $(\lambda, \lambda') \in M(R_{A\&B})$, then $R_{A\&B}(\lambda, \lambda') \in \mathcal{M}_{\mathrm{ccc}}$. By Property (1.7) of Lemma 1.17 either $R_A(\lambda, \lambda') \in \mathcal{M}_{\mathrm{ccc}}$ or $R_B(\lambda, \lambda')) \in \mathcal{M}_{\mathrm{ccc}}$. Hence, by construction, $(\lambda, \lambda') \in M(R_A)$ or $(\lambda, \lambda') \in M(R_B)$.

The proof for cases (QuotingE) and (QuotingI) are similar to the case for (Union).
Case (Ax2): Let λ be an arbitrary element of $Func(\mathcal{L}_P, \mathcal{L}_L)$. (*Only if*:) Assume $\lambda \in M([\neg\alpha]^*)$, then $\neg\alpha : \lambda \in \mathcal{M}_{\mathrm{ccc}}$ and by Property (1.6) of Lemma 1.16 $\alpha : \lambda \notin \mathcal{M}_{\mathrm{ccc}}$ and $\lambda \notin M([\alpha]^*)$. (*If*:) The proof is similar..

The proofs for (Ax1) and (Ax3) are similar to the proof for (Ax2).
Case (Ax4): Let λ be an arbitrary element of $Func(\mathcal{L}_P, \mathcal{L}_L)$. Assume $\lambda \notin M([A \texttt{ says } \alpha]^*)$, then by construction $A \texttt{ says } \alpha : \lambda \notin \mathcal{M}_{\mathrm{ccc}}$ and by Property (1.11) of Lemma 1.17 $R_A(\lambda, s^A_\alpha(\lambda)) \in \mathcal{M}_{\mathrm{ccc}}$ and $\alpha : s^A_\alpha(\lambda)) \notin \mathcal{M}_{\mathrm{ccc}}$. Therefore $(\lambda, s^A_\alpha(\lambda)) \in M(R_A)$ and $s^A_\alpha(\lambda) \notin M([\alpha]^*)$, and hence M satisfies $\neg(R_A(\lambda, s^A_\alpha(\lambda)) \rightarrow [\alpha]^*(s^A_\alpha(\lambda)))$.
Case (Ax7): Let λ be an arbitrary ground term of $Func(\mathcal{L}_P, \mathcal{L}_L)$ and assume that $\lambda \in M([A \Longrightarrow B]^*)$. Hence $A \Longrightarrow B : \lambda \in \mathcal{M}_{\mathrm{ccc}}$. Suppose

for arbitrary $\lambda^{'}$ and $\lambda^{''}$ that $(\lambda^{'}, \lambda^{''}) \in M(R_B)$ and hence by construction $R_B(\lambda^{'}, \lambda^{''}) \in \mathcal{M}_{\text{ccc}}$. By Property (1.12) of Lemma 1.17 $R_A(\lambda^{'}, \lambda^{''}) \in \mathcal{M}_{\text{ccc}}$ and hence $(\lambda^{'}, \lambda^{''}) \in M(R_A)$.

The proofs for cases (Ax5) and (Ax6) are similar to the proofs for Cases (Ax4) and (Ax7). ■

Using the result of Lemma 1.18 and following the steps given in Section 2.3 the completeness of the AC_{CLDS} system with respect to its semantics is proved.

6 Discussion

This paper introduces the AC_{CLDS} system as a logic for access control and proves its soundness and completeness. This system is more flexible than the logic in [24] in the following ways. It can allow reasoning about formulae of the form $P \Longrightarrow Q$ where neither P nor Q is restricted to the atomic case, and reasoning about implicit dynamic properties on the accessibility relations. These are facilitated by the use of a labelling algebra that allows explicit inferences about the accessibility relation of compound principals, which can only be done implicitly in Massacci's tableau system. As a consequence the AC_{CLDS} proof system is complete with respect to the semantics given in Section 2.3. This semantics captures the same semantic principles of Massacci's logic, for which his tableau system is only partially complete [24]. A formal correspondence between Massacci's logic and the AC_{CLDS} system could be proved following the methodology described in [7, 11]. This would require to show that the derivability relation in Massacci's system is equivalent to the AC_{CLDS} derivability relation. Informally, this would require to prove the following two steps. If a closed tableau in his system exists for a formula $1 : \neg\phi$, then the declarative unit $\phi : w_0$, for any arbitrary world w_0, can be shown to be derivable from an empty configuration. And, if there is an open branch in the complete tableau for $1 : \neg\phi$, then it can be shown that there exists an AC_{CLDS} model where $\phi : w_0$ is false.

In both [1] and [24] the definition of the `controls` operator causes peculiarities in the reasoning process, namely that "if a principal A is making a request r on which it is trusted, then every principal will be trusted on r". This arises from their definition of the formula A `controls` r as a shorthand for A `says` $r \rightarrow r$. In fact, assuming A `controls` r (read as A can be trusted on r) and A `says` r (read as A makes the request r), it can be proved that B `controls` r for any principal B as follows. From the definition of `controls` the two assumptions give r, which implies B `controls` r for any B. Clearly, this is a rather unsatisfactory result and should not be the case for such a plausible scenario when B has no relationship to A.

The underlying problem is that in the definition of `controls` the consequence that r gets granted is unrelated to the principal making the request. This could be addressed by, for instance, redefining A `controls` r either as A `says` $r \rightarrow \Diamond(A)r$, or as A `says` $r(A) \rightarrow r(A)$. In the first case it would require the accessibility relation for each principal to satisfy the seriality property, whereas the second case is not so restrictive, but would require the parameter of the modality to be a ground term in the language, in a similar way as in Fitting's Term Modal Logics [17]. The current AC_{CLDS} system can be extended to include either of these solutions.

An additional peculiarity with the definition of `controls` is regarding the concept of denial of access. Assuming denial of access by a principal B on request r to be expressed by $\neg(B$ `controls` $r)$, an inconsistency would immediately arise whenever a second unrelated principal A makes a request r on which it is trusted. Clearly this is an unexpected behaviour of an access control system. The above suggested solutions would also stop this problem from occurring and allow the concept of negative authorisation to be expressed using classical negation Other approaches, in which the concepts of control and denial are taken as independent primitives, don't seem to suffer from these problems [4, 13, 22, 23].

An additional benefit of the AC_{CLDS} system is its first-order translation-based semantics, which provides already a "semi-compiled" formalisation of the multi-modal logic into first-order logic. This semi-compiled theory (*i.e.* the extended labelling algebra) can be used to develop a first-order automated theorem prover for the AC_{CLDS} system. The following example illustrates this process. Consider the derivation from $\neg(Q$ `says` $\bot) : w_0$ of Q `says` $(P \Longrightarrow Q) \rightarrow P \Longrightarrow Q$ given in Figure 1.5. The OTTER theorem prover [25] can be used to show the same derivation by means of resolution. Resolution is a refutation technique and requires the conclusion to be negated, which, after translation gives the first-order sentences $\neg[Q$ `says` $\bot]^*(w_0)$, $[Q$ `says` $(P \Longrightarrow Q)]^*(w_0)$ and $\neg[P \Longrightarrow Q]^*(w_0)$.

Using the extended labelling algebra the instantiations of the axiom schema needed for this proof are the following:

$$\forall x([P \Longrightarrow Q]^*(x) \rightarrow \forall y, z(R_Q(y,z) \rightarrow R_P(y,z))) \quad \text{(Ax 7)}$$
$$\forall x, y([Q \texttt{ says } (P \Longrightarrow Q)]^*(x) \rightarrow \forall y(R_Q(x,y) \rightarrow [P \Longrightarrow Q]^*(y))) \quad \text{(Ax 5)}$$
$$\forall x((R_Q(sp^1_{P,Q}, sp^2_{P,Q}) \rightarrow R_P(sp^1_{P,Q}, sp^2_{P,Q})) \rightarrow [P \Longrightarrow Q]^*(x)) \quad \text{(Ax 6)}$$
$$\forall x((R_Q(x, s^Q_\bot(x)) \rightarrow [\bot]^*(s^Q_\bot(x))) \rightarrow [Q \texttt{ says } \bot]^*(x)) \quad \text{(Ax 4)}$$

In order to use OTTER, some further notational changes are needed; some terms and predicates are introduced as a short-hand, P and Q are replaced by p and q and variables x, y and z are replaced by X, Y and Z and are assumed to be universally quantified.

1. $\neg says(q, b1, w0)$	data
2. $says(q, s1(p, q), w0)$	negation of goal(i)
3. $\neg s2(p, q, w0)$	negation of goal(ii)
4. $(s2(p, q, X) \wedge R(q, Y, Z)) \rightarrow R(p, Y, Z)$	instance of (Ax 7)
5. $(says(q, s1(p, q), X) \wedge R(q, X, Y)) \rightarrow s2(p, q, Y)$	instance of (Ax 5)
6. $(R(q, sp1, sp2) \rightarrow R(p, sp1, sp2)) \rightarrow s2(p, q, X)$	instance of (Ax 6)
7. $(R(q, X, s(X)) \rightarrow b2(s(X))) \rightarrow says(q, b1, X)$	instance of (Ax 4)

Figure 1.8. Compiled data required for an automatic derivation

```
list(sos).
1 [] -says(q,b1,w0).
2 [] says(q,s1(p,q),w0).
3 [] -s2(p,q,w0).
4 [] -s2(p,q,X)| -R(q,Y,Z)|R(p,Y,Z).
5 [] -says(q,s1(p,q),X)| -R(q,X,Y)| s2(p,q,Y).
6 [] s2(p,q,X)|R(q,sp1,sp2).
7 [] s2(p,q,X)| -R(p,sp1,sp2).
8 [] says(q,b1,X)|R(q,X,s(X)).
9 [] says(q,b1,X)|-b2(s(X)).
end_of_list.
```

```
1 [] -says(q,b,w0).
2 [] says(q,s1(p,q),w0).
3 [] -s2(p,q,w0).
4 [] -s2(p,q,X)| -R(q,Y,Z)|R(p,Y,Z).
5 [] -says(q,s1(p,q),X)| -R(q,X,Y)| s2(p,q,Y).
6 [] s2(p,q,X)|R(q,sp1,sp2).
7 [] s2(p,q,X)| -R(p,sp1,sp2).
8 [] says(q,b1,X)|R(q,X,s(X)).
11 [binary,6.1,3.1] R(q,sp1,sp2).
12 [binary,7.1,3.1] -R(p,sp1,sp2).
13 [binary,8.1,1.1] R(q,w0,s(w0)).
15 [binary,4.2,11.1,unit_del,12] -s2(p,q,A).
16 [binary,5.1,2.1,unit_del,15] -R(q,w0,A).
17 [binary,16.1,13.1] $F.
```

Figure 1.9. Proof using OTTER

In particular, in the formulation shown in Figure 1.8, the term $s1(p,q)$ stands for $P \Longrightarrow Q$ when it occurs as a sub-formula of Q `says` $(P \Longrightarrow Q)$, while the predicate $s2(p,q,X)$ is shorthand for the atom $[P \Longrightarrow Q]^*(X)$ and the atom $says(q, s1(p,q), Z)$ stands for $[Q$ `says` $(P \Longrightarrow Q)]^*(Z)$. Similarly, the constant $b1$ represents $\bot$ when it occurs as a sub-formula in Q `says` $\bot$, the term $s(X)$ is a short-hand for the Skolem $s^q_\bot(X)$ and the atom $b2(s(X))$ stands for $[\bot]^*(s^Q_\bot(x))$. The constants $sp1$ and $sp2$ are short-hands for the Skolem terms in the instance of (Ax 6). Atoms such as $R(p,X,Y)$ stand for $R_P(x,y)$. More general instances of the axiom schema could have been used, such as $(s2(A,B,X) \wedge R(B,Y,Z)) \rightarrow R(A,Y,Z)$ for the instance of Axiom (Ax 7). After converting the data in Figure 1.8 into clausal form and submitting it to the OTTER theorem prover the refutation shown in Figure 1.9 was returned, which is equivalent to the natural deduction proofs in Figures 1.5 and 1.6.

BIBLIOGRAPHY

[1] M. Abadi, M. Burrows, B. Lampson and G. Plotkin. A calculus for access control in distributed systems. *ACM Trans. on Prog. Lang. and Sys.* 15(4), pp 706-734, 1993.

[2] D. Agrawal, J. Giles, K-W Lee and J. Lobo. Policy Ratification, *Proceedings of IEEE Policy 2005*, IEEE Computer Society, 2005.

[3] A. Artosi, G. Governatori and A. Rotolo. Labelled Tableaux for non-monotonic reasoning: Cumulative consequence relations, *Journal of Logic and Computation*, Oxford University Press, 12(6), pp 1027-1060, 2002.

[4] S. Barker and P.J. Stuckey. Flexible Access Control Policy Specification with Constraint Logic Programming. *ACM Trans. on Inf. and Sys. Security*, 6(4), pp 501-546, 2003.

[5] J. van Bentham. Correspondence Theory, in *Handbook of Philosophical Logic*, Vol II, Reidel, 1986.

[6] P. Blackburn. Internalising Labelled Deduction, *Journal of Logic and Computation*, 10(1), Oxford University Press, pp 137-168, 2000.

[7] K. Broda, M. D'Agostino, and A. Russo. Transformation Methods in LDS in *Logic, Language and Reasoning: An Essay in Honour of Gabbay*, Ed. J.J. Ohlbach and U. Reyle, Kluwer, pp 335-376, 1999.

[8] K. Broda, M. Finger and A. Russo. Labelled natural deduction for substructural logics, *Logic Journal of the IGPL*, 7(3), pp 283-318, 1999.

[9] K. Broda, D. Gabbay, L. C. Lamb and A. Russo. Labelled natural deduction for conditional logics of normality, *Logic Journal of the IGPL*, 10(2), pp 123-163, 2002.

[10] K. Broda and D. Gabbay. Labelled Abduction - Compiled Labelled Abductive Systems, in *Labelled Deduction*, Eds. D. Basin et al, Kluwer Academic Publishers, 2000.

[11] K. Broda, D. M. Gabbay, L. C. Lamb and A. Russo. *Compiled Labelled Deductive Systems: A Uniform Presentation of Non-Classical Logics*, Research Studies Press Ltd, London, 2004.

[12] K. Broda, A. Russo and D. Gabbay. A unified compilation style natural deduction system for modal, substructural and fuzzy logics. In *Discovering World with Fuzzy Logic: Perspectives and Approaches to Formalization of Human-consistent Logical Systems*, Springer, 2000.

[13] L. Cholvy and F. Cuppens. Analyzing Consistency of Security Policies, *Proc. of IEEE Symposium on Security and Privacy, SP-97*, pp 103-112, IEEE Computer Society, 1997.

[14] K. L. Clark, Negation as Failure, In *Logic and Databases*, Ed. H. Gallaire and J. Minker, Plenum Press, 1978.

[15] M. D'Agostino, D. Gabbay and K. Broda. Tableau Methods for Substructural Logics, in *Handbook of Tableau methods*, Eds. M. D'Agostino et al, Kluwer Academic Publishers, pp. 397-467, 1999.

[16] N. Damianou, N. Dulay, E. Lupu and M. Sloman. The Ponder Policy Specification Language, *Proc. of Policy 2001: Workshop on Policies for Distributed Systems and Networks*, LNCS 1995, pp18-39, 2001.

[17] M. Fitting, L. Thalmann and A. Voronkov. Term Modal Logics, *Studio Logica* 69(1), pp 133-169, 2001.

[18] D. Gabbay. *Labelled Deductive Systems: Volume I*, Oxford Logic Guides, Vol. 33, Oxford Clarendon Press, 1996.

[19] D. Gabbay and N. Olivetti. Goal oriented deduction, in D. Gabbay and F. Guenthner, Eds, *Handbook of Philosophical Logic*, Vol. 9, Kluwer Academic Publisher, 2nd Ed. pp 199-285, 2002.

[20] M. Hitchens and V. Varadharajan. On the Design and Specification of Role Based Access Control Policies, *IEEE Process-Software*, Vol. 147(4), pp 117-129, 2000.

[21] G. Hughes and M. Cresswell. *A new introduction to Modal Logics*, Methuen, London, 1996.

[22] H. Kamoda, A. Hayakawa, M. Yamaoka, S. Matsuda, K. Broda, and M. Sloman. Policy Conflict Analysis Using Tableaux for On Demand VPN Framework, *Proc. of the Sixth IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, IEEE Computer Society, pp 565–569, 2005.

[23] E. Lupu and M. Sloman. Conflicts in Policy-Vased Distributed Systems Management. *IEEE Trans. on Software Eng.*, 25(6), pp 852-869, 1999.

[24] F. Massacci, Tableaux Methods for Access Control in Distributed Systems, *TABLEAUX-97*, LNAI 1227, pp 246 -260, 1997.

[25] W. McCune. *Otter 3.3 reference Manual and Guide*, Argonne National Laboratory, Argonne, Illinois, 2003.

[26] A. Russo. *Modal Logics as Labelled Deductive Systems*, PhD Thesis, Dept. of Computing, Imperial College London, 1996.

[27] R.S.Sandhu. The Typed Access Matrix Model. *Proc. of IEEE Symposium on Security and Privacy, SP-92*, IEEE Computer Society, pp 122-136, 1992.

[28] A. Sympson. *The Proof Theory and Semantics of Intuitionistic Modal Logics*, PhD Thesis, University of Edinburgh, 1993.

[29] T. C. Son and J. Lobo. Reasoning about Policies using Logic Programs, *Proc. of American Assoc. for Artificial Intelligence*, pp 210-216, 2001.