UNIVERSITY OF BERGEN
DEPARTMENT OF INFORMATICS

Applications of Access Control Logics to Online Banking Systems

Author: Ragnhild Bratli

Supervisor: Håkon Robbestad Gylterud

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

November 22, 2021

Abstract

Digitalisation has in many ways helped us become more productive and efficient. Technology has transformed the way we do banking. Online banking systems have made banking significantly simpler, and are offered by most modern banks today. Most users are uncritical as to how the functionality of these conform to the expectations. These systems deal with sensitive information, and attackers have a lot to gain by being able to exploit these systems. Because of this, users are dependent on correct and secure systems.

In 2021, broken access control is regarded by OWASP as the most common security risk to web applications. This means that a large number of web applications can be exploited by attackers, by bypassing the intended permissions of users.

In this thesis, we design a specification for an online banking system. These specifications are derived from research on existing systems and testing. Further, we introduce a formal language combining modal logic with first order logic, and introduce an axiomatic system based on this logic, created with the intent of reasoning about the access control of our system. We reason about the correctness of our axiomatic system, and use these axioms to formalize examples relevant to the design specification. Completion of these elements was a continuous process.

Some challenges surfaced regarding domain knowledge, as well as with the use of first order logic for reasoning about continuous elements of the system. We found that modal logic was convenient for allowing us to represent the actions of actors and their corresponding outcomes in the system. We conclude that the project represents a starting point for further development of a standardized approach to formal verification of access control systems.

Acknowledgments

First and foremost, I would like to thank my supervisor, Håkon R. Gylterud, for guidance and moral support during our weekly meetings and throughout my degree. His input and involvement has been invaluable for my work and motivation.

I would also like to thank my family, for their love and support. In particular, I want to thank my siblings, Elisabeth and Erling, for being my biggest supporters, and for setting the bar so high.

Ragnhild Bratli
November 22, 2021

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

ACID	**A**tomicity, **C**onsistency, **I**solation and **D**urability
ACL	**A**ccess **C**ontrol **L**ist
ABAC	**A**ttribute **B**ased **A**ccess **C**ontrol
CBAC	**C**apability **B**ased **A**ccess **C**ontrol
CCS	**C**alculus of **C**ommunicating **S**ystems
CVE	**C**ommon **V**ulnerabilities and **E**xposures
CWB-NC	**C**oncurrency **W**ork**b**ench of **N**orth **C**arolina
DAC	**D**iscretionary **A**ccess **C**ontrol
DBMS	**D**ata**b**ase **M**anagement **S**ystem
FOL	**F**irst **O**rder **L**ogic
MAC	**M**andatory **A**ccess **C**ontrol
OWASP	The **O**pen **W**eb **A**pplication **S**ecurity **P**roject
RBAC	**R**ole **B**ased **A**ccess **C**ontrol

Chapter 1

Introduction

Until recently, banks were dependent on having physical branches where their customers would have to attend to pay their bills, withdraw or deposit cash, amongst other banking services. Although this served its purpose and was a workable solution at the time, banking used to be time consuming, and a rather mundane process. However, in recent years, financial institutions world wide have improved the way these processes take effect [61]. As most other industries, there is considerable potential for digitalisation and more effective solutions. Today, banks are dependent on providing digital systems for their customers in order to survive [13]. Almost all financial institutions now provide online banking services, allowing their customers to not only make payments through electronic payment systems, but also have virtually all other banking service needs met by these systems. During the Covid-19 pandemic, number of consumers making use of digital banking systems increased considerably [33].

When these online banking services started being launched in the late 1990's [13], and as they became more standard, cumbersome methods such as cheque books and payments via post could be phased out. Customers no longer had to physically attend to the bank to pay their bills, and individuals and businesses can in a much greater degree control and monitor their own finances. Efficiency has improved immensely when it comes to banking, which also allows customers to pay less in fees and earn more money through interest rates, as well as allowing a market for banks without physical branches that can offer a higher interest rate on deposits [3].

Until 2007, online banking services were mainly used for payments and account services. Since then, the usability of these services have extended and improved immensely. Today, nearly anything you would previously have to do at a physical branch is now possible through online banking services. Due to its convenience, a vast majority of Norwegian residents now mainly

use online and mobile banking [52].

According to information from Norges Bank, the central bank of Norway, there were 2,557 million card payments and 880 million giro payments in 2019 in Norway alone [5]. This totalled to a value of 900 billion NOK for card payments, and 20.014 billion NOK for giro payments. This makes Norway one of the countries with the most card payments per capita per year. In short, quite large sums of currency are moving around in these systems, distributed on a large amount of transactions.

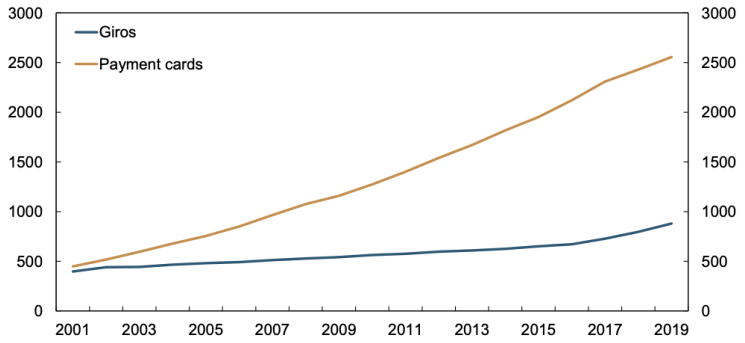


Figure 1.1: Chart of use of payment instruments. In millions of payments. 2001-2019.

Source: Norges Bank

Although digitalisation of banks comes with benefits for consumers, a substantial challenge is that these systems are a favored target of digital attackers [61]. This is due to attackers having a lot to gain from being able to exploit these systems. Because of this, it means that the security of these digital systems is crucial for financial institutions to be able to provide this service to its customers.

1.1 Motivation

The Common Vulnerabilities and Exposures (CVE) database is a database containing information on publicly disclosed cyber security vulnerabilities. When searching the database for the keyword “bank”, just over one hundred records match that search [19]. Not all, but most of these entries refer to vulnerabilities in online and mobile banking systems. Out of all online programs and services we use, the one dealing with personal finances is one we want to be able to trust will behave and function as intended.

As briefly discussed in the previous section, digitalisation in the banking industry has made the process of making payments and transfers significantly simpler. Modern multi-factor authentication solutions, such as the Norwegian BankID, ensure that the process of signing in to these services is considered to be very safe [6]. They do, however, still require a secure access control model and an implementation that conforms to the specifications in order to be secure and behave as intended. For example, secure authentication is of little use in a bank if users are able to access accounts they should not have permission to access. This is in order to protect users, their information and their finances, from attackers looking to exploit. If an attacker was to be able to exploit these systems, it could have severe consequences for both the bank and its customers.

The access control model is what describes how decisions regarding what actions users are allowed to perform are made within a system. This is what allows a user to perform the actions required for their needs, while preventing them from performing actions they should not be able to execute. The Open Web Application Security Project (OWASP) is a nonprofit foundation that works to improve the security of software, and annually release a list of most critical security risks related to web applications [27]. In 2021, broken access control tops the OWASP Top Ten list. This security risk had more occurrences in applications than any other category [26], and means that in many of the applications tested, users could act outside of their intended permissions [25].

In order to avoid vulnerabilities and exploitation, we ensure that systems behave as intended through functional verification. One way of doing so is by simulation and testing. However, this is not an exhaustive process. It is impossible to simulate all possible states in a complex design, and this approach is likely to miss corner case scenarios. Another approach to verification is formal verification, which is a method of applying mathematical reasoning to the verification of design specifications. This approach allows us to explore all possible states, and detect corner case bugs [56].

1.2 Scope

In this section, we clarify what the project covers, and the boundaries of the problem to be investigated.

One security measure of an online bank is taken during authentication. When attempting to sign in to this kind of system, there are two possibilities. If the login fails when a customer

attempts to log in to an account using their login credentials, they are returned to the login page. If the login succeeds, they are presented with their account interface. Here, they have an overview of their own accounts, as well as other accounts they are delegates rights for, and their respective balances. The user is now operating under the identity the login credentials correspond to.

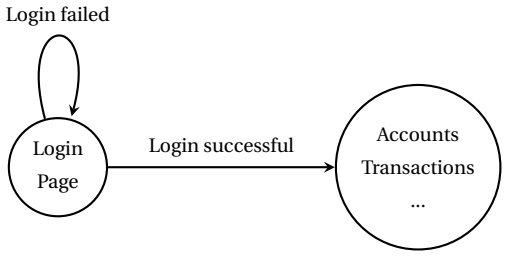


Figure 1.2: Sign in process.

Authentication and central administration are not topics we are concerned with in this project, but a requirement for ensuring safety of customers in actual systems.

When a consumer wants to become a customer in a bank, a user is created for the customer. This user is identified by a unique identification number. For authentication, a user has their own sign in credentials. All customers in a bank is an owner of one or more accounts. When an account is set up, the starting balance is always zero. No account can be created with an initial value for balance – positive or negative. The value of the balance of an account at any given time is the resulting sum of all transactions to and from the account.

Online banks also require a role of bank administrator. The purpose of this role is to have an overview of user behaviours, assist users having issues with their accounts or login, and assess requests needing their approval, which could for example be loan applications, setting up a new account, etc. This role is also a security measure, as they work on detecting unusual behaviour from the users, which could be a result of leaked card information and a security threat.

An owner of an account also has the ability to delegate privileges for the account to another user. This is very relevant to access control, and a concept we want to investigate with our formal logic. Many banks offer the possibility of setting very specific restrictions, such as which payments a user can approve and maximum amount for transfers can also be registered. Details like this are not prioritized in the scope of this study.

The main focus of this thesis is to study how we can use formal logic to represent and reason about the access control of online banking. It is the transaction aspect we are most concerned

with in this project, as this is what is most relevant for the access control. We also cover some examples of accounts with special requirements, such as BSU (young people's housing savings) and accounts with a restricted number of yearly withdrawals.

1.3 Aim and Objectives

Specifying and verifying computer-based systems is a challenging task, and can be solved using a number of approaches. The purpose of this project is to investigate whether modal logic [1] can be applied to express access control specifications for online banks. We wish to gain an understanding of how, and to what extent, we can apply modal logic to reason about the correctness of access control models and how this can improve security. We want to evaluate to what degree using modal logic is successful, and what challenges we face with this approach.

The objectives for this project are to:

1. Investigate the application of formal verification and modal logic to reason about information systems and its security aspects.
2. Investigate the rules and expected behaviour of access control systems for online banks, and specify an access control model for a hypothetical online banking system.
3. Define the syntax and semantics of a modal language that can be used to reason about access control.
4. Using our previously defined components, translate the access control for the hypothetical online banking system we have designed to modal logic to the form of an axiomatic system.
5. Use our modal language to formally reason about functionality of the access control model, and evaluate our approach.

1.4 Related Work

In this section, we will discuss some previous work that is relevant for the key topics of this project. We begin by looking at how formal methods can be useful in the verification of system

[1] Modal logic: formal systems concerned with modalities, elaborated in 2.4.

security. Further, we look at how formal methods have already been used for banking processes. Lastly, we look at some applications of modal logic. Modal logic has a number of applications, many of which are within computer science and information security. Here, we focus on work combining modal logic with access control logic and formal verification.

1.4.1 Formal Models and Security

Formal models for computer security

The pursuit of building secure computer systems has been ongoing for decades. The first proper security program being created in the early 1970s [48]. Already in 1981, Landwehr reviews the need for formal security models in [41]. According to them, in order to design a secure system, it must first be decided upon what the definition of "secure" is in the respective context. It would be meaningless to reason about whether a a particular system is secure without a precise definition of what this entails. The need for formal models is emphasized by arguing that a mathematical model functions as a concise and precise description of the behaviour of the system.

In his paper, the focus is on military security. In these kind of systems, information is assigned different sensitivity levels. One significant reason for this is due to the costs associated with protecting information, as this can often be a great expense. The security requirements for military systems are based on the existence of information that might damage the national security if known by the enemy. This kind of system can be compared to a role based access control (RBAC) model, assigning permissions to users based on their roles. This is explained in further detail in the next chapter. The recognized sensitivity levels in decreasing order of effect on national security are:

- Top secret
- Secret
- Confidential
- Unclassified

Where all levels above unclassified are regarded as classified. Permissions are delegated on a need-to-know basis, where the fundamental idea is that unless an individual has both the required clearance and requires the information as a prerequisite for a job, that individual should not have the information. This idea corresponds to the principle of least privilege, an information security aspect based on giving users the minimum permissions needed to perform their

job functions.

Landewehr further describes various formal structures to model the military security environment. These different models apply different interpretations of the general finite-state machine model, which is the basis for all the presented models. This model views a computer system as a finite set of states along with a transition function to determine what the next state will be, based on the current state and input. With respect to their definitions of security, the models can be divided roughly into three groups:

- Those that are concerned only with controlling direct access to particular objects.
- Those that are concerned with information flows among objects assigned to security classes.
- Those that are concerned with an observer's ability to deduce any information at all about particular variables.

Whether a particular model is appropriate depends on the application for which it is to be used. This is something to be kept in mind when designing an access control model.

Efficient Formal Verification in Banking Processes

Formal verification in banking processes is addressed by Intilangelo et al. in [57]. Here, it is emphasized that model checking is not widely applied partly due to the state explosion problem, which says the state space grows exponentially in the number of concurrent processes. They consider property-based methodology developed to combat the state explosion problem. In their approach, they use Milner's Calculus of Communicating Systems (CCS) presented in [47] for modelling the implementation of the banking process. Further, the properties of the expected behaviours are expressed in selective mu-calculus logic in [7]. Lastly, the results obtained from their approach are validated by comparing them to those previously obtained by Concurrency Workbench of North Carolina (CWB-NC).

The focus of this paper can be divided into two parts. The first part is to show how model checking can be applied in the context of business modelling. The second part is applying a case study of a real-world banking workflow of a loan origination process in order to evaluate and test the property-based methodology.

This approach can prove correctness obtaining a considerable reduction of both state space, size and time compared to CWB-NC. The authors express their belief that the business community, and banking field in particular, can benefit from utilizing formal methods to prevent significant errors.

The paper introduces how model checking can be applied and why it is a useful approach. The experimental results they obtained allowed to verify that applying the CCS model to the loan origination process satisfies the properties formalized in selective mu-calculus. It is concluded that model checking can be used to prove correctness and reliability of systems, and that it is a powerful technique for validation of business processes [57].

A modal logic framework for security policies

Cuppens and Demolombe also pitch into the idea of applying formal logic with the intent of improving security, pointing out the increasing importance of security in [18]. From a model theoretic point of view, they present a modal logic framework that can be used to model non-trivial security policies. This framework combines doxastic [2] and deontic [3] logic.

They also point out that most difficulties when it comes to reasoning about security policies come from the fact that the policies may be incomplete. By incomplete, it is meant that for some data, it is not explicitly stated by the policy whether users have permissions or not to access these data. Another challenge mentioned is that the set of data that have a given security level is not necessarily consistent. It is to overcome these difficulties that the modal logic framework was designed.

The focus here is on the problem of confidentiality of data. The general idea for the definition of a policy is that a user can access some data only if the user's clearance level is greater than or equal to the classification level of this particular data.

Out of convenience, as shown by Denning in [20], confidentiality is composed into two sub-problems. The first problem is to control information flow inside the database management system (DBMS), and the second to control derived information. The framework proposed is general enough to model both of these problems, but in the paper the main focus is the latter. This problem is usually referred to as the inference problem. The approach is a continuation of work presented by earlier in [17], where the same authors present a modal logical framework for reasoning about multilevel security policy.

To start with, they introduce their logical framework and its most important features. Further, they show how this framework can be used to formalize the notions of consistency and completeness for the security policy, as well as different kinds of security constraints to prevent inference problems due to deductive or abductive reasoning.

[2] Doxastic: concerned with belief.
[3] Deontic: concerned with obligation.

In the logical framework they present, it is assumed that the data base content is represented by a consistent set of sentences of a language of propositional calculus. These sentences are not assumed to be true, but are considered to be beliefs. How users interact with the database is represented by modal formulae, where a modality is defined for each individual user. Each modality is represented by a relation, and is assumed to be serial.

Confidentiality policy is what defines what part of the database each individual has permission to access. They provide a formalization of a given confidentiality policy, which is based on the concept of role. For each role it is designed what permissions, obligations and prohibitions the role entails for the holder.

From the semantics, it is shown that the modalities satisfy a set of properties. In order to formally represent a given multilevel security policy, a function which assigns a classification level to some formulas of the language is defined. Finally, consistency of completeness of the multilevel security policy is presented. For each security level, a modality and formula are defined.

The framework they present is proven to be sound and complete given that the language contains a finite number of atomic sentences. They conclude with the idea that combining deontic modalities with action modalities would enable us to consider more general security policies, particularly those combining confidentiality and integrity requirements.

1.4.2 Modal Logic for Access Control

Modal logic has previously been applied to reasoning about RBAC in [39]. Kosiyatrakul, Older and Chin address the importance of making correct access control decisions from a security perspective. In their paper, they introduce the syntax and semantics of a modal language for reasoning about a RBAC model. They further use this logic to describe policies of the access control. By defining the policies of RBAC in their logic, the ability to reason about privileges, authority, credentials, and RBAC is soundly united in a single logic.

A translation from three basic access control logics to classical modal logic has also been presented by Garg and Abadi in [28]. In particular, his translation enabled for studying of definitions and axiomatization of the “speaks for” relation. With the help of existing algorithms and provers for S4[4], these results may serve as the basis for theorem provers for logics of access control.

In their final section they emphasize how difficult it is to ensure the security of information

[4] S4: Axiomatic system for modal logic, elaborated in 2.4.6.

systems, and that in order to reason about access control systems there is a need for a simple, formal and rigorous logic. This logic should be able to account for a variety of structures and situations.

1.4.3 Summary

Using formal methods in verification allow us to verify a system's properties in a thorough manner, and testing alone is not enough to ensure correctness of a system. We can see that there exists some previous work where modal logic is applied to reason about access control system, and for formal verification of these systems, which is what we are hoping to achieve. This methodology is not widely applied in industry, however.

1.5 Outline

- **Chapter 1** presents the motivation for this thesis, problem formulation, aim and objectives, and lastly, a discussion on related work. We also introduce the evolvement of online banking and its need for secure systems.
- **Chapter 2** presents a theoretical background for the relevant subjects addressed throughout the thesis. First, we introduce access control and the various established models. Then, the need for verification and describe the appeal of formal verification. Finally, the different components of modal logic and its applications are presented to give an idea of how this can be used in the verification process.
- **Chapter 3** specifies the methods used in this project. This includes the process of designing an access control model, constructing a formal language, and finally, an analysis. The tools and approaches used are outlined.
- **Chapter 4** presents a design specification for an online bank, including its objects and functionalities.
- **Chapter 5** presents a formal language and axiomatic system to reason about the system described in Chapter 4.
- **Chapter 6** presents an analysis of how our formal language can be used to reason about the access control system.
- **Chapter 7** discusses the findings and concludes the thesis. We consider how successful our approach is, as well as presenting suggestions and ideas for further work.

Chapter 2

Background

In the previous chapter, we introduced banking and its digital evolution, which is a central element for the purpose of this thesis. In this chapter, an introduction to the remaining key subjects that are relevant to the thesis is provided. First, we introduce the purpose of security and access control in digital systems, as well as look into the different models of access control. The following section is about formal verification. We look at what the benefits are of including formal verification in the development process, and the commonly adopted methods. Lastly, we look at first order logic and modal logic, including their key concepts and how they can be used collectively. We also look at applications of modal logic.

2.1 Access Control

Globally, many nations are prioritizing cyber security, and have established national programs in regards to this [22]. The background for these measures has already been introduced in Chapter 1, and the goal is to protect against attacks and unauthorized access. Cyber security in general refers to a collection of tools, policies, concepts, approaches, technologies, etc. that can be used to protect the system and its users' assets [68]. The concept that we are concerned with for this thesis is access control.

In general, access control simply refers to a method of separating permissions. Methods of doing so can be either physical or logical [16]. Access control is central in our everyday lives. A very relevant example of this is that we have locks on the door to our home, which is a physical form of access control. An example of a logical form of access control is that we have a password on our computer. Both of these measures are implemented with the intent of preventing

intruders from accessing something they should not have access to, as this could be a threat for the safety of the owner.

In a digital system, when an actor makes a request to perform a particular action, a decision somehow needs to be made about whether or not to accept and perform this action on the actors behalf. The way this decision is made will depend on the system's implementation of an access control model. Access control is the part of the system which controls authentication and authorization of individual users, referring to which permissions they have and what they have access to. Authentication refers to the verification of the identity of users in a system, while authorization refers to a security mechanism determining permissions for users in a system. A secure access control model is a key component when it comes to designing secure and reliable systems.

Some questions still remain to be answered, especially regarding what makes access control so important. There is a number of reasons why corporations and businesses are concerned about protecting the integrity of its customers. Without secure access control, a digital program will not be usable. Security is not implemented only to satisfy users, so that they will actually continue to utilize their services, but authorities also have strict regulations when it comes to this in order to protect integrity of users. Hacks and data thefts enabled by weak security, and attempts by the provider to cover up or mistakes that could have been avoided have over the years cost large companies in total hundreds of millions of dollars [32]. Of course, these sorts of incidents will also have a severely undesirable effect on the reputation and trustworthiness of a company. It is also worth specifying that the importance of secure access control increases when dealing with sensitive data. The threat is bigger for an online banking application, which retains a lot of sensitive information, than an application where this is not the case.

Types of access control in cyber security

A number of models for access control exist and are implemented in various systems for the purpose of protecting sensitivity of data. The type of model a business prefers will depend on each business' needs. Statically, these models are all equivalent [31]. Dynamically however, they behave differently.

Access control list (ACL)

Systems using ACL for their access control have a list of permissions associated with this system, and specify for each user which permissions they possess from this list. This model is quite commonly used by all mainstream operating systems [34].

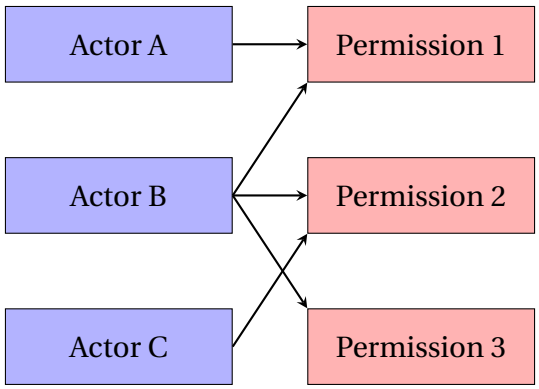


Figure 2.1: Assignment of permissions in ACL.

We see ACL in online banking systems when delegating specific permissions to other users.

Role based access control (RBAC)

This model has already been introduced in Chapter 1. RBAC is a popular model for access control, especially in commercial application [23]. This model assigns actors roles, and grants them individual accesses based on their assigned roles. RBAC also enables the implementation of key security principles such as “principle of least privilege” and “separation of privilege”. These principles imply that users can only access data that is necessary for the role they attain, and have the least amount of privilege required in order to execute their tasks [55].

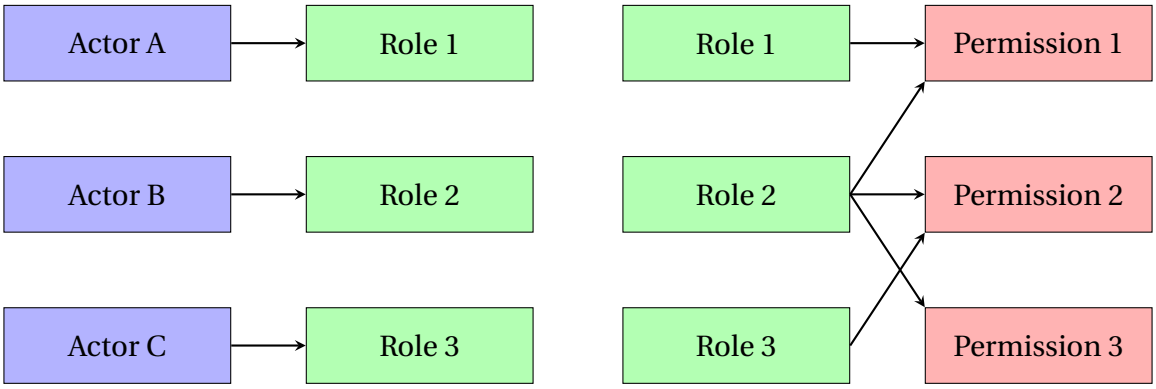


Figure 2.2: Assignment of permissions through role assignment in RBAC.

We see example of RBAC in online banks, where we can assign roles such as account signatory, where users get full access to an account, or non-account signatory, where users can be delegated either full access or view-only access [53].

Mandatory access control (MAC)

Access control policies in MAC are fixed by a central authority [31]. The operating system provides users with access based on data confidentiality and user clearance levels. Permissions in this model are granted on a need to know basis, meaning that users must have a reason for attaining a certain permission. Out of all access control models, MAC is considered to be the most secure. Downsides to MAC include the fact that these systems are challenging to maintain, as manual configuration of security levels and clearances require constant attention from administrators. Additionally, users having to request access to each new piece of data negatively impacts the model's user-friendliness [62].

An example of how MAC is incorporated in online banking is that many actions, such as giving someone power of attorney or terminating a security deposit account, need to go through a central authority [21], which in this case is an employee in the bank.

Discretionary access control (DAC)

With DAC, permissions can be specified for, or access can be transferred to, another user by a subject who has access to an object [31]. The idea is that subjects can determine who has what permissions to their objects [64]. Compared to MAC, DAC is quite a popular model due to the

fact that this model allows for a lot of freedom for its users and does not cause administrative overhead. Despite its advantages, it does also have several considerable limitations. Allowing users to share their data however they like means that this can also include sharing of malicious files. There is also no centralized access management, which means that in order to find out access parameters, one has to inspect each ACL [62].

We see DAC in online banks, such as with the functionality that an owner of an account can give another actor permissions to the account, typically chosen from an ACL with permissions such as viewing or make transfers.

Summary

ACL and RBAC are both examples of permission based access control models. All of the models listed above have their strengths and weaknesses. Which of the models is the best suited will ultimately depend entirely on the application. A combination of these models is also a possibility. For example, systems using DAC usually implement ACLs for allowing users to define permissions on an object. This is typical for a file system, where the owner of a file can assign read or write permissions to others.

2.2 Formal Verification

In order for a digital system to be usable, particularly one that handles sensitive data, it must be ensured that the system functions as intended. This functionality is described in a design specification. Verification of a system is the evaluation of whether the implementation of a system complies with the design specification, and is a critical element in the process of development. Numerous methods exist for this purpose. Formal verification is the process of doing so using formal methods of mathematical reasoning.

It is a straight forward task to prove that something does not hold if that is the case, as this can simply be done by presenting a counter example. It is a greater challenge to formally prove that something actually does hold. In his article on the topic [56], Alok Sanghavi addresses why we should be using the method of formal verification, as opposed to solely relying on simulation to verify. A number of solid arguments are presented, and the main idea is that formal verification gives us the possibility to explore all possible states of a system. When it comes to simulation, this is an exhaustive effort.

In order to perform a formal verification, the design and its specifications need to be simplified to a verifiable format. There are two particularly popular types when it comes to formal verification:

Model checking is a method for the analysis of dynamical systems that can be modeled by a state transition system [15]. This approach involves confirming whether or not the model semantically entails the specification, and is presented in detail in [54].

Deduction based verification is a method based on deductive reasoning, which is when we confirm whether or not the conclusion follows with certainty from the premises [36]. We can check formula satisfiability by decomposing the formula and examining whether it follows from some statements or formulas. Such an approach is presented by Radoslaw Klimek in [37].

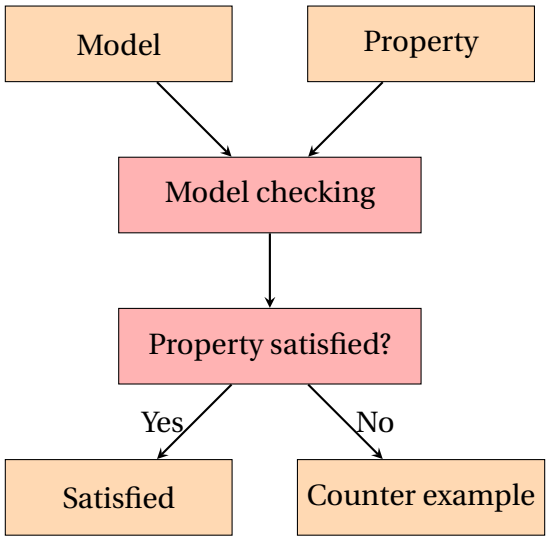


Figure 2.3: Process of model checking.

Klimek also argues in the introduction of his article that software modelling enables us with a better understanding of domain problems, as well as developed systems, through goal oriented abstractions in all phases of software development. He supports the claim made by Sanghavi, that program testing can be used to demonstrate the presence of bugs, but fail to demonstrate their absence.

For this kind of formal approach to verification, there are two main important and closely related parts. Firstly, a formal specification which establishes fundamental system properties. Secondly, a formal verification which is the act of proving correctness of the system.

2.3 First Order Logic

First order logic (FOL), also commonly referred to as predicate logic, is a formal logic that uses quantified variables over formulas [65]. It is an extension of classical propositional logic. Propositional logic is a branch of logic that studies the ways statements can interact with each other. A proposition in this logical language is simply a statement [43]. FOL adds expressiveness to propositional logic. The two main parts to FOL are its syntax and semantics. The syntax determines what symbols form well-formed expressions, and the semantics determines the meaning behind these expressions. FOL is both sound and complete. A system is sound if everything that is provable is true, and complete if everything that is true has a proof [69].

2.3.1 Syntax

The syntax of FOL consists of logical symbols and non-logical symbols. The logical symbols generally include all symbols from propositional logic, such as variables, connectives and other auxiliary symbols [43]. In addition, the FOL syntax has two quantifier symbols:

$\forall$: the universal quantifier, meaning "for all".
$\exists$: the existential quantifier, meaning "for some" or "there exists".

These quantifiers allow us to express properties of classes of objects. Non-logical symbols in FOL are function symbols and predicate symbols.

2.3.2 Semantics

Semantics is what relates the syntax to some context. In first order logic, semantics is given by a domain and an interpretation [67]. An interpretation is a model for the language. The interpretation specifies what objects are in the domain we want to talk about, and assigns a denotation to each non-logical symbol in the language. Whether a formula in FOL is true or not depends on what the interpretation specifies.

2.3.3 Limitations

Although FOL is widely used, both within mathematics and computer science, it does have its limitations. These limitations mainly have to do with its expressiveness. For example, although FOL is complete, it is undecidable [30]. This means that it cannot be proved or disproved. Addi-

tionally, when it comes to modelling natural language, more complicated features are not easily expressed in FOL [49].

2.4 Modal Logic

Modal logic dates back to 1912, when the first systems were being developed by C.I. Lewis. The first systems were presented in 1918. The relational semantics were not developed until the mid 20^{th} century, by Arthur Prior, Jaakko Hintikka, and Saul Kripke [4]. Like FOL, modal logic is an extension of propositional logic, and can be viewed as the logic of modalities.

2.4.1 Syntax

Like first order logic, modal logic extends classical propositional logic, and includes all the syntactic properties found in propositional logic. In addition to these, modal logic has two additional operators known as modal operators, $\Box$ and $\Diamond$. The intended meaning of these operators vary, but traditionally they represent necessity [70], elaborated in 2.4.2. These operators allow modal logic to behave more along the lines of a natural language than many other formal languages, and allow us to reason beyond the idea of all formulas being strictly valuated as true or false.

2.4.2 Necessity

A key concept of modal logic is the concept of necessity. The modal operators let us easily adapt modal logic to areas of computer science, due to the flexibility these operators introduce [40]. Two modal operators are introduced as part of the syntax for modal logic. These operator symbols are $\Box$ and $\Diamond$, translating to 'necessarily' and 'possibly' respectively, such that:

- $\Box\phi$ = necessarily ϕ
- $\Diamond\phi$ = possibly ϕ

Generally, we view possibility as the dual of necessity and vice versa, so $\Diamond\phi$ is equivalent to $\neg\Box\neg\phi$, and $\Box\phi$ is equivalent to $\neg\Diamond\neg\phi$ [66]. The meaning of these modal operators will vary depending on the modality in question, and have often been used for logics including deontic, temporal, and epistemic. These different logics are useful for reasoning about different areas of computer science.

Figure 2.4: Examples of Kripke structures.

The nature of these modal operators must follow a certain set of rules, known as an axiomatic system. Some axioms are very commonly included in this set, such as $\Box\phi \rightarrow \Box\Box\phi$ (4), but the set of rules in its entirety varies. These varieties are what are what lead to different variations of modal logic.

2.4.3 Possible Worlds Semantics

The semantics of modal logic is based on possible worlds semantics. These semantics observe different worlds, which are not necessarily actual worlds, but can represent different states, points in time, etc. A Kripe frame is a pair ⟨W, R⟩ where W represents a set of "worlds" and R represents a binary relation on W, known as an accessibility relation [29]. A Kripke structure is a tripe ⟨W, R, V⟩ consisting of a Kripke frame in addition to a valuation V for all formulas in worlds w ∈ W. ⟨W, R, V⟩ $\models \phi$ means that for every world w ∈ W, V(w,ϕ) valuates to true [60].

In propositional logic, it is the case that for a model $\mathscr{M}$ that either $\mathscr{M} \models \phi$ or $\mathscr{M} \models \neg\phi$ will always hold [63]. This is a tautology. One of the main differences between propositional logic and modal logic is that this is not the case for modal logic. In modal logic, you have the possibility of ϕ holding in one or some worlds but not others.

We also see this in another example of how modal operators are not truth-functional. Although if ϕ is false in a world accessible from w, it follows that $\Box\phi$ is false in w, it does however not follow from ϕ is true in a world accessible from w that $\Box\phi$ is true in w.

If a formula ϕ holds at every world accessible from a world w, then ϕ is necessary with respect to w. If ϕ holds at some world accessible from w, then ϕ is possible with respect to w.

In Figure 2.4 we see two examples of Kripke structures. In the first structure, ϕ is necessary with respect to w_0, as it is the case in all worlds accessible from it, which in this example is w_2 and w_3. In the figure to the right, w_0 is also accessible from itself, and ϕ is not the case in w_0, meaning ϕ is only possible with respect to w_0.

2.4.4 Modalities

Modality is defined as "a particular mode in which something exists or is experienced or expressed" [44]. When it comes to modal logic, we can use this idea to reason about a number of modalities, and the interpretations of the modal operators will depend thereafter. Presented in [29], some commonly adopted modalities and the interpretations of the modal operators based on the modality are explained in the following table:

Modality	$\Box\phi$	$\Diamond\phi$
Alethic - logic of necessity	Necessarily ϕ	Possibly ϕ
Epistemic - logic of certainty	a knows that ϕ	For all a knows, it may be true that ϕ
Temporal - logic of time/tense	It is always the case that ϕ	It is sometimes the case that ϕ
Deontic - logic of obligation	You must do ϕ	You may do ϕ
Doxastic - logic of belief	It is believed that ϕ	At some state, ϕ is believed to be true

Table 2.1: Overview of commonly practiced modalities and respective interpretations of the modal operators.

As presented in the table above, the modal operators have different meanings depending on the context. It is also possible to use more than one modality, resulting in a multi-modal logic. The example of epistemic logic above will typically include an additional subscript, such that $\Box_a\phi$ has the meaning "a knows that ϕ", and similarly for $\Diamond$. Epistemic logic alone is an example of a multi-modal logic. This is elaborated in the following section.

2.4.5 Multimodal Logic

Multimodal logic is a modal logic dealing with more than one modality. Having this possibility can be helpful when reasoning about theory within computer science, and it also has its philosophical applications. There exists infinitely many of these systems and they can have various levels of complexity.

A typical example of a multimodal system is in temporal logics. Temporal logics can be viewed as a multimodal system in the sense that we may want to reason about time by utilizing various distinct modal operators. For example, we might want to talk about the possibility of ϕ in the past or possibility of ϕ in the future. In order to reason about both of these tenses in the same systems, we need two separate modal operators [12].

Another example of multimodal logic is epistemic logic. Since epistemic logic deals with reasoning about knowledge for agents, and we want to be able to keep track of the knowledge of each agent, there will exist one modality for each agent in the domain. This is often done by indexing $\Box$ over the set of agents, such that $\Box_i a$ reads "agent i knows that a is true" [46]. In a Kripke structure, this means that there exists one relation R_i for each agent.

2.4.6 Axiomatic Systems

In mathematics and logic, an axiomatic system is a collection of axioms which can be used to derive theorems. An axiom is a statement that is presumed to be true, and generally considered to be self evident. Axioms serve as the basis for inference, as they are used collectively to prove or disprove statements. The main goal for the development of an axiomatic system is to determine which other properties can be deduced from these axioms. If an axiomatic system can be used to prove whether or not any statement holds, we say that the axiomatic system is complete [42].

Since its origin, several axiomatic systems have been introduced for modal logic. All the generally accepted axioms today are based on system K, which is named after Saul Kripke [66]. This system is the smallest normal modal logic, composed of three axioms:

- **Necessitation Rule** says that if A is a theorem, then $\Box$A is as well: A $\rightarrow \Box$A
- **Distribution Axiom** says that if it is necessary that A implies B, then necessarily A implies necessarily B: $\Box$(A $\rightarrow$ B) $\rightarrow$ ($\Box$A $\rightarrow \Box$B)
- **Definition of Possibility** states the relationship between $\Diamond$ and $\Box$: $\Diamond$A = $\neg\Box\neg$A

We use 'A' as a formula of the language. Some other common systems are composed of system

K along with additional axioms:

- **D** system K with additional axiom (D): $\Box A \rightarrow \Diamond A$.
- **T** system K with additional axiom (M): $\Box A \rightarrow A$
- **S4** system T with additional axiom (4): $\Box A \rightarrow \Box\Box A$
- **S5** system T with additional axiom (5): $\Diamond A \rightarrow \Box\Diamond A$
- **B** system T with additional axiom (B): $A \rightarrow \Box\Diamond A$

System D is a relatively weak system, often used when dealing with deontic logic [59]. System T forms again the basis for other systems. System T is system K with axiom (M), asserting that whatever is necessary is the case. Building on this system we have systems S4 and S5. S4 adds to system T axiom (4), stating that whatever is necessary is necessarily necessary. S5 adds axiom (5), stating whatever is possible is necessarily possible. We also have system B, composed of adding axiom (B), which says that whatever is the case is necessarily possible, to system T. S5 can alternatively be formulated by adding axiom (B) and axiom (4) to system T.

2.4.7 First Order Modal Logic

First order logic and modal logic as separate entities have already been introduced earlier in this chapter. The concept of first order modal logic, as summarized in [11] based on [24], is a form of modal logic where the underlying propositional logic is replaced with first order logic. This kind of logic is also known as quantified modal logic.

In first order modal logic, we have all the same connectives and modal operators as in ordinary modal logic. In addition, we have the quantifiers introduced for first order logic. What this means is that if ϕ is a first order formula, then $\Box\phi$ and $\Diamond\phi$ are formulas as well. This logic also adds the property of predicate symbols to the modal logic.

2.4.8 Applications

Since its origin in the early 1900s, modal logic has had many applications, including within philosophy, linguistics, and for its mathematical apparatus. Many of these are presented in detail in [9]. We also introduced some examples relevant for this thesis in the previous chapter.

The field we are most interested in for this thesis is modal logic in computer science. Within computer science, modal logic has been used for system verification, database theory, cryptography theory, etc. It has also been used within game theory, web design, and more. Here, we are

talking a range of different modalities which are utilized. Some examples include:

- Epistemic logic, which can be used to capture agent knowledge.
- Deontic logic, suitable for reasoning about normative law.
- Propositional dynamic logic, used for program verification.

Modal logic also has its uses in game theory, as it allows analysis of players' knowledge and beliefs, logic also generally provides a more abstract and general perspective on games. In [10], Bonanno describes how modal logic can be applied, and focuses on two views of game theory:

1. Game theory as a description of how rational individuals behave.
2. Game theory as a prescription or advice to players on how to act.

Here, it is argued that modal logic can be used to model both of these two conceptually different views of game theory.

From the examples presented in this section, it is clear that modal logic has a wide range of applications. We see that this is largely due to its ability to model expressions similar to natural language, which is a trait lacked by other classic logics. Another central trait of modal logic is that it allows us to express the knowledge and actions of actors, which is applicable to a variety of user systems.

Chapter 3

Method and Design

In this chapter, we describe the methods adapted in the implementation of the project, which includes the use of tools and technologies. As mentioned previously, the implementation of the project can be split into three main segments. The first section of this chapter presents the method for the design of our system and its functionality. The next section describes how we used the specifications from the first section to formulate a formal first order modal language and axiomatic system, to be able to express semantic properties. Lastly, we outline how we perform an evaluation of the correctness of our system and suitability of our approach.

3.1 Designing an Access Control Model

In order to perform some sort of analysis and evaluation of a system, we must first specify the intended use and the requirements of the system. The first part of the implementation entails designing an access control model for an online bank. This is done by laying out a design specification. This specification includes what types we have in our system and how they relate to each other, as well as describing the functional requirements and restrictions of the system.

Online banking is a tool most people use on a regular basis, and some of the features are well known or simple to deduce. However, there remains a number of functionalities where it is not as obvious how they should be formalized. Collecting a set of rules for our system is done using a variety of methods, including research and testing. Some assumptions about the specifics also have to be made. For example, when it comes to who is the owner of a loan account, as this is not explicitly clear.

A lot of our research on functionality was based on the systems of the Norwegian bank, DNB. Some information on features is available from DNB's website, along with tutorials found at [21]. In combination with other articles, such as [35] and [53] on how to share permissions for accounts, we can design a system where these features can be represented.

Another aspect of this part was testing in an existing online banking environment, in order to investigate what we were actually able to do and what restrictions there are. The testing is based on the online and mobile systems from DNB. Here, we had the possibility to test features such as assigning admin rights and restricted permissions in relation to transactions.

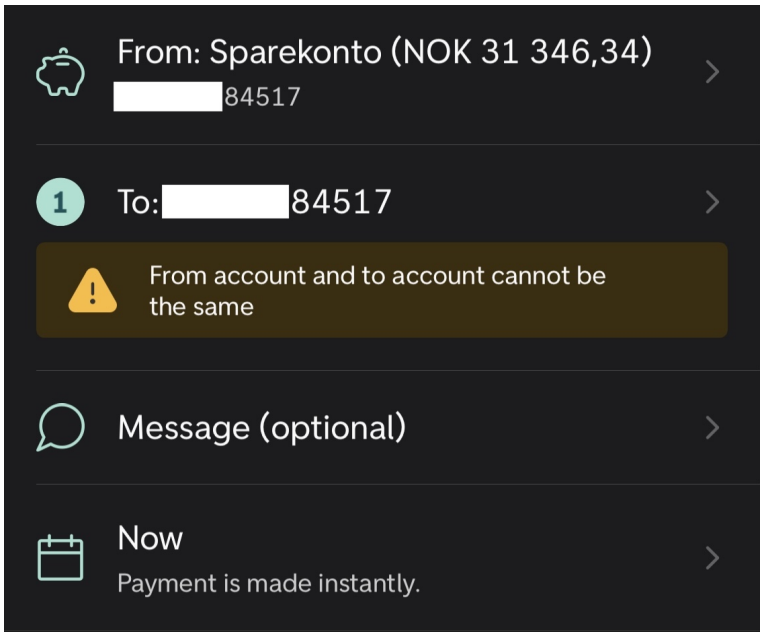


Figure 3.1: A screenshot from DNB's mobile bank.

In Figure 3.1, we see an example of how testing in an existing online banking system assisted us in the process of designing the rules for our system. In this example, we are trying to make a transfer with the same account as source and target. Immediately, we get an error message stating that this is not allowed. This complies to the principle of least privilege which we want our system to comply with.

3.1.1 Haskell

For visualisation and experimentation purposes, we implement some basic functionality of our system in Haskell. Haskell is a general purpose, statically typed and purely functional language with type inference and lazy evaluation [45]. The Haskell implementation allows us to clearly

represent what the various entities in our system consist of and how they relate to each other, as well as demonstrate some of the basic features of the system. For this project, we are mostly concerned with the access control. Other aspects of online banking, such as those having to do with changes in balance, are not our main focus and not accounted for in this thesis.

3.2 Constructing a Formal Language

Another requirement in order to perform a verification is a formal model of the system. This model is derived from the requirements that are specified for the system. This formal model does not need to contain details regarding the implementation of the system, but is rather an abstraction, or simplification, of the system. To achieve this, we want to construct a modal language. In order to construct a modal language suitable for reasoning about our system, we had to reflect on our design and what we needed to model. To improve the expressiveness of our language, we replace the underlying propositional logic of modal logic with first order logic, leading to a first order, or quantified, modal logic. Prior to constructing our own formal language, we did research on how modal logic has been used in similar work.

3.2.1 Syntax

Specifying a language for reasoning about our access control means that we needed to decide on all the syntactic qualities of a first order modal logic. This includes the modalities, predicates, relations and functions we need in order to be able to represent and reason about our system. This was a progressive process, where we introduced new elements as we discovered the need for them when considering different features and cases. We also had a look at similar work, such as [8] and [39], and took inspiration from the approaches presented.

Epistemic Modality

We saw in similar work that it is quite common to utilize a modality with syntax of the form i `says` ϕ, as opposed to $\Box_i\phi$ for readability, in order to model that actor i wants ϕ to happen. We use this syntax as a technique to model clearly what permissions the actors have and what they want to happen.

3.2.2 Axiomatic System

Once we have the basis for the syntax of our language, the next step is to use this to create an axiomatic system, which represent the rules in our system. These axioms are based on the design specification from the previous section, and formally model the behaviour of our system using our modal language. The goal is to be able to use these axioms, which are the basic requirements of our system, to show more complex results. The process of building this system happened gradually, as we considered different cases and discovered the need for various axioms.

3.2.3 Semantics

The semantics of our modal logic is based on structures known as Kripke structures and possible world semantics, introduced in Section 2.4.3. We use this semantics to represent models and behaviour of our system. In possible world semantics, each possible world is a complete representation of what a possible state in our system could consist of. The formulas that are true in this world tell us about what is true in the related possible worlds.

3.2.4 Agda

In order to ensure type correctness of our syntax, we implement the language and axioms in Agda, a proof assistant. Agda is an extension of Martin-Löf's intuitonistic type theory [50][2]. It is a dependently typed functional programming language and interactive proof assistant. This means that compared to other languages of the same style, the line between types and values is not as distinct. In Agda, types can depend on arbitrary values and appear as arguments and results of ordinary functions [51]. Another convenient feature of Agda is that it supports Unicode characters, allowing us to write proofs with high readability that are simple to comprehend. Using Agda as a tool for the syntax allows us to be less subjected to risk of human error in this section, and provides some credibility.

3.3 Analysis of Formal Language

As mentioned, the purpose of our formal specifications is to be able to reason about the logical correctness of our access control and examine how well our formal language can be used to

reason about access control. The final step is to be able to make an assessment on its usefulness and suitability. We take into consideration some concrete examples, which give us an idea of how well our approach works.

We use the previously designed system and our modal logic to model these examples, and show whether the formal logic conforms to the design specification. In order to do this, we specify a case and formalize it using our modal language. Thereafter, we can use our axiomatic system and formally show whether or not the expected outcome can be deduced from these. This will tell us about the correctness of our defined system.

In our approach in this section, we consider various properties that we have specified and investigate whether the property can be deduced from the axioms of the system. The idea is to use a model of our system and check whether it satisfies a given property. A model is a collection of formulas which are true at some point in time in the system. If the specified property does not hold in this model, it will induce a counter example.

The strategy for our verification is to separate our axioms into two categories: the axioms that are relatively simple to show correctness of and the more complicated axioms. We want to show how the axioms in the first category can be used to show the axioms in the second category in models of our system.

3.4 Summary

This chapter describes the methods used for developing the various components. It is worth mentioning that the three components consist of different tasks that are sequentially dependent on each other. However, it is not necessarily the case that the development is done strictly in this sequence. Elements of all three parts could be developed continuously in the implementation process.

Chapter 4

Design Specification

The method and design of the different components have been presented in the previous chapter. In this chapter, we present the implementation of the first component, which is designing a specification for an access control model for a hypothetical online bank. This specification describes what types we have in our system, and what the functionality is. In order to do this accurately, we have to familiarize ourselves with online banking systems and how they behave. Even though some elements of such a system are fairly obvious, a lot of the specifications are not trivial.

We have previously introduced a number of established access control models. Due to security risk this would induce in that it would make it easier for attackers looking to exploit, there is very limited content that is available to the public about the actual access control implemented in systems developed and used by banks. However, being a user many of the basic rules and restrictions a system needs to adhere to in order to be secure can be deduced. When it comes to more complicated functionality, the underlying logic becomes a bit more challenging to specify.

4.1 Sorts

The sorts that are specific for this system are:

- Identity
- Account
- Transaction

In addition to these, we also reason about components such as time and currency.

A bank account consists of a number of fields. Each account must have at least one owner, and can possibly have multiple owners. Additionally, each account must have a unique account number, an account type and a current balance.

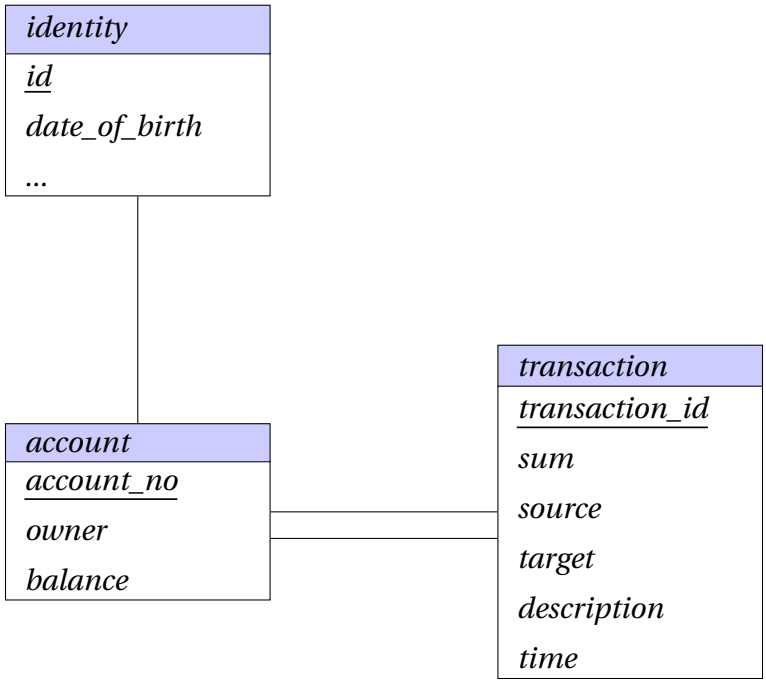


Figure 4.1: Schema diagram for transaction system.

As shown in Figure 4.1, we have a relation from account to identity. This is a many-to-many relationship. An identity can be the owner of multiple accounts, and each account can have multiple owners. Between transaction and account we have two relations, since both source and target in transaction are accounts. These relations are one-to-many, as each transaction has exactly one account as the source and one account as the target, but each account can be involved in multiple transactions. All accounts and transactions between them, as well as identities, are identified by a unique identifier.

In some particular cases, accounts can also have limits for how much can be deposited within a year or in total, or limitations regarding number of withdrawals. For example, BSU accounts have quite a few restrictions. As per 2021, one can only deposit a total of 27.500NOK per year, and a total of 300.000NOK. This type of account also requires owners to be at least eighteen years of age. As for customers under the age of 18, their accounts must, by law, be managed by a guardian. It is also quite typical for savings accounts with higher interest rates to limit how many withdrawals a year are free of charge.

Only a few examples of special accounts are included in this project in order to show that it is possible to take into account these special cases. We include account type BSU, which we consider as two accounts. One account with the total balance, including deposits made previous to the beginning of the current year. The second one only contains balance from deposits made the current year. This is because a BSU account will always give two balances, which are the available balance and total balance. Only the deposits made the current year are considered to be available balance. We only want to reason about this balance since customers cannot access funds that are not available. In addition to BSU, we also want to include accounts with a limited number of annual withdrawals.

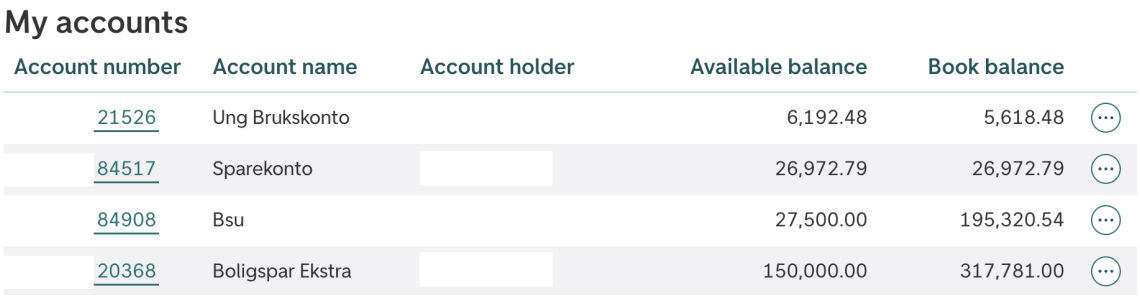


Figure 4.2: How balance is represented for BSU accounts in DNB's online bank.

In Figure 4.2, we see that the available balance in a BSU account is only from the current year. DNB offers a "Boligspar Ekstra" account, which is a similar concept to BSU. Here, we also see that the available balance and book balance for the checking account, called "Ung Brukskonto", are different amount. This difference is due to a reserved deposit of 714NOK to the account, that has not yet been booked. Normally, these two amounts would be the same, such as in the savings account, "Sparekonto".

A simple online bank prototype in Haskell would contain the following data types:

```
data Account = Account {owner :: [Identity], balance :: Currency}
        deriving Eq
data Identity = User {dob :: Date} deriving Eq
data Transaction = Transaction {amount :: Currency, source :: Account,
    target :: Account, description :: String, transactionTime :: Time,
    transactionDate :: Date}
data Bank = Bank {accounts :: Map Int Account,
                  transactions :: Map Int Transaction,
                  users :: Map Int Identity}
```

```
type Date = Day
type Time = TimeOfDay
type Currency = Integer
```

Listing 4.1: Sorts implemented in Haskell.

4.2 Functionality

This kind of system has very concrete functionality requirements that it needs to adhere to.

Roles are not a central part of this kind of system. We only have one role in our system, which is the role of an owner of an account. One property is that an owner of an account can delegate privileges, which is a useful trait. An administrator can add users, and from there have the possibility to delegate administrator access, or delegate individual permissions to accounts. These permissions can be one or more of the following:

- View transactions
- Initiate transactions

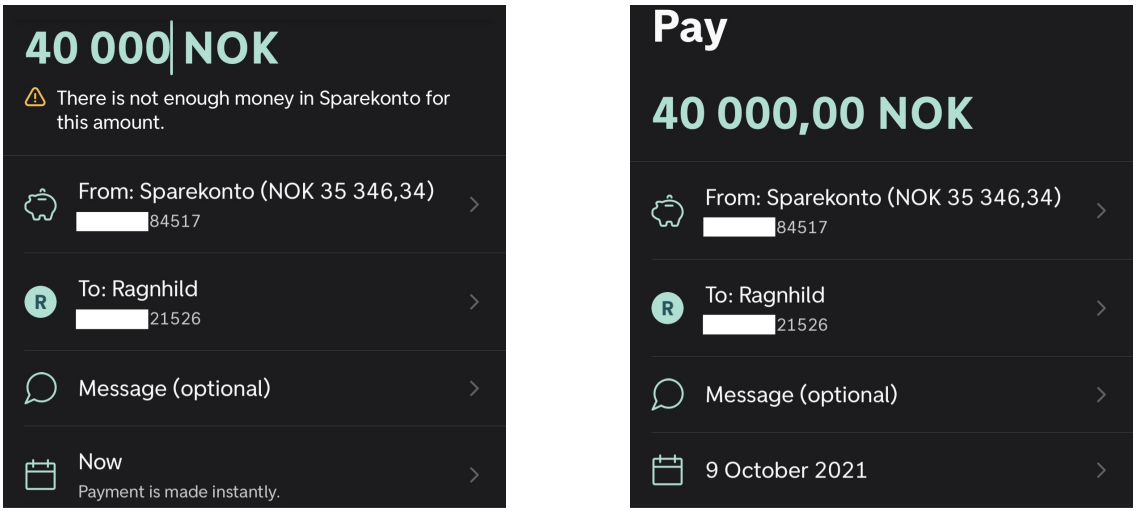
These permissions can be delegated on the basis of an ACL.

In order for this system to work as intended, a basic restraint is that a user can only transfer from an account they are the owner of, or have been delegated rights to. There is no need to regulate what accounts can be transferred to, as this generally is not a security issue. One restriction, however, is that one cannot perform a transaction that has the same account as the source and target. This was also mentioned in the previous chapter. This would be an entirely unnecessary feature, and avoiding it complies with the principle of least privilege. This principle refers to the information security concept in which a user is given the minimum amount of permissions necessary in order to perform their required functions.

Transactions must adhere to a set of properties. One of these properties is atomicity, we mean that for any transaction, it will either be executed completely or it will have no effect at all. They must also satisfy consistency, by which we mean that any transaction takes the system from one consistent state to another consistent state. We must also achieve isolation, so

that the transactions are performed serially and do not interfere with each other. Lastly, they must satisfy durability, meaning that once an interaction has been executed it will always have been executed. This set of properties is commonly known as ACID (for atomicity, consistency, isolation and durability) [38].

In order for a transaction to be specified and executed, some other criteria must be met. As we saw in the previous section, a transaction must have different accounts as the source and target, and cannot be created if this is not satisfied. In order for an account to be executed, the current date must be equal to the date of the transaction, and there must be coverage on the source account for the amount specified in the transaction. Here, we have to be careful with our definitions. The criteria for execution must not be satisfied at the time the transaction is created, but at the time of execution. If there is not coverage at the time specified, the transaction will not be executed.



(a) Specifying an immediate transaction.

(b) Specifying a transaction in the future.

Figure 4.3: Screenshots from DNB's mobile bank.

In our Haskell implementation, we also introduce a set of rules in order for a transaction to be valid:
A general check for any transaction:

```
isValid :: Transaction -> Bool
isValid (Transaction amount a0 a1 _ _ d) = a0 /= a1 && not(past d)
```

Listing 4.2: Check for if a transaction is valid in Haskell.

We can also represent rules for accounts with specific constraints, such as a maximum amount of yearly deposits:

```
maxDeposits :: Integer -> Transaction -> [Transaction] -> Bool
maxDeposits max (Transaction amount a0 a1 _ _ date) prevTransactions
                = amountIn prevTransactions a1 year 0
                - amountOut prevTransactions a1 year 0 + amount <= max
                    where year = getYear date
```

Listing 4.3: Check for if an account has reached maximum amount of annual deposit in Haskell.

As well as for accounts with a maximum number of yearly withdrawals:

```
maxWithdrawals :: Integer -> Integer -> Transaction -> [Transaction]
               -> Bool
maxWithdrawals max year (Transaction amount a0 _ _ _ date) transactions =
                noWithdrawals transactions a0 year 0 /= max
```

Listing 4.4: Check for if an account has reached maximum number of annual withdrawals in Haskell.

The full code with the helper functions is included in Appendix A.

4.2.1 Loans

We already mentioned that payments, including those on loans, are just considered transactions. Generally, we have two different types of loans. We distinguish between regular and flexible loans. For a regular loan, the loan taker will pay a certain amount on the loan each month, covering interest and installments. For this type of account, the loan taker cannot withdraw money from the loan account. With a flexible loan, the loan taker only pays interest on the amount of the loan that has been utilized each month, and the loan taker can withdraw and deposit money within the loan amount as they please. A credit card account also falls under the category of a flexible loan.

We conclude that the loan taker is not the owner of a loan account when it comes to regular loans, but that the loan taker has been delegated view permissions for the account from

the lender. However, with flexible loans, the loan taker is an owner, and they can manage the account as they wish within its boundaries.

4.3 Summary

Above we have discussed the design of the system and the behaviour. Here, we summarize the rules our system must adhere to. As stated, we are only concerned about the access control aspect, and do not worry about keeping track of changes in balances, etc.

Rules about transactions:

- Each transaction must have different accounts as source and target.
- Each account can only be involved in one transaction at a point in time.
- A transaction can only be made by an owner of the source account, or an actor that has been delegated permission by an owner of the source account.
- A transaction can be viewed by an owner of the source account, and an actor that has been delegated permission by an owner of the source account. After it has been executed, it can also be viewed by all owners of the target account, and any actor that has been delegated permission by an owner of the source account.
- In order for a transaction to be executed, the account must have coverage at the time of the transaction.
- A transaction must take into consideration requirements for account types at the time of execution.

Trivial concepts applying to real life, such as linearity of time, also apply to our system. This means that once a time has passed, it will always have passed in the future. From this, it follows that a transaction cannot be specified at a time that has already passed. It also follows that once a transaction has been executed, it will always have been executed in the future.

Chapter 5

Formal Logic

The next step is to construct a formal language for our system. There are a number of mathematical logics that can be used to reason about access control models. The logic chosen for this project is a multimodal logic, with an underlying first order logic. This allows us to reason using a combination of quantifiers and modalities. Primarily, first order logic lets us describe the state of a program, while modal logic lets us describe the dynamic changes between the states. The modalities are temporal and epistemic.

5.1 Syntax

We will split the definition of our language into several parts. Firstly, we have a set of variables V of different sorts in our system. In our syntax, each variable $v \in V$ is denoted by a lower case letter, and will be of one of the sorts included in our system.
Our sorts are:

- Identity: abbreviated to I.
- Account: abbreviated to A.
- Transaction: abbreviated to T.
- Date: abbreviated to D.
- Currency: abbreviated to C.
- AccountType: abbreviated to AT.

We then introduce S, which is the name of the entirety of all our sorts.

S ::= I | A | T | D | C | AT

For practical reasons, we also introduce a set of functions allowing us to reason in a simpler manner using only certain properties of our propositional types:

source :: Transaction → Account

target :: Transaction → Account

td :: Transaction → Date

amount :: Transaction → Currency

balance :: Account → Currency

accountType :: Account → AccountType

We also have two functions that are concerned with properties of accounts:

$\textbf{balance}_{min}$ :: Account → Currency

$\textbf{balance}_{max}$ :: Account → Currency

5.1.1 First Order Logic

We use our terms to define our first order logic:

$$\frac{v : V}{v : \text{Term}} \qquad \frac{p : \text{Formula}}{\neg p : \text{Formula}}$$

$$\frac{\phi_0 : \text{Formula} \quad \phi_1 : \text{Formula}}{\phi_0 \wedge \phi_1 : \text{Formula}} \qquad \frac{\phi_0 : \text{Formula} \quad \phi_1 : \text{Formula}}{\phi_0 \vee \phi_1 : \text{Formula}}$$

$$\frac{\phi_0 : \text{Formula} \quad \phi_1 : \text{Formula}}{\phi_0 \Rightarrow \phi_1 : \text{Formula}}$$

$$\frac{}{\top : \text{Formula}} \qquad \frac{}{\bot : \text{Formula}}$$

$$\frac{d : \text{Term D}}{\text{P}(d) : \text{Formula}}$$

$$\frac{t : \text{Term T}}{\text{E}(t) : \text{Formula}}$$

$$\frac{i : \text{Term I} \quad a : \text{Term A}}{\text{CanView}(i, a) : \text{Formula}} \qquad \frac{i : \text{Term I} \quad a : \text{Term A}}{\text{CanInit}(i, a) : \text{Formula}}$$

$$\frac{a : \text{Term A}}{\text{mw}_k(a) : \text{Formula}}$$

$$\frac{d_0 : \text{Term D} \quad d_1 : \text{Term D}}{d_0 < d_1 : \text{Formula}}$$

$$\frac{s : \text{S} \quad v_0 : V \quad v_1 : V}{v_0 =_s v_1 : \text{Formula}}$$

$$\frac{i : \text{Term I} \quad a : \text{Term A}}{\text{Owner}(i, a) : \text{Formula}}$$

$$\frac{v : V \quad p : \text{Formula}}{\forall v.\ p : \text{Formula}} \qquad \frac{v : V \quad p : \text{Formula}}{\exists v.\ p : \text{Formula}}$$

This definition includes our variables, which can be of any sort, as well as our predicate symbols:

Predicate	Description
P(d)	A unary predicate whose argument is a term d, and the intended interpretation is that d is in the past.
E(t)	A unary predicate whose argument is a term d, and the intended interpretation is that t is executed.
CanView(i, a)	A binary predicate whose arguments are two terms i and a, and the intended interpretation is that i can view transactions involving a.
CanInit(i, a)	A binary predicate whose arguments are two terms i and a, and the intended interpretation is that i can initiate transactions with a as the source.

Predicate	Description
$\mathrm{mw}_k(a)$	A unary predicate whose argument is a term a, and the intended interpretation is that a has a maximum of k withdrawals yearly.
$d_0 < d_1$	A binary predicate whose arguments are two terms d_0 and d_1, and the intended interpretation is that d_0 occurs before d_1.
$v_0 =_s v_1$	A binary predicate whose arguments are two variables of the same sort s, and the intended interpretation is that v_0 is equal to v_1.
$s_0 \geqslant s_1$	A binary predicate whose arguments are two numeric values, and the intended interpretation is that s_0 is greater than or equal to s_1.
$\mathrm{Owner}(i, a)$	A binary predicate whose arguments are two terms i and a, and the intended interpretation is that i is an owner of a.

Table 5.1: Predicates in our first order logic.

From the predicate P it follows that for a transaction t, if P applies to the transaction date of t, t has already been executed. Once executed, there does not exist a point in time in the future where it has not been executed. Likewise, once a date is in the past it will always be in the past. These truths follow from the necessitation rule, which is axiomatized later on.

The predicates $\mathrm{CanView}(i, a)$ and $\mathrm{CanInit}(i, a)$ have to do with permissions. These are used to assign permissions in the manner of ACL. For each identity i and account a, i will either have these permissions for account a, or they will not. The predicate $\mathrm{mw}_k(a)$ is related to permissions, and tells us how many withdrawals are permitted from account a annually. We can use this to ensure that no more than k withdrawals will be made from this account.

The relation $<$ is a linear order telling us about the order of points in time. An order being linear is an order that is irreflexive, transitive and total. For an order that is both irreflexive and transitive, it follows that the order is also antisymmetric.

The predicate Owner tells us about whether an actor is the owner of an account. Most accounts will only have one owner, but it is necessary to be able to model an account with more than one owner. For example, in the case of joint accounts.

Lastly, we also include the universal and existential quantifiers in our first order logic:

- $\forall v \in S.\ (p)$: p holds for all v
- $\exists v \in S.\ (p)$: p holds for some v

The convention of these quantifiers is to refer to our different sorts:

Variable	Sort
i	Identity
a	Account
t	Transaction
d	Date
c	Currency
at	AccountType

Table 5.2: How the different sorts are quantified.

This is because we want to have the possibility of referring to variables of only a certain sort. For example, when we write $\forall i$, we mean for all variables of sort I, and so on for each row in Table 5.2. This simplifies our axioms considerably, as we do not have to clarify further what sorts we are reasoning about.

5.1.2 Modal Logic

Further, we specify a definition of a formula ϕ including our modal logic properties.

$$\frac{i : \text{Term I} \quad \phi : \text{Formula}_M}{i \texttt{ says } \phi : \text{Formula}_M} \qquad \frac{i : \text{Term I} \quad \phi : \text{Formula}_M}{i \texttt{ observes } \phi : \text{Formula}_M}$$

$$\frac{\phi : \text{Formula}_M}{\Box\phi} \qquad \frac{\phi : \text{Formula}_M}{\Diamond\phi}$$

The modal operator `says` is used for readability, and is a binary volitive modality of where i `says` ϕ is another syntax for $\Box_i\phi$, meaning identity i wants ϕ to happen. Likewise, i `observes` ϕ means that identity i can observe ϕ. This also prevents confusing the modalities used with one another. The other two modal operators $\Box$ and $\Diamond$ are here used to represent temporal operators,

so $\Box\phi$ and $\Diamond\phi$ translate to always ϕ and sometime ϕ, respectively.

5.2 Axiomatic System

To be able to reason about our system and its functionality, we introduce a set of axioms. We can separate these axioms into three categories – axioms for first order logic, axioms for modal logic and axioms combining the two.

5.2.1 Shorthand Notations

In order to simplify a number of the axioms, and for readability, we introduce some shorthand notations. These notations are for transaction specification, current time, as well as for basic requirements of a valid transaction:

$$\begin{aligned}\text{TransactionSpec}(t, a_0, a_1, c, d) := \ & \text{source}(t) = a_0 \\ & \wedge\ \text{target}(t) = a_1 \\ & \wedge\ \text{amount}(t) = c \\ & \wedge\ \text{date}(t) = \text{d}\end{aligned}$$

$$\text{N}(d_{now}) := \text{P}(d_{now}) \wedge \forall d.\ (\text{P}(d) \Rightarrow (d \leq_D d_{now})$$

$$\text{IsValid}(a_0, a_1, td) := \neg\text{P}(td) \wedge \neg(a_0 =_A a_1)$$

The shorthand notation TransactionSpec(t, a_0, a_1, c, d) allows us to refer to the various properties of a transaction in a simpler manner. N(d_{now}) tells us that d_{now} is the current time, by comparing d_{now} to all other dates d. If another date is in the past, it will either be previous to the current time or equal to the current time, as the current time is the most recent time that is considered to be in the past. The notation IsValid(a_0, a_1, td) only takes into consideration the criteria necessary in order for a transaction to be specified, not executed. These criteria are that the transaction time is not in the past, and that the source and target accounts are not the same account, as mentioned previously. A transaction with specifications like this will never be executed, and therefore do not need to exist to begin with. Some other criteria will come in addition to these, but will not be checked until the time of execution.

Lastly, we want to be able to reason about the time interval of a year. We introduce a constant, BY, for our system, which is a date. This constant represent the beginning of the year. This constants will of course change when a new year commences. Trivially, this constants represent

the beginning of the year of the current time.

5.2.2 Isolation

As we mentioned in Section 4.2, this kind of system must implement concept of isolation when it comes to transactions. The purpose of isolation is to control the execution of simultaneously executed transactions, so that the results are not effected by each other. For example, this prevents the balance of an account from drop bellow the permitted minimum amount due to numerous transactions being executed from the account at the same time, which all are valid for execution individually. In a banking system, customers are not themselves responsible for ensuring isolation. When a user specifies a transaction, they only specify the date. The bank then assigns a point in time on this date as the time of execution.

5.2.3 Modal Logic Axioms

First, we list axioms that are common for modal languages:

- $\Box\phi \Rightarrow \Box\Box\phi$ (4)
- $\Box\phi \Rightarrow \phi$ (T)
- $\Box\phi \Rightarrow \Diamond\phi$ (D)
- $\Diamond\phi \Rightarrow \Box\Diamond\phi$ (5)

These are standard modal logic axioms that are applicable to all systems.

Further, for a principal I, our logical connective `says` satisfies the general axioms:

- $\phi \Rightarrow (i \texttt{ says } \phi)$ (unit)
- $(i \texttt{ says } (\phi_0 \Rightarrow \phi_1)) \Rightarrow (i \texttt{ says } \phi_0) \Rightarrow (i \texttt{ says } \phi_1)$ (cuc)
- $(i \texttt{ says } i \texttt{ says } \phi) \Rightarrow (i \texttt{ says } \phi)$ (idem)

The first axiom, unit, states that every formula that is true is supported by each identity i. The converse of this is not true, as identities are free to make statements that are false. The second axiom, cuc, tells us that an identity i's statements are closed under logical consequence. If i

states that ϕ_0 implies ϕ_1, and that ϕ_0, it also states ϕ_1. Lastly, the axiom idem shows that we can collapse the applications of `says`.

Last in our modal logic axioms we introduce a set of axioms formally describing how time is a linear concept:

- $\Diamond \mathrm{P}(d)$ (Past)
- $d_0 < d_1 \Rightarrow P(d_1) \Rightarrow P(d_0)$ (L-T-0)
- $P(d) \Rightarrow \Box P(d)$ (L-T-1)

The axiom (L-T-0) follows from the relation < being a linear relation. The axiom (L-T-1) also follows from the idea that time is linear – once a time has passed, it will never not have passed in the future.

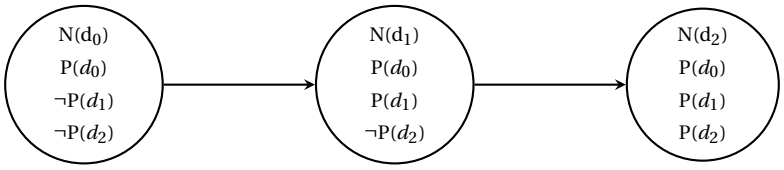


Figure 5.1: Linear representation of passed moments in time.

In Figure 5.1 we also see that once d is the current time, d has passed. Since the order of these points in time is that d_0 occurs before d_1 and d_1 occurs before d_2, it is also the case that that d_0 occurs before d_2. It will also never be the case that d_2 has passed without d_1 having passed.

5.2.4 First Order Modal Logic Axioms

The axioms in this section are the ones used to represent permissions in our access control system. Here, we deal with permissions and delegations of them. We also look at requirements for transactions to be specified and executed.

Axioms describing permissions to initiate transactions:

- $\forall a, i.\ \mathrm{Owner}(i, a) \Rightarrow \mathrm{CanInit}(i, a)$ (Init)
- $\forall a, i, i'.\ \mathrm{Owner}(i, a) \wedge i\ \texttt{says}\ \mathrm{CanInit}(i', a)$
  $\Rightarrow \mathrm{CanInit}(i', a)$ (Init-Delegation)

- $\forall a_0, a_1, c, d, i.\ \text{CanInit}(i, a) \wedge \text{IsValid}(a_0, a_1, d)$
  $\wedge\ \text{i}\ \texttt{says}\ (\exists t.\ \text{TransactionSpecification}(t, a_0, a_1, c, d))$
  $\Rightarrow \Diamond\, \exists t.$
  $\quad \text{TransactionSpecification}(t, a_0, a_1, c, d)$ (Initiate)

Axioms describing permission to view transactions:

- $\forall a, i.\ \text{Owner}(i, a) \Rightarrow \text{CanView}(i, a)$ (View)
- $\forall a, i, i'.\ \text{Owner}(i, a) \wedge i\ \texttt{says}\ \text{CanView}(i', a)$
  $\Rightarrow \text{CanView}(i', a)$ (View-Delegation)
- $\forall t, i, a.\ \text{CanView}(i, a) \wedge ((\text{source}(t) =_A a) \vee (\text{target}(t) =_A a \wedge \text{E}(t)))$
  $\Rightarrow i\ \text{observes}\ (\exists t'.$
  $\quad \text{TransactionSpec}(t', \text{source}(t), \text{target}(t), \text{amount}(t), \text{date}(t)))$ (Observe)

These axioms clearly separate these two types of permissions, and how a user can attain these permissions. It is also described by axioms (Initiates) and (Observe) how transactions are initiated and observed, given that the correct permissions are in place. Transactions can be initiated by an owner of the source account, and an owner can also specify that another identity can make transactions from that account. An identity i can initiate a transaction t if t is valid, and one of the following is satisfied:

- i is an owner of the source of t.
- An owner of the source account of t specifies that i can initiate transactions from the account.

An identity i can observe a transaction t if one of the following is satisfied:

- i is an owner of the source of t.
- An owner of the source account of t specifies that i can observe transactions for the account.
- i is an owner of the target of t and t has been executed.
- An owner of the target account of t specifies that i can observe transactions for the account and t has been executed.

We have now described how a transaction can be initiated. What remains is the execution of the transaction. In this step, we also need to take into consideration specific requirements accounts may have.

As we mentioned, we also have different types of accounts. These types vary from bank to bank, and there is initially no limit to what types of accounts banks can offer their customers. We introduce another couple of axioms to demonstrate how the special rules for these types of accounts can be formalized. Here, BSU_i represents the BSU account that has to do with the current year.

- $\forall a.\, \text{accountType}(a) =_{AT} \text{BSU}_i \Rightarrow \text{balance}_{max}(a) =_C 27500$ (BSU)
- $\forall t.\, \text{N}(\text{td}(t)) \Rightarrow ((\text{balance}(\text{source}(t)) \geqslant \text{amount}(t))$
  $\wedge\ (\text{balance}_{max}(\text{target}(t)) \geqslant (\text{balance}(\text{target}(t)) + \text{amount}(t)))$
  $\wedge\ (\text{balance}(\text{source}(t)) \geqslant (\text{balance}_{min}(\text{source}(t)) + \text{amount}(t)))$
  $\wedge\ (\text{mw}_k(\text{source}(t)) \wedge (\forall \overrightarrow{t} \in T^k.\, \left(\bigwedge_{i=1}^{k} (\text{BY} < \text{td}(t_i)) \wedge \text{E}(t_i)\right.$
  $\left.\wedge\ (\text{source}(t_i) =_A \text{source}t)) \Rightarrow \left(\bigvee_{i=1}^{k} \left(\bigvee_{j=i+1}^{k} (t_i =_T t_j)\right)\right)\right)\right)$
  $\Rightarrow \Box\text{E}(t))$ (Exec)
- $\forall a.\, \text{mw}_k(a) \Rightarrow (\forall \overrightarrow{t} \in T^k.\, \left(\bigwedge_{i=0}^{k} (\,\text{BY} < \text{td}(t_i)) \wedge \text{E}(t_i) \wedge (\text{source}(t_i) =_A a)\,\right)$
  $\left.\Rightarrow \left(\bigvee_{i=0}^{k} \left(\bigvee_{j=i+1}^{k} (t_i =_T t_j)\right)\right)\right)$ (Max-Withdraw)
- $\neg\exists a.\, \text{balance}(a) \geqslant \text{balance}_{max}(a)$ (Max-Balance)
- $\neg\exists a.\, \text{balance}_{min}(a) \geqslant \text{balance}(a)$ (Min-Balance)
- $\neg\exists t.\, \text{source}(t) =_A \text{target}(t)$ (Valid-Acc)

We also describe how transactions are executed when the current time is the transaction time. A transaction can only exist when specified by someone with permission, but this does not mean that it will be executed. Once the executed time is the current time, some additional requirements must be met. One of these being that there must be coverage on the account. In addition, there could be other criteria for special account types.

The last axioms follows from our criteria for accounts in axiom (Exec). For accounts with a maximum of k withdrawals a year. This axiom states that given that there has been executed k+1 transactions this year from that account, at least two of them must be the same transaction.

Axioms (Max-Balance) and (Min-Balance) state that the balance of an account will stay within its permitted range. The need for axiom (Valid-Acc) is explained in Section 4.2. This axiom simply states that there does not exist any transactions where the source is the same account as the target, this to comply with principle of least privilege.

5.2.5 Agda

To ensure type correctness of our axioms, we have implemented the syntax in Agda and defined the axioms. The Agda code can be found in Appendix B. By loading the Agda file, Agda type checks the contents [1]. If the file does not type check, Agda will raise an error.

5.3 Semantics

As mentioned in the previous chapter, the semantics of modal logic is based on structures known as Kripke structures. Since our logic includes a great number of modalities, there exists an equal number of relations over our set of worlds. This structure is a tuple $\mathcal{M} = \langle W^{\mathcal{M}}, \{S_i\}_{i\in I}, \{O_i\}_{i\in I}, T, V\rangle$, where $W^{\mathcal{M}}$ is a set of possible worlds, S_i, O_i and T are binary relations between the worlds in $W^{\mathcal{M}}$, and V is a valuation function mapping each atomic formula to a binary relation over $W^{\mathcal{M}}$.

We have one relation for our temporal modality, T. In addition, we have one relation for each $i \in I$, in R and O. What this means is that we have one relation for each identity in our system. These relations tell us about the relations between states. The sorts in our system are global sorts, meaning they are in all $w \in W^{\mathcal{M}}$. V tells us about what is true and what is false in each world. Each world contains a model for our language, meaning an interpretation of the terms, which includes functions, relations and predicates. For each $w \in W^{\mathcal{M}}$, we have:

$$\begin{aligned} V(w) = \langle\ &\text{source}^w, \text{target}^{\,w}, \text{td}^w, \text{amount}^w, \text{balance}^w, \\ &\text{accountType}^w, \text{balance}^w_{min}, \text{balance}^w_{max}, \text{P}^w, \text{E}^w, \\ &\text{CanView}^w, \text{CanInit}^w, \text{mw}^w_k, <^w, =^w_s, \geqslant^w, \text{Owner}^w\ \rangle \end{aligned}$$

Where each element in the tuple is the interpretation of the respective relation, function or predicate in w. For example, for each date there is an interpretation telling us whether or not the date has passed.

5.3.1 Interpretation of Formulas

The interpretation of a formula is a set of tuples with elements interpreting the free variables. Hence, for a closed formula, true and false correspond to the empty tuple, $\{\langle\rangle\}$, and the empty set, $\varnothing$, respectively.

We inductively define the truth of a formula ϕ at a state w in structure $\mathcal{M}$. The interpretation is defined relative to α, which is a list of sorts for the free variables. The valuation of a predicate is a collection of tuples of variables that are true for the predicate in w.

$$[\![\phi]\!]^{\alpha}_{\mathcal{M},w} \subseteq \mathcal{M}^{\alpha}$$

This use of double brackets is commonly used in logic, and has the intended meaning in the equation above that the interpretation of formula ϕ in model $\mathcal{M}$ and world w. For example, we have an interpretation of the predicate E in each $w \in W^{\mathcal{M}}$. Here, x is used to denote variables.

$$[\![\mathrm{E}(x)]\!]^{\langle T\rangle}_{\mathcal{M},w} := \mathrm{E}^{w}$$

Induction on negation, conjunction, disjunction and implication is similar to how valuation is determined in propositional logic.

$$\begin{aligned}
[\![\neg\phi]\!]^{\alpha}_{\mathcal{M},w} &:= \mathcal{M}^{\alpha} \setminus [\![\phi]\!]^{\alpha}_{\mathcal{M},w} \\
[\![\phi_0 \vee \phi_1]\!]^{\alpha}_{\mathcal{M},w} &:= [\![\phi_0]\!]^{\alpha}_{\mathcal{M},w} \cup [\![\phi_1]\!]^{\alpha}_{\mathcal{M},w} \\
[\![\phi_0 \wedge \phi_1]\!]^{\alpha}_{\mathcal{M},w} &:= [\![\phi_0]\!]^{\alpha}_{\mathcal{M},w} \cap [\![\phi_1]\!]^{\alpha}_{\mathcal{M},w} \\
[\![\phi_0 \Rightarrow \phi_1]\!]^{\alpha}_{\mathcal{M},w} &:= \{\, x \mid x \in [\![\phi_0]\!]^{\alpha}_{\mathcal{M},w} \text{ implies } x \in [\![\phi_1]\!]^{\alpha}_{\mathcal{M},w}\}
\end{aligned}$$

Next, we define the semantics of our first order logic quantifiers:

$$\begin{aligned}
[\![\forall x \in S.\phi]\!]^{\alpha}_{\mathcal{M},w} &:= \{v \in \mathcal{M}^{\alpha} \mid \text{for all } s \in S^{\mathcal{M}} \text{ such that } s, v \in [\![\phi]\!]^{s\alpha}_{\mathcal{M},w}\} \\
[\![\exists x \in S.\phi]\!]^{\alpha}_{\mathcal{M},w} &:= \{v \in \mathcal{M}^{\alpha} \mid \text{there exists } s \in S^{\mathcal{M}} \text{ such that } s, v \in [\![\phi]\!]^{s\alpha}_{\mathcal{M},w}\}
\end{aligned}$$

Lastly, we have the semantics of our modal operators:

$$\begin{aligned}
[\![i \,\texttt{says}\, \phi]\!]^{\alpha}_{\mathcal{M},w} &:= \bigcap_{u \mid wS_i u} [\![\phi]\!]^{\alpha}_{\mathcal{M},u} \\
[\![i \,\texttt{observes}\, \phi]\!]^{\alpha}_{\mathcal{M},w} &:= \bigcap_{u \mid wO_i u} [\![\phi]\!]^{\alpha}_{\mathcal{M},u} \\
[\![\Box\phi]\!]^{\alpha}_{\mathcal{M},w} &:= \bigcap_{u \mid wTu} [\![\phi]\!]^{\alpha}_{\mathcal{M},u} \\
[\![\Diamond\phi]\!]^{\alpha}_{\mathcal{M},w} &:= \bigcup_{u \mid wTu} [\![\phi]\!]^{\alpha}_{\mathcal{M},u}
\end{aligned}$$

Lastly, we present the rules for tautology and contradiction:

$$[\![\top]\!]^{\alpha}_{\mathscr{M},w} := \mathscr{M}^{\alpha}$$
$$[\![\bot]\!]^{\alpha}_{\mathscr{M},w} := \varnothing$$

Each model of a language can be described as a snapshot of the dynamic system. The semantics in this case consists of a set of these snapshots. In a Kripke structure, each node represents a model – a possible system state. In our case, these snapshots include the individuals, transactions, accounts, etc. that are relevant in the current snapshot. This means that at the state when the bank is first established, before a single customer or account exists, none of these things exist at the current snapshot. However, this snapshot has a relation to a set of possible next states, where there might exist a customer. The axiomatic system is what tells us about what the further possible states are.

When it comes to open formulas, the interpretation is neither true nor false, but is a subset of all possible variable insertion which makes the formula true. If no such subset exists, the interpretation can only be a subset of the empty set, meaning the formula is false. The interpretation $[\![P(d)]\!]$ will give us the set of all past dates, $[\![P(d) \vee \neg P(d)]\!]$ will give us the union of the set of dates that are in the past and the set of dates that are not in the past, which is the set of all dates,as all dates will fall under one of these categories. Likewise, a statement that is a contradiction, such as, $[\![P(d) \wedge \neg P(d)]\!]$ will give us the intersection of these two sets, which is just the empty set $\varnothing$, and means that the formula is false in the model, because there is no date that exists in both of these sets.

Introducing quantifiers, these will bind one or more of the variables in the formula. Only the free variables will be present in the resulting set. For example, $[\![\exists d.\ P(d) \vee \neg P(d)]\!]$ will give us $\{\langle\rangle \mid \text{there exists } a \text{ such that } \langle a\rangle \in [\![\ P(d) \vee \neg P(d)]\!]\}$. Again, this is a contradiction, and $[\![\exists d.\ P(d) \wedge \neg P(d)]\!]$ will give us the empty set $\varnothing$. However, if we change one of the variables to be a free variable, $[\![\exists d.\ P(d) \wedge \neg P(e)]\!]$, we would get $\{\langle b\rangle \mid \text{there exists } a \text{ such that } \langle a, b\rangle \in [\![\ P(d) \vee \neg P(e)]\!]\}$. The same idea goes for the universal quantifier. The interpretation of $[\![\forall d.\ P(d) \vee \neg P(d)]\!]$ will be $\{\langle\rangle \mid a \text{ such that all } \langle a\rangle \in [\![\ P(d) \vee \neg P(d)]\!]\}$.

Chapter 6

Analysis

The goal with this project and the purpose of the previously implemented elements is to consider how formal verification can be beneficial for online banking systems. This chapter aims to demonstrate how we can model different cases and states of an online banking system.

6.1 Example Models

What relations we have between worlds in our dynamic system is decided by our axioms. Let's say we have a world w_0 where we have an actor i who wants to initiate a transaction t, so we get the interpretation $[\![i \texttt{ says } \exists t, a_0, a_1, c, d.\ \text{TransactionSpec}(t, a_0, a_1, c, d)]\!]^{\alpha}_{\mathscr{M}, w_0}$. Our axioms determine whether or not the transaction will actually be executed.

If, in w_0, t has the same account as source and target, w_0 has no relations to any models where t exists. In the example in Figure 6.1, even though w_0 contains more information, we do not have to worry about whether i has permission to make this transfer, t already has a contradiction at axiom (Valid-Acc), and can not be initiated in w_0, and will therefore not exist in a related world.

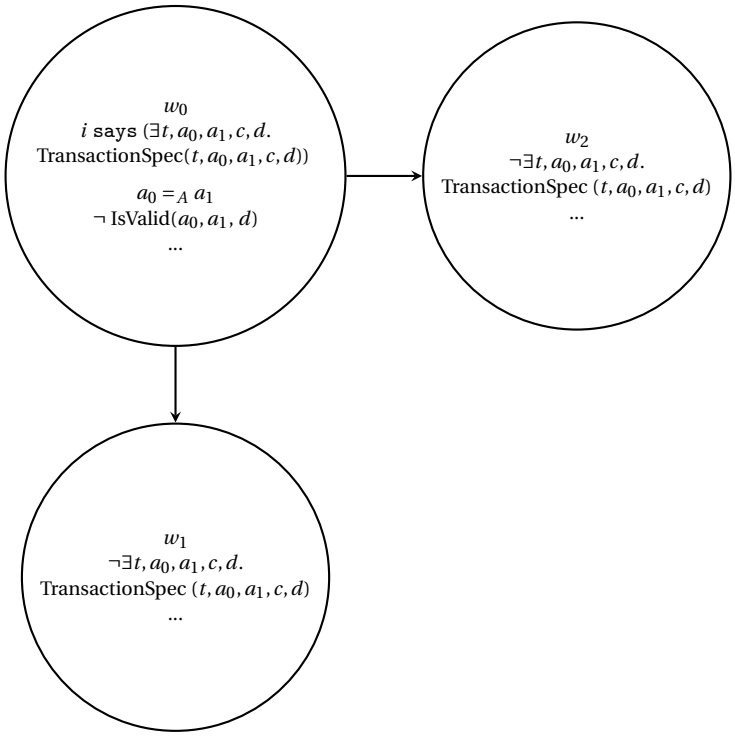


Figure 6.1: Example model of invalid transaction specification.

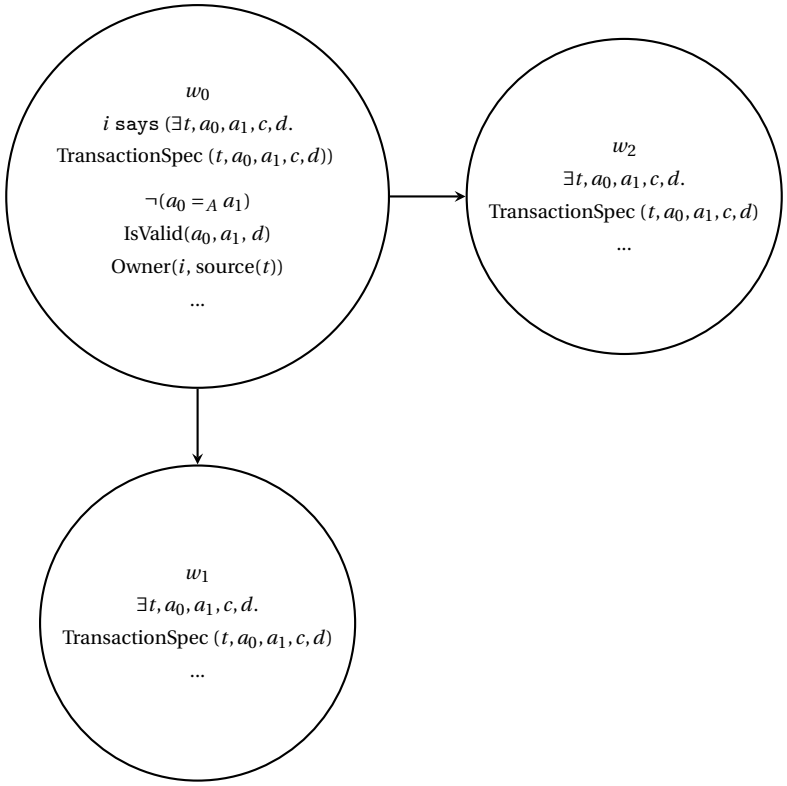


Figure 6.2: Example model of valid transaction specification.

On the other hand, if t actually is valid with respect to the axioms, and i has permission to execute t, it is not possible that we have a relation to a world where t is not initiated. This is shown in Figure 6.2. If i wants t to be initiated, it follows from our axioms that $\exists t, a_0, a_1, c, d.$ TransactionSpec(t, a_0, a_1, c, d).

In Figure 6.2, we have that a transaction specification for transaction t exists in all worlds accessible from w_0. This only means that t has been initialized, t has not yet been executed. If one of these worlds include formulae that satisfy the conditions for execution, this will lead to t being executed. We see an example of this in Figure 6.3 where we expand on w_1. Here, the current time is the time of the transaction. From axiom (Exec), it follows that as long as the requirements for execution are satisfied, then t will be executed in all worlds accessible from w_1.

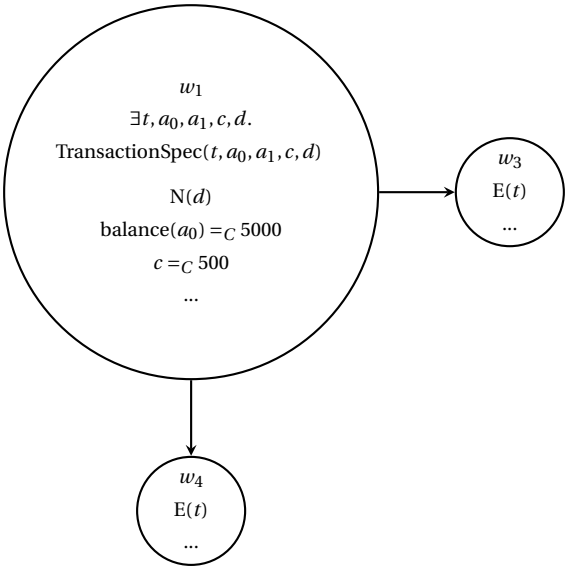


Figure 6.3: Expansion of w_1 in Figure 6.2

Here, it is implied that the accounts involved do not have any special requirements with respect to withdrawals or deposits. This leaves only the requirement that the account has coverage to make the transaction. This is clearly satisfied, as the balance of the source account is greater than the amount of t.

6.2 Delegation of Permissions

For some more advanced examples, let us consider what circumstances we might have related to family relationships within a bank. This is a relevant case to consider, as it is very common to be involved in the economy of one another within a family.

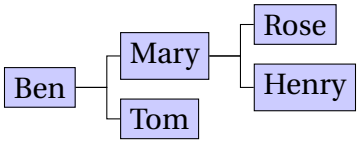


Figure 6.4: Hypothetical family tree.

Let us consider the family structure represented in the family tree above as an example. We say that Mary is a customer in the bank. Her partner, both her parents, and her son also have accounts in this same bank. Mary's parents are elderly, and not comfortable using the online banking services. They are not used to banking this way, and are worried about doing something wrong. For this reason, Mary has access to all of their accounts, so that she can help them with payments and keep an eye on their finances. In this case, Mary's parents have simply delegated permissions to Mary so that she can view transactions related to their accounts, as well as make transactions from them. Formally:

Rose says $(\forall a.\ \text{Owner}(\text{Rose}, a) \Rightarrow \text{CanView}(\text{Mary}, a))$

Henry says $(\forall a.\ \text{Owner}(\text{Henry}, a) \Rightarrow \text{CanView}(\text{Mary}, a))$

Both of Mary's parents give Mary permission to view transactions related to all of their accounts, and all that have been made to any of their accounts. From this it follows from axioms (View-Delegation) and (Observe) that Mary can observe all transactions from their accounts or that have been executed to their accounts. Formally:

Rose says $(\forall a.\ \text{Owner}(\text{Rose}, a) \Rightarrow \text{CanInit}(\text{Mary}, a))$

Henry says $(\forall a.\ \text{Owner}(\text{Henry}, a) \Rightarrow \text{CanInit}(\text{Mary}, a))$

They also give her permission to initiate transactions from their accounts. From this it follows from axioms (Init-Delegation) and (Initiates) that Mary can specify transactions from accounts that they own, as long as the transactions are valid.

Mary's children have no issues using the online services, but minors are by law unable to open their own accounts. For this reason, Mary has established joint bank accounts for her

children. A joint account for a minor generally entails having both the minor and a parent as an owner until the minor turns 18. This in practice means that both the parent and the minor can make deposits and withdraw funds. This can be helpful when it comes to regulating monthly spending, as well as keeping an overview of the spending.

Mary also has joint finances with her husband, Tom. This means that both Mary and her husband are registered as owners of their joint account and its contents. This means that from axiom (Init), they both have permission to initiate transactions from their joint account.

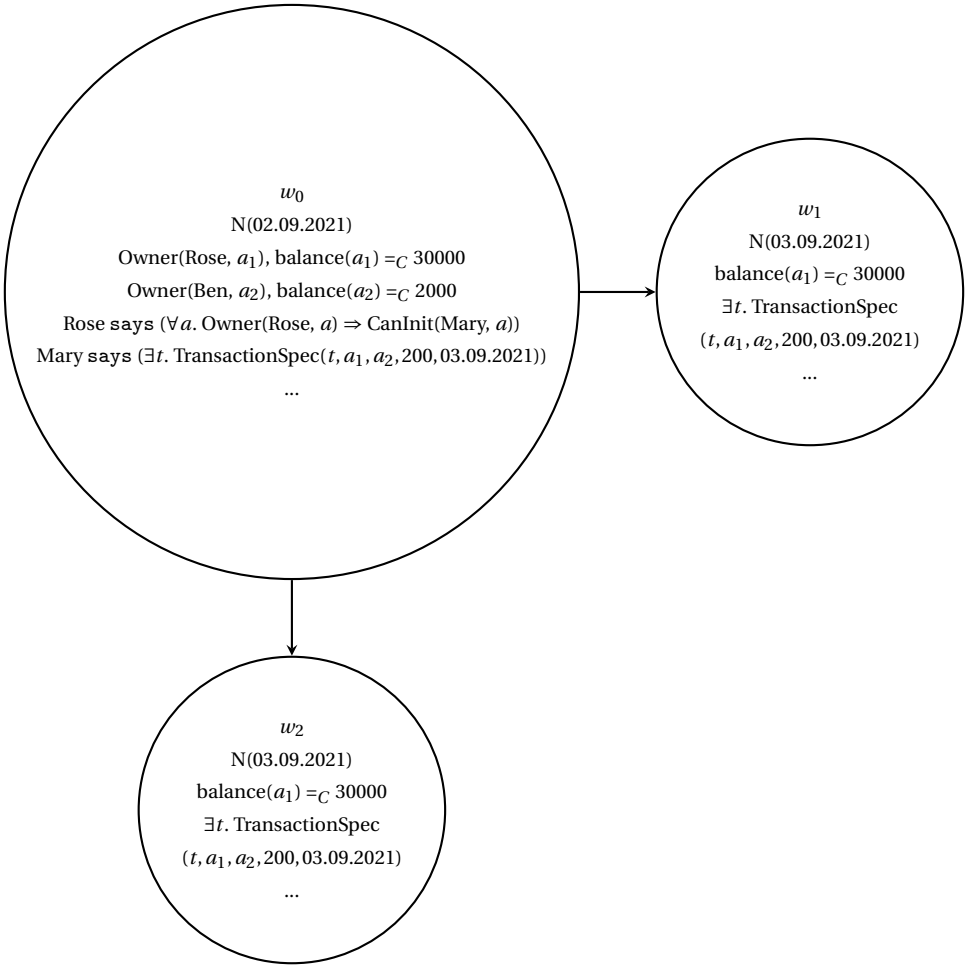


Figure 6.5: Example model demonstrating delegation of privileges.

In the example in Figure 6.5, Mary wants to help her mother transfer money to her child as a birthday present. It follows from our axioms that the transaction specified by Mary from her mother's account is valid and will be executed. The transaction is valid because it has different accounts as source and target and the time specified has not already passed. Since Mary has permission to initiate transactions from her mother's accounts, Mary is allowed to make trans-

actions from this account. World w_0 has a relation to possible worlds w_1 and w_2 where the date is the transaction date, and it follows from our axiom (Exec) that the transaction will be executed in all worlds accessible from w_1 and w_2.

In general, there may also be other reasons a customer would want to give others access to see their transactions. An example is for the purpose of accounting. Programs such as Quicken, Mint, and many more allow you to view and pay bills, create budgets, etc. to assist with money management. In order to be able to do this, these programs need to be able to view the users' account activity and transactions.

When it comes to proving that our axiomatic system is correct, the axioms can be divided into two categories. We have some axioms that are very simple to show correctness of, and some that are considered more complex.

6.3 Simple Axioms

We will first consider the simpler axioms in our system.

6.3.1 First Order Logic Axioms

All transactions that are specified must satisfy the criteria for IsValid, which states that source account and target account cannot be equal. From this, the axiom (Valid-Acc) follows.

6.3.2 Modal Logic Axioms

Axioms (4), (T), (D), (5), (unit), (cuc) and (idem) are all general axioms in modal logic, and are characteristics of modal logic. These statements are assumed to be true.

All the time axioms are trivial with respect to the linearity of time. This is also shown in Figure 5.1.

6.4 Complex Axioms

The complex axioms mostly consist of longer axioms that combine properties of first order logic and modal logic, and have to do explicitly with access control.

In Chapter 4, we communicated how permissions are assigned. This is formalized in Section 5.2.4. Here, predicates CanInit and CanView represent permissions of initiating and viewing transactions, respectively. Axioms (Init) and (View) show how the owner of an account automatically has permission to view transactions involving the account. Further, axioms (Init-Delegation) and (View-Delegation) specify that an owner can delegate these permissions to other actors.

Lastly, axioms (Initiate) and (Observe) formalize how any actor with the correct permissions can initiate and observe transactions.

We also want to show how we could account for accounts with special requirements, and include some examples in our system and axioms. Axiom (Max-Withdraw) follows from our axiom (Exec) for some k. The latter ensures that a transaction is not executed if maximum number of unique withdrawals is already executed. The repercussion is that more than k transactions from account a that year will not occur, which is what is stated by axiom (Max-Withdraw). The same goes for axioms (Max-Balance) and (Min-Balance). Since it is checked before execution that transactions will not cause balances to result in being below minimum allowed amount or above maximum allowed amount, it follows that the balance of an account will stay within this range.

The decision of whether a transaction with an account of type BSU as the source should be executed is done in a similar manner.

In Figure 6.6, we see that the transaction t from a regular account to a BSU account is initiated in w_0. This means that the transaction exists in worlds following from w_0. At the time the transaction is initiated for, the requirements for a transaction to be made with a BSU account as the target account will be checked, and the transaction will only be executed given that these requirements are satisfied. We assume that the rest of the formulas from w_0 are still true in the following worlds.

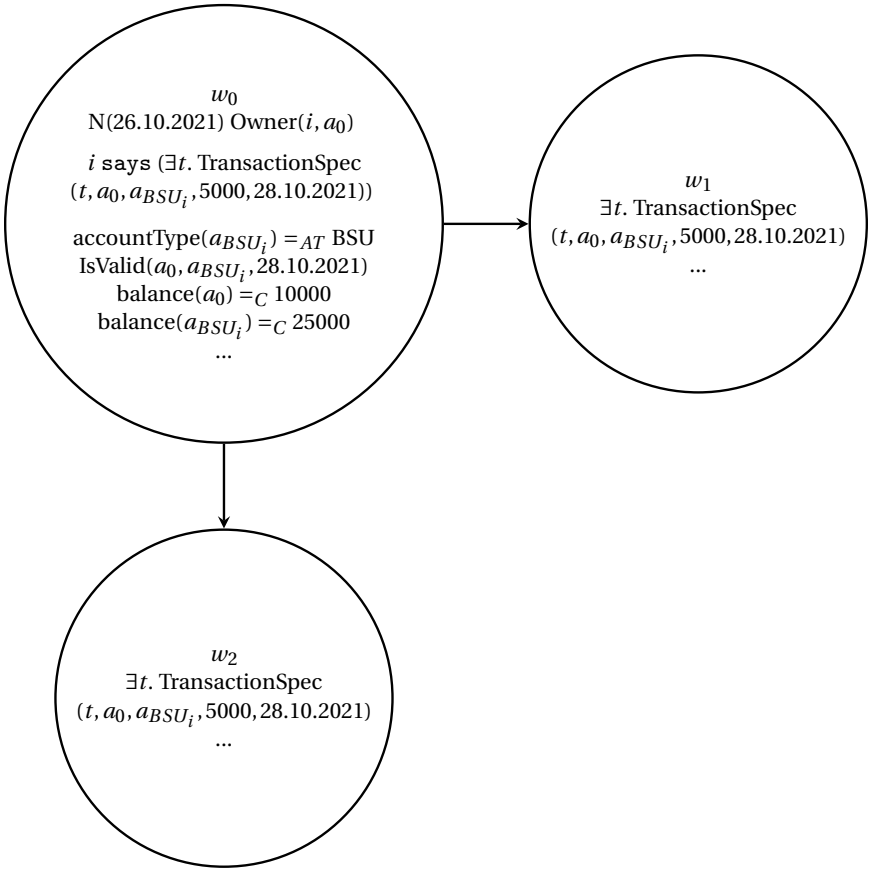


Figure 6.6: Example model demonstrating validity of transactions with a BSU account as target account.

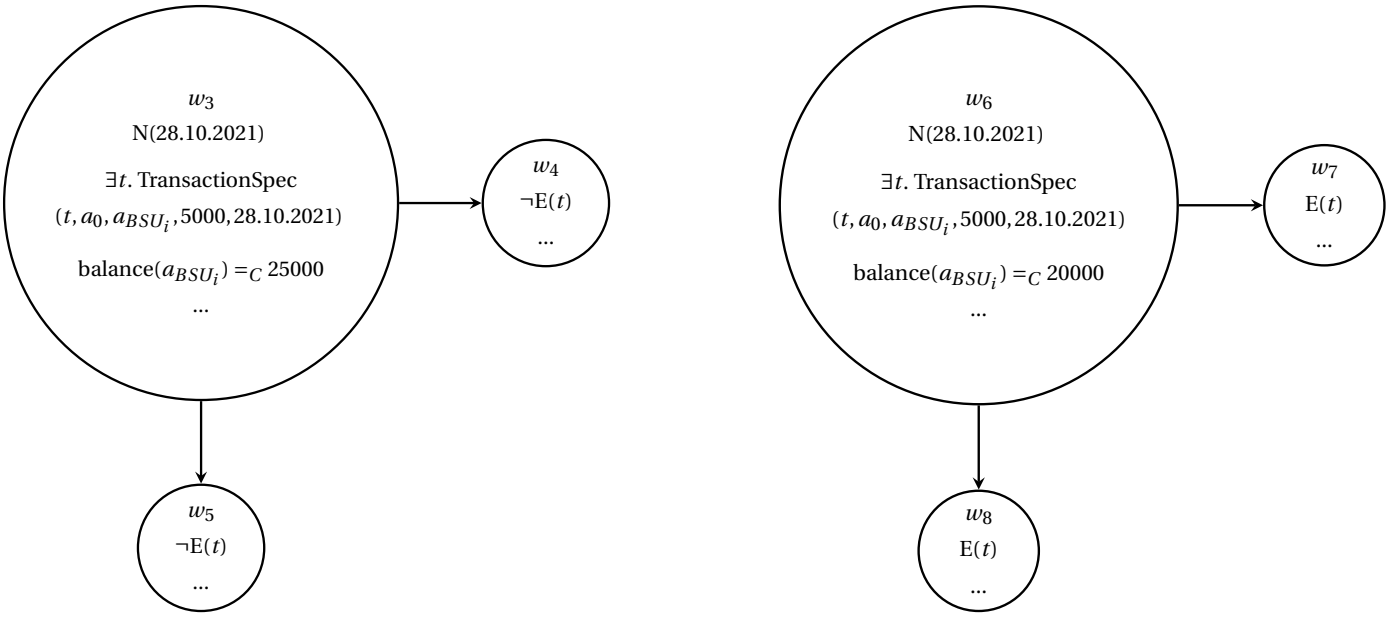


Figure 6.7: Expansion of example in Figure 6.6.

In w_3 and w_4 in 6.7, the current time is the time of the transaction t. In w_3, we see that the transaction will not be executed, as it does not satisfy the criteria for maximum balance of a BSU account:

$$\text{balance}_{max}(\text{target}(t)) \geqslant \text{balance}(\text{target}(t)) + \text{amount}(t)$$
$$\text{balance}_{max}(a_{BSU_i}) \geqslant 25000 + 5000$$
$$27500 \geqslant 30000$$

From axiom (BSU) we have that the max balance of the BSU the current year is 27.500NOK. Obviously, 27.500 is less than 30.000, and we have a contradiction here. On the other hand, in w_6, we see that there has been a withdrawal from the account since the specification of the transaction:

$$\text{balance}_{max}(\text{target}(t)) \geqslant \text{balance}(\text{target}(t)) + \text{amount}(t)$$
$$\text{balance}_{max}(a_{BSU_i}) \geqslant 20000 + 5000$$
$$27500 \geqslant 25000$$

In w_6, the criteria for the account type is satisfied since the balance has decreased, and as a consequence, the transaction will be executed.

6.5 Summary

In this chapter, we discussed all the axioms in Section 5.2. We discussed their significance, as well as how they relate to each other. The axioms serve as a correctness specification for our system, meaning that a program is correct if its output aligns with the specification [40].

Chapter 7

Conclusions, Discussion, and Recommendations for Further Work

In this final chapter, we summarize the process and the results. We will discuss the findings, what went well and what the challenges were. Lastly, we discuss some possibilities and recommendations for further work.

7.1 Summary and Conclusions

In Chapter 1, we listed our objectives for the project.

The first objective is related to the knowledge that was necessary for the project. This objective was to investigate the use of formal verification and modal logic to reason about information systems. We investigated how formal verification had been used previously, and focused on how modal logic has been applied in these cases. Some previous work and background are presented in Chapters 1 and 2.

Further, we investigated the functionality of online banks, and specified an access control model for an online banking system, our second objective. The methods used are outlined in Chapter 3. In Chapter 4, we discussed what the rules are in such a system, and why these were necessary. We implement a prototype and some of its functions in Haskell to demonstrate some of its intended functionality.

In Chapter 5, we introduced our formal language. We defined a first order modal language intended to be used to reason about our previously defined system, which is the next objective.

This chapter includes the syntax and semantics of our formal language.

The following objective was translate the rules of our system into our modal language, in the form of an axiomatic system using our first order modal logic syntax. These axioms are presented in Section 5.2. This was an iterative process, and axioms were added as the need transpired. We implemented the syntax and defined the axioms in Agda, in order to type check and verify that the axioms were syntactically correct.

The last objective was to formally reason about the functionality and completeness of the access control model, and how well our approach using first order modal logic was suitable for reasoning about access control of online banking systems. We did this by considering specific examples and demonstrating how our axiomatic system supported the intended functionality in Chapter 6.

The goal of the project was to investigate to what extent our method could be used to reason about access control. Through an iterative research and development process, we conclude that our approach represents a sufficient starting point for further applications of formal logic in this sector. What remains is to apply the approach to a greater extent to the concrete act of formally verifying systems.

7.2 Discussion

Our formal language for this project is a modal language, with the underlying propositional logic replaced by first order logic. Using a combination of first order logic and modal logic allowed us to express more functionality than using either of these individually would have. Having the modal operator `says` made it quite straight forward to model what the intentions of the actors in the system were, regardless of whether what they wanted to happen was an actual possibility at the time. Combining the modal operators `says` and `observes` allows us to easily model users delegating privileges to other users, and allows us to specify both accounts and whether the privileges include making transfers from the accounts or just view the activity on the accounts.

First order logic allows us to introduce predicates into our model. As an example, this lets us model time in the sense of whether or not a time is in the past. The use of predicates allows us to compare variables to each other. Particularly being able to model whether two variables are equal to one another is a practical expressive quality. Having the possibility of relations and functions also allows us to model relationships between variables. Through quantification, first order logic lets us express whether we are talking about all, none, just some or not all of some

object.

Combining these two forms of logic has its benefits. For example, it is very practical to be able to express that at some point in the future, there will exist an object where some particular formula is the case. It also allows us to express features such as that all identities that are the owner of an account are able to execute transactions from the account.

7.2.1 Limitations

The execution of this project was limited by multiple factors.

For this particular type of system, we face some challenges with first order modal logic, which mostly have to do with the expressiveness of the language. Although it is a powerful tool, its expressive power does have some limitations. One of these challenges being tracking changes over time, which is due to time being continuous as opposed to discrete. For example, keeping track of balance is not easily done with our language, because of its lacking ability to count and keep track of changes over time. We decided that keeping track of the balance of each account was not essential, as this is not directly related to access control, although it would have been a nice feature to have and allow us to reason about the system as a whole to a greater extent.

Another challenge is that we are unable to talk about a time interval and specific points in time in a simple manner, which is relevant when talking about accounts with a maximum amount of withdrawals or maximum sum of deposit per year. When it comes to maximum amount of withdrawals, this is also an example for how it would be helpful to be able to more easily quantify a specific number of objects. The best way to formally verify these features kinds are not self-evident.

Also serving as a significant obstacle for this project was our lack of knowledge of the domain and specifics of online banking, and accessibility to this information. After reviewing some literature, it became apparent that specifics about these kinds of systems are not publicly available. This made it difficult to know whether our design specifications were coherent with the functionality of existing systems and their requirements. Banking systems are complicated and in detail can be difficult to understand the specifics of. For example, as we mentioned in Section 4.2.1, it is not clear who is the owner of a loan account for the different types of loans. Without having this knowledge available, it was not possible to get an overview of what the missing knowledge was, and additionally makes it difficult to assess to what extent this affected what we were trying to do. In general, building a logical specification of a system is not an obvious task,

which is also described as a significant obstacle for untrained users in [37].

A common problem with the model checking approach is state space. This problem refers to the exponential growth of system state space as the number of state variables in the system increases [14]. This is mostly relevant to this project due to the time restriction. A system with considerably fewer variables would be much simpler to exhaustively model. A significant improvement to this section would be automating a process for model checking using our approach.

Lastly, time constraints limited the scope of our implementation. This prevented us from being able to explore whether other methods and approaches could have been more suitable for this project, as well as inclusion of more features. Additionally, we would have preferred for Chapter 6 to be more elaborate and for each feature to be explicitly modelled formally.

7.3 Recommendations for Further Work

Our exploration of application of access control logics to online banking systems opens up possibilities for further work. In this section, we discuss some of these possibilities, categorizing them into sections depending on how far into the future we consider them to be.

7.3.1 Short Term

The main recommendation short term is to involve someone with domain knowledge of banking and online banking systems, as we think this project would have more potential if we had access to better knowledge of the domain. Involving a domain expert who possesses detailed knowledge regarding how these systems work would allow for a better assessment of how realistic the design of our system is, and how well it takes into consideration and covers situations that occur in real life. This would also be advantageous when deciding on what to focus on for medium term goals, and give an idea of what approaches can be worth exploring.

7.3.2 Medium Term

Functionality

Currently, we are not able to reason about changes of currency using our formal language and axioms. Ideally, it would be beneficial to have the possibility of keeping track of balances of the various accounts and changes over time. In order to do this, we would need to encode mathematics in our first order logic. There is no simple way to do this, but some similar work is presented in [58].

As mentioned in Section 1.2, we might also want to expand our formalisation of delegation of permissions to allow representation of more specific restrictions. For example, an owner of an account allowing others to make transactions from the account of only up to a certain amount. Other things that could be included are actions such as opening and closing of accounts, and creation and termination of users.

Following from the suggestion in the previous section, including someone with knowledge of the domain may also help with identifying other functionality that is currently lacking.

Approach

Another medium term goal would be to explore different approaches to formal verification. One of the challenges of this project was the time constraint, which did not allow room to investigate different approaches of formal verification. Within modal logic, it would be worth exploring whether a different, or more inclusive, set of modalities could give better expressiveness.

For this point, it is advantageous to have previously included a domain expert in the process, as mentioned in the section above. This would hopefully give an idea about whether there are more suitable approaches to reason about banking systems, as we would have a better picture of what is lacking from the current structure, and what remains to be modeled.

Automation

Doing proofs and verifications manually is a time consuming and error prone process. Therefore, it would be helpful to automate this process using some sort of proof assistant or model checker. This could, for example, be done by extending our implementation of the syntax in Agda. There is a number of model checking tools available, although it is uncertain if any are suitable for this project.

7.3.3 Long Term

The benefits of formal verification have been discussed throughout the thesis. Long term, a goal is for the use of formal verification in the development process to be more widespread. In the development process of any system, formal verification to ensure that the system conforms to the design specification should be a natural step to take in the process of developing secure systems. In order for this to be achievable, formal verification should be more straight forward to apply. This could be done by developing a framework simplifying and automating this process.

References

[1] A. Abel et al. Quick guide to editing, type checking and compiling agda code, 2017. URL https://agda.readthedocs.io/en/v2.5.4.1/getting-started/quick-guide.html.

[2] A. Abel et al. What is agda?, 2019. URL https://agda.readthedocs.io/en/v2.6.0.1/index.html.

[3] K. Amadeo. What Is Banking? *The Balance*, 2021. URL https://www.thebalance.com/what-is-banking-3305812.

[4] R. Ballarin. Modern Origins of Modal Logic. In E.N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition, 2021.

[5] Norges Bank. Retail payment services 2019, 2020. URL https://www.norges-bank.no/contentassets/b5b8151d59fe48b3bae68b6e5cb99971/papers_1_20_report.pdf?v=05/18/2020172432&ft=.pdf.

[6] BankID. Generelt om BankID, n.d. URL https://www.bankid.no/privat/los-mitt-bankid-problem/ofte-stilte-sporsmal/generelt-om-bankid/.

[7] R. Barbuti, N. De Francesco, A. Santone, and G. Vaglini. Reduced Models for Efficient CCS Verification. *Formal Methods in System Design*, 26(3):319–350, May 2005. ISSN 1572-8102. doi: 10.1007/s10703-005-1634-6. URL https://doi.org/10.1007/s10703-005-1634-6.

[8] C. Benzmüller. Automating Access Control Logics in Simple Type Theory with LEO-II. *IFIP Advances in Information and Communication Technology*, page 387–398, 2009. ISSN 1868-422X. doi: 10.1007/978-3-642-01244-0_34. URL http://dx.doi.org/10.1007/978-3-642-01244-0_34.

[9] P. Blackburn, J. van Benthem, and F. Wolter, editors. *Handbook of modal logic*. Number 3 in Studies in logic and practical reasoning. Elsevier, Amsterdam, 1. ed., reprinted edition, 2007. ISBN 978-0-444-51690-9. OCLC: 255545888.

[10] G. Bonanno. Modal logic and game theory: Two alternative approaches. *Risk, Decision and Policy*, 7:309–324, 12 2002. doi: 10.1017/S1357530902000704.

[11] R.J. Buehler. First-Order Modal Logic, 2015.

[12] W. Carnielli and C. Pizzi. Multimodal logics. In *Modalities and Multimodalities*. Springer, Dordrecht, January 2008. ISBN 978-1-4020-8589-5. doi: 10.1007/978-1-4020-8590-1_8.

[13] D.C. Chou and A.Y. Chou. A Guide to the Internet Revolution in Banking. *Information Systems Management*, 17(2):47–53, March 2000. ISSN 1058-0530, 1934-8703. doi: 10.1201/1078/43191.17.2.20000301/31227.6. URL http://www.tandfonline.com/doi/abs/10.1201/1078/43191.17.2.20000301/31227.6.

[14] E.M. Clarke, W. Klieber, M. Nováček, and P. Zuliani. Model Checking and the State Explosion Problem. In B. Meyer and M. Nordio, editors, *Tools for Practical Software Verification: LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures*, pages 1–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35746-6. doi: 10.1007/978-3-642-35746-6_1. URL https://doi.org/10.1007/978-3-642-35746-6_1.

[15] E.M. Clarke, T.A. Henzinger, and H Veith. *Handbook of Model Cheking*, chapter Introduction to Model Checking, pages 1–22. Springer, Cham, 2018.

[16] L. Collins. Chapter 11 - Access Controls. In J.R. Vacca, editor, *Cyber Security and IT Infrastructure Protection*, pages 269–280. Syngress, Boston, 2014. ISBN 978-0-12-416681-3. doi: https://doi.org/10.1016/B978-0-12-416681-3.00011-2. URL https://www.sciencedirect.com/science/article/pii/B9780124166813000112.

[17] F. Cuppens and R. Demolombe. *Deontic Logic, Agency and Normative Systems*, chapter A Deontic Logic for Reasoning about Confidentiality, pages 66–79. Springer, 1996. Workshops in Computing.

[18] F. Cuppens and R. Demolombe. A Modal Logical Framwork for Security Policies. Technical report, ONERA-CERT, 2 Avenue E. Belin, 31055, Toulouse Cedex, France, 1997. Proceedings of the 10th International Symposium on Foundations of Intelligent Systems, Lecture Notes in Computer Science, Vol.1325. Springer.

[19] CVE. CVE list, 2021. data retrieved from CVE, https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=bank.

[20] D. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[21] DNB. Gjør det selv, n.d. URL https://www.dnb.no/bedrift/hjelp-og-veiledning/gjor-det-selv.

[22] J. Emmitt. Cybersecurity Is a Top Priority - What to Do About It?, 2020. URL https://www.kaseya.com/blog/2020/11/09/cybersecurity-is-a-top-priority/.

[23] D. Ferraiolo, D.R. Kuhn, and R. Chandramouli. *Role-based access control*. Artech house, 2003.

[24] M. Fitting and R.L. Mendelsohn. *First-Order Modal Logic*. Kluwer Academic Publishers, 1998.

[25] The OWASP Foundation. A01:2021 - Broken Access Control, 2021. URL https://owasp.org/Top10/A01_2021-Broken_Access_Control/.

[26] The OWASP Foundation. OWASP Top Ten Web Application Security Risks | OWASP, 2021. URL https://owasp.org/www-project-top-ten/.

[27] The OWASP Foundation. About the OWASP Foundation, n.d. URL https://owasp.org/about/.

[28] D. Garg and M. Abadi. A Modal Deconstruction of Access Control Logics. In R. Amadio, editor, *Foundations of Software Science and Computational Structures*, pages 216–230, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-78499-9.

[29] J. Garson. Modal Logic. In E.N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2021 edition, 2021.

[30] B. Guram and S.M. Lawrence. Introduction to Automata and Complexity Theory, Lecture notes: Undecidability of First-Order Logic, 2002.

[31] H. Gylterud. UiB Software Security, Lecture notes: Access control, 2020.

[32] M. Hill and D. Swinhoe. The 15 biggest data breaches of the 21st century, 2021. URL https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html.

[33] B. Holzhauer. Digital Banking As The New Normal In 2021: What To Expect From Banks. *Forbes*, 2021. URL https://www.forbes.com/advisor/banking/digital-banking-as-new-normal-2021-what-to-expect/.

[34] Imperva. What is Access Control List | ACL Types & Linux vs Windows | Imperva, n.d. URL https://www.imperva.com/learn/data-security/access-control-list-acl/.

[35] H.D. Johnson. What is a joint bank account? *Bankrate*, 2019. URL https://www.bankrate.com/banking/what-is-a-joint-bank-account/.

[36] P.N. Johnson-Laird. Deductive reasoning. *Annual Review of Psychology*, 50(1):109–135, 1999. doi: 10.1146/annurev.psych.50.1.109. URL https://doi.org/10.1146/annurev.psych.50.1.109. PMID: 15012459.

[37] R. Klimek. Deduction-Based Formal Verification of Requirements Models with Automatic Generation of Logical Specifications. In L.A. Maciaszek and J. Filipe, editors, *Evaluation of Novel Approaches to Software Engineering*, pages 157–171, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-45422-6.

[38] C. Kong, M. Gao, W. Qian, M. Zhou, X. Gong, R. Zhang, and A. Zhou. ACID Encountering the CAP Theorem: Two Bank Case Studies. In *2015 12th Web Information System and Application Conference (WISA)*, pages 235–240, 2015. doi: 10.1109/WISA.2015.63.

[39] T. Kosiyatrakul, S. Older, and S. Chin. A Modal Logic for Role-Based Access Control. Technical report, Syracuse University, 2005.

[40] L. Lambert. Modal logic in computer science. Master's thesis, Rochester Institute of Technology, 2006. URL https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=7898&context=theses.

[41] E. Landwehr. Formal Models for Computer Security. Technical report, Naval Research Laboratory, Washington D.C., 1981.

[42] C.W. Lee. Axiomatic systems. Technical report, Department of Mathematics, University of Kentucky, 1997.

[43] O. Levin. Discrete mathematics: An open introduction. Technical report, School of Mathematical Science, University of Northern Colorado, 2019.

[44] Lexico. Modality, n.d. URL https://www.lexico.com/definition/modality?locale=en.

[45] M. Lipovača. *Learn You a Haskell for Great Good! A Beginner's Guide*. No Starch Press, USA, 1st edition, 2011. ISBN 1593272839.

[46] F. Miklós. Matematikai logika (in hungarian), 2002.

[47] R. Milner. *A Calculus of Communicating Systems*. Lecture Notes in Computer Science. Springer-Verlag, Berlin Heidelberg, 1980. ISBN 978-3-540-10235-9. doi: 10.1007/3-540-10235-3. URL https://www.springer.com/gp/book/9783540102359.

[48] R.S. Mohanta. Evolution and Shift in Trend of Cyber Crime: An Overview. *Cyber Times International Journal of Technology & Management*, 10, 2017.

[49] A. Mpagouli and I. Hatzilygeroudis. Converting first order logic into natural language: A first level approach. Technical report, University of Patras, School of Engineering Department of Computer Engineering & Informatics, 01 2007.

[50] U. Norell. *Towards a practical programming language based on dependent type theory*. PhD Thesis, Chalmers University of Technology, 2007.

[51] U. Norell and J. Chapman. Dependently typed programming in agda. Technical report, Chalmers University of Technology, 2013.

[52] Finans Norge. De fleste bruker nettbank - også de eldre, 2017. URL https://www.finansnorge.no/aktuelt/nyheter/forbruker-og-finanstrender/forbruker--og-finanstrender-2017/de-fleste-bruker-nettbank--ogsa-de-eldste/.

[53] Bank of Scotland. Delegated authority, n.d. URL https://business.bankofscotland.co.uk/3m-25m-turnover/online-banking/delegated-authority.html.

[54] M. Ouimet. Formal Software Verification: Model Checking and Theorem Proving. Technical report, Massachusetts Institute of Technology, 2007.

[55] R.S. Sandhu. Role-based Access Control. In M.V. Zelkowitz, editor, *Advances in Computers*, volume 46, pages 237–286. Elsevier, 1998. doi: https://doi.org/10.1016/S0065-2458(08)60206-5. URL https://www.sciencedirect.com/science/article/pii/S0065245808602065. ISSN: 0065-2458.

[56] A. Sanghavi. What is formal verification? *EE Times Asia*, 2010.

[57] A. Santone, V. Intilangelo, and D. Raucci. Efficient Formal Verification in Banking Processes. In *2013 IEEE Ninth World Congress on Services*, pages 325–332, 2013. doi: 10.1109/SERVICES.2013.79.

[58] N. Schwikardt. *Arithmetics, First-Order Logic, and Counting Quantifiers*. ACM, 2004.

[59] M. Sergot. Modal and temporal logic, lecture notes: Systems of modal logic, 2008.

[60] S.A. Seshia. Computer-Aided Verification, Lecture notes: Intro. to Model Checking: Models and Properties, 2011.

[61] ITP Staff. Digitalisation in banking: Key opportunities and challenges you need to know. *ITP.net*, 2021. URL https://www.itp.net/business/95962-digitalisation-in-banking-key-challenges-and-opportunities-you-need-to-know.

[62] Ekran System. Mandatory Access Control vs Discretionary Access Control: Which to Choose?, March 2020. URL https://www.ekransystem.com/en/blog/mac-vs-dac.

[63] S. Toida. Discrete Structures/Discrete Mathematics, Lecture notes: Introduction to Reasoning, 2009.

[64] S. Toida. System Security, Lecture notes: Discretionary Access Control, 2015.

[65] S. Toida. Logic and Proof, Lecture notes: First Order Logic, 2016.

[66] J. van Benthem. Modal Logic: A Contemporary View, n.d. URL https://iep.utm.edu/modal-lo/.

[67] M.Y. Vardi. Logic in Computer Science and Artificial Intelligence, Lecture notes: Semantics of First Order Logic, 2005.

[68] R. von Solms and J. van Niekerk. From information security to cyber security. *Computers & Security*, 38:97–102, 2013. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2013.04.004. URL https://www.sciencedirect.com/science/article/pii/S0167404813000801.

[69] M. Walicki. *Introduction to Mathematical Logic*. World Scientific Publishing Co Pte Ltd, 2016.

[70] E.N. Zalta. Basic Concepts in Modal Logic. Technical report, Center for the Study of Language and Information, Stanford University, 1995.

Appendix A

Bank Prototype Implementation in Haskell

```
import Data.Time
import Data.Map
import Data.Tuple
import Control.Monad.State
import Data.List

data Account = Account {owner :: [Identity], balance :: Currency} deriving Eq
data Identity = User {dob :: Date} deriving Eq
data Transaction = Transaction {amount :: Currency, source :: Account,
    target :: Account, description :: String, tt :: Time, td :: Date}
data Bank = Bank {accounts :: Map Int Account, transactions :: Map Int Transaction,
    users :: Map Int Identity}

type Date = Day
type Time = TimeOfDay
type Currency = Integer

data BankState = BankState
    { trans :: [Transaction],
    ids :: [Identity],
    accs :: [Account]
    }
```

```
-- undefined: used for demonstration to say that a date has passed
past :: Date -> Bool
past = undefined

-- basic rules applying to all types of accounts
isValid :: Transaction -> Bool
isValid (Transaction amount a0 a1 _ _ _) = a0 /= a1 && not(past d)

-- check for accounts with maximum allowed yearly deposit
maxDeposits :: Integer -> Transaction -> [Transaction] -> Bool
maxDeposits max (Transaction amount _ target _ _ date) prevTransactions
                = amountIn prevTransactions target year 0
                - amountOut prevTransactions target year 0 + amount <= max
                    where year = getYear date

-- check for accounts with maximum number of withdrawals yearly
maxWithdrawals :: Integer -> Integer -> Transaction -> [Transaction] -> Bool
maxWithdrawals max year (Transaction amount source _ _ _ date) transactions =
                noWithdrawals transactions source year 0 /= max
-- retrieves the year of a date
getYear :: Date -> Integer
getYear d = year where (year,month,day) = toGregorian d
-- retrieves amount into an account for a given year
amountIn :: [Transaction] -> Account -> Integer -> Integer -> Integer
amountIn [] _ _ sum = sum
amountIn ((Transaction prevAmount _ a1 _ _ date):xs) acc year sum
    | a1 == acc && checkYear year date = amountIn xs acc year sum+prevAmount
    | otherwise = amountIn xs acc year sum
-- retrieves amount out of account for a given year
amountOut :: [Transaction] -> Account -> Integer -> Integer -> Integer
amountOut [] _ _ sum = sum
amountOut ((Transaction prevAmount a0 _ _ _ date):xs) acc year sum
    | a0 == acc && checkYear year date = amountOut xs acc year prevAmount+sum
    | otherwise = amountOut xs acc year sum
```

```haskell
-- checks if a date is in a given year
checkYear :: Integer -> Date -> Bool
checkYear year date = getYear date == year
-- checks number of withdrawals from an account for a given year
noWithdrawals :: [Transaction] -> Account -> Integer -> Integer -> Integer
noWithdrawals [] _ _ sum = sum
noWithdrawals ((Transaction _ source _ _ _ date):xs) acc year sum
    | source == acc && checkYear year date = noWithdrawals xs acc year sum+1
    | otherwise = noWithdrawals xs acc year sum
-- retrieves owner of an account
getOwner :: Account -> [Identity]
getOwner (Account owner _) = owner
-- returns balance of a given account from set of transactions
getBalance :: [Transaction] -> Account -> Currency
getBalance [] _ = 0
getBalance ((Transaction amount source target _ _ _):ts) a
    | intersect (getOwner source) (getOwner a) /= [] = (-amount) + getBalance ts a
    | intersect (getOwner target) (getOwner a) /= [] = amount + getBalance ts a
    | otherwise = getBalance ts a
```

Appendix B

Syntax Implementation in Agda

```
module lang where

open import Premises.Logic hiding (⊤ ; ⊥ ; ¬; lift; _+_)
open import Premises.Natural hiding (_≤_; _+_)

infixr 100 _∧_

data Sort : Set where
    Identity : Sort
    Account  : Sort
    Transaction : Sort
    Date : Sort
    Currency : Sort
    AccountType : Sort

Context : Set₁
Context = Sort → Set

data _◁_ (Γ : Context) (s : Sort) : Sort → Set where
    old : ∀ {s'} -> Γ s' → (Γ ◁ s) s'
    new : (Γ ◁ s) s

_◁[_]_ : (Γ : Context) → N → Sort → Context
```

```
Γ ◁[ zero ] s = Γ
Γ ◁[ succ n ] s = (Γ ◁[ n ] s) ◁ s

-- Finite sets
data Fin : N → Set where
  new : ∀{n} → Fin (succ n)
  old : ∀{n} → Fin n → Fin (succ n)

-- Functions for indexing in big-functions

toN : ∀{n} → (Fin n) → N
toN new = zero
toN (old x) = succ (toN x)

include : ∀{n} (i : Fin n) → (Fin (toN i)) → Fin n
include new ()
include (old x) new = new
include (old x) (old y) = old (include x y)



ix : ∀ {Γ s} n → Fin n → (Γ ◁[ n ] s) s
ix zero ()
ix (succ n) new = new
ix (succ n) (old k) = old (ix n k)

data Term (Γ : Context) : Sort → Set where
    var : ∀ {s} → Γ s → Term Γ s
    source  : Term Γ Transaction → Term Γ Account
    target  : Term Γ Transaction → Term Γ Account
    td      : Term Γ Transaction → Term Γ Date
    amount  : Term Γ Transaction → Term Γ Currency
    balance : Term Γ Account     → Term Γ Currency
    at      : Term Γ Account     → Term Γ AccountType
    maxb    : Term Γ Account     → Term Γ Currency
```

```
    minb    : Term Γ Account     → Term Γ Currency
    _-_     : Term Γ Currency → Term Γ Currency → Term Γ Currency
    _+_     : Term Γ Currency → Term Γ Currency → Term Γ Currency
    by      : Term Γ Date

-- Lifting
lift : {Γ : Context} {s u : Sort} → Term Γ s →  Term (Γ ◁ u) s
lift (var v) = var (old v)
lift (source t) = source (lift t)
lift (target t) = target (lift t)
lift (td t) = td (lift t)
lift (amount t) = amount (lift t)
lift (balance t) = balance (lift t)
lift (at t) = at (lift t)
lift (maxb a) = maxb (lift a)
lift (minb a) = minb (lift a)
lift (a + b) = (lift a) + lift (b)
lift (a - b) = (lift a) - lift (b)
lift by = by

data Formula (Γ : Context) : Set where
    -- Logic
    _==_ : ∀ {s} → Term Γ s → Term Γ s → Formula Γ
    _∨_ : Formula Γ → Formula Γ → Formula Γ
    _∧_ : Formula Γ → Formula Γ → Formula Γ
    _⇒_ : Formula Γ → Formula Γ → Formula Γ
    ¬_ : Formula Γ → Formula Γ
    ⊥ : Formula Γ
    ⊤ : Formula Γ
    ∀' : ∀ s → Formula (Γ ◁ s) → Formula Γ
    ∃' : ∀ s → Formula (Γ ◁ s) → Formula Γ
    mw : ℕ → Term Γ Account → Formula Γ
    -- Modalities
    -- Time
```

```
    □_ : Formula Γ → Formula Γ
    ◇_ : Formula Γ → Formula Γ
    -- Intention
    _says_ : Term Γ Identity → Formula Γ → Formula Γ
    -- Observation
    _observes_ : Term Γ Identity → Formula Γ → Formula Γ
    -- Time related
    Past : Term Γ Date → Formula Γ
    Executed : Term Γ Transaction → Formula Γ
    _views_ : Term Γ Identity → Term Γ Account → Formula Γ
    _inits_ : Term Γ Identity → Term Γ Account → Formula Γ
    _before_ : Term Γ Date → Term Γ Date → Formula Γ
    -- Currency
    _≤_ : Term Γ Currency → Term Γ Currency → Formula Γ
    _owns_ : Term Γ Identity → Term Γ Account → Formula Γ

-- Big-functions


big-∨ : ∀{Γ}
      → (k : ℕ)
      → ((Fin k) → Formula Γ)
      → Formula Γ
big-∨ zero ϕ = ⊥
big-∨ (succ k) ϕ = ϕ new ∨ big-∨ k (ϕ ∘ old)

big-∧ : ∀{Γ}
      → (k : ℕ)
      → ((Fin k) → Formula Γ)
      → Formula Γ
big-∧ zero ϕ = ⊤
big-∧ (succ k) ϕ = ϕ new ∧ big-∧ k (ϕ ∘ old)

big-∀ : ∀{Γ} n s → Formula (Γ ◁[ n ] s) → Formula Γ
```

```
big-∀ zero s φ = φ
big-∀ (succ n) s φ = (big-∀ n s (∀' s φ))

-- Lifting in big
big-lift : {Γ : Context} {s u : Sort} {n : N}
         → Term Γ s →  Term (Γ ◁[ n ] u) s
big-lift {n = zero} t = t
big-lift {n = succ k} t = lift (big-lift t)

-- Axioms

TransactionSpec : ∀{Γ} → Term Γ Transaction
                → Term Γ Account
                → Term Γ Account
                → Term Γ Currency
                → Term Γ Date
                → Formula Γ
TransactionSpec t a₀ a₁ c d = (source t == a₀)
                            ∧ (target t == a₁)
                            ∧ (amount t == c)
                            ∧ (td t == d)

Now : ∀{Γ} → Term Γ Date → Formula Γ
Now d₀ = (Past d₀) ∧ ∀' Date (((Past d) ⇒ ((d before (lift d₀)) ∨ ((lift d₀) == d))))
    where
    d = var new

isValid : ∀{Γ} → Term Γ Account → Term Γ Account → Term Γ Date → Formula Γ
isValid a₀ a₁ d = (¬(Past d)) ∧ (¬(a₀ == a₁))

-- First order logic
valid-accounts : ∀ {Γ} → Formula Γ
valid-accounts = ∀' Transaction (¬ ((source t) == (target t))) where
    t = var new
```

```
-- Modal logic
four : ∀ {Γ} → (ϕ : Formula (Γ)) → Formula Γ
four ϕ = (□ ϕ) ⇒ (□ (□ ϕ))

t-axiom : ∀ {Γ} → (ϕ : Formula (Γ)) → Formula Γ
t-axiom ϕ = ((□ ϕ) ⇒ ϕ)

d-axiom : ∀ {Γ} → (ϕ : Formula (Γ)) → Formula Γ
d-axiom ϕ = ((□ ϕ) ⇒ (◇ ϕ))

five : ∀ {Γ} → (ϕ : Formula (Γ)) → Formula Γ
five ϕ = ((◇ ϕ) ⇒ (□ (◇ ϕ)))

-- says
unit : ∀ {Γ} → (ϕ : Formula (Γ ◁ Identity)) → Formula Γ
unit ϕ = ∀' Identity (ϕ ⇒ (i says ϕ)) where
    i = var new

cuc : ∀ {Γ} → (ϕ ψ : Formula (Γ ◁ Identity)) → Formula Γ
cuc ϕ ψ = ∀' Identity ((i says (ϕ ⇒ ψ)) ⇒ ((i says ϕ) ⇒ (i says ψ))) where
    i = var new

idem : ∀ {Γ} → (ϕ : Formula (Γ ◁ Identity)) → Formula Γ
idem ϕ = ∀' Identity ((i says (i says ϕ)) ⇒ (i says ϕ)) where
    i = var new

-- Time
time-passes : ∀ {Γ} → Formula Γ
time-passes = ∀' Date (◇ (Past d)) where
    d = var new

time-continous : ∀ {Γ} → Formula Γ
time-continous = ∀' Date (∀' Date ((d₀ before d₁) ⇒ (Past d₁ ⇒ Past d₀))) where
```

```
    d₀ = var (old new)
    d₁ = var new

time-linear : ∀ {Γ} → Formula Γ
time-linear = ∀' Date (Past d ⇒ (□ (Past d))) where
    d = var new

-- First order modal logic
-- Initialize permissions
init-owner : ∀ {Γ} → Formula Γ
init-owner = ∀' Account (∀' Identity ((i owns a) ⇒ (i inits a)))
    where
    a = var (old new)
    i = var new

init-del : ∀ {Γ} → Formula Γ
init-del = ∀' Account (∀' Identity (∀' Identity ((i owns a) ∧ ((i says (i' inits a))
    ⇒ (i' inits a)))))
    where
    a = var (old (old new))
    i = var (old new)
    i' = var new

initiate : ∀ {Γ} → Formula Γ
initiate = ∀' Account (∀' Account (∀' Currency (∀' Date (∀' Identity (
    ((i inits a₀) ∧ (isValid a₀ a₁ d) ∧
    (i says (∃' Transaction (TransactionSpec t (lift a₀) (lift a₁) (lift c) (lift d)))))
    ⇒ (◇ (∃' Transaction(TransactionSpec t' (lift a₀) (lift a₁) (lift c) (lift d))))
    )))))
    where
    a₀ = var (old (old (old (old new))))
    a₁ = var (old (old (old new)))
    c = var (old (old new))
    d = var (old new)
```

```
    i = var new
    t = var new
    t' = var new

-- View permissions
view-owner : ∀ {Γ} → Formula Γ
view-owner = ∀' Account (∀' Identity ((i owns a) ⇒ (i views a)))
    where
    a = var (old new)
    i = var new

view-del : ∀ {Γ} → Formula Γ
view-del = ∀' Account (∀' Identity (∀' Identity ((i owns a) ∧ ((i says (i' views a))
    ⇒ (i' views a)))))
    where
    a = var (old (old new))
    i = var (old new)
    i' = var new

observe : ∀ {Γ} → Formula Γ
observe = ∀' Transaction (∀' Identity (∀' Account (
    ((i views a) ∧ (((source t) == a) ∨ (((target t) == a) ∧ (Executed t))))
    ⇒ (i observes (∃' Transaction (TransactionSpec t'
    (source (lift t)) (target (lift t)) (amount (lift t)) (td (lift t)))))
    )))
    where
    t = var (old (old new))
    i = var (old new)
    a = var new
    t' = var new

-- Execute
exec : ∀ {Γ} → N → Formula Γ
exec k = ∀' Transaction ((Now (td t)) ⇒ (
```

```
        (((amount t) ≤ (balance(source t)))
        ∧ (((balance(target t)) + (amount t)) ≤ (maxb(target t)))
        ∧ (((minb(target t)) + (amount t)) ≤ (balance(source t)))
        ∧ (mw k (source t)
        ∧ big-∀ k Transaction (
            (big-∧ k (λ i → ((by before td(var (ix k i))) ∧ (Executed (var (ix k i)))
            ∧ ( (source (var (ix k i))) == (source (big-lift t) ) ))))
            ⇒ (big-∨ k (λ i → big-∨ (toN i)
            (λ j → (var (ix k i) == var (ix k (include i j))))))
        )))
       ⇒ (□ (Executed t))))
    where
    t = var new

-- Axioms regarding accounts following from (exec)
max-withdraw : ∀ {Γ} → N → Formula Γ
max-withdraw k = ∀' Account (mw k a ⇒ (big-∀ k Transaction (
            (big-∧ k (λ i → ((by before td(var (ix k i))) ∧ (Executed (var (ix k i)))
            ∧ ( (source (var (ix k i))) == (big-lift a)))))
            ⇒ (big-∨ k (λ i → big-∨ (toN i)
            (λ j → (var (ix k i) == var (ix k (include i j))))))
        )))
        where
        a = var new

min-balance : ∀ {Γ} → Formula Γ
min-balance = ¬ ∃' Account(balance(a) ≤ minb(a))
    where
    a = var new

max-balance : ∀ {Γ} → Formula Γ
max-balance = ¬ ∃' Account(maxb(a) ≤ balance(a))
    where
    a = var new
```