FCL: A FORMAL LANGUAGE FOR WRITING CONTRACTS

FCL: A FORMAL LANGUAGE FOR WRITING CONTRACTS

By

QIAN HU, M.Sc.

A Thesis

Submitted to the Department of Computing and Software

and the School of Graduate Studies

of McMaster University

in Partial Fulfilment of the Requirements

for the Degree of

Doctor of Philosophy

Doctor of Philosophy (2018) McMaster University

(Computing and Software) Hamilton, Ontario, Canada

TITLE:	FCL: A formal language for writing contracts
AUTHOR:	Qian Hu M.Sc. McMaster University, Hamilton, Ontario, Canada
SUPERVISOR:	Dr. William M. Farmer
NUMBER OF PAGES:	xi, 132

To my family

Abstract

Contracts are legally enforceable agreements between two or more parties. The agreements can contain temporally based conditions, such as actions taken by the contract parties or events that happen, that trigger changes to the state of the contract when the conditions become true. Since the structure of these conditions can be very complex, it can be difficult to write contracts in a natural language in a clear and unambiguous way. A better approach is to have a formal language with a precise semantics to represent contracts. Contracts expressed in such a language have a mathematically precise meaning and can be written, analyzed, and manipulated by software.

This thesis presents FCL, a formal language with a precise semantics for writing general contracts that may depend on temporally based conditions. Motivated by carefully selected examples of contracts, we derive a set of desirable requirements that a formal language of contracts should support. Based on the requirements, we clearly define the notion of contract and address what it means to fulfill or breach a contract. We present the formal syntax and semantics of FCL. We also successfully formalize different kinds of contracts in FCL and develop a reasoning system for FCL.

Acknowledgements

First and foremost, I would like to express my gratitude to my academic supervisor Dr. William M. Farmer for his unwavering support, invaluable guidance, and continuous encouragement throughout the development and advancement of my research studies and my life.

My special thanks go to the members of my supervisory committee: Dr. Jacques Carette, Dr. Ridha Khedri, and Dr. Tom Maibaum for all of their valuable advices and thoughtful comments on my research work. I also appreciate the help and moral support from all of other individuals whom I have interacted with throughout my graduate studies.

Furthermore, I want to extend my sincere thanks to the Natural Sciences and Engineering Research Council of Canada (NSERC) and McMaster University graduate scholarship which have provided the financial support during my doctoral studies.

Last, but certainly not least, I would like to thank my dear family and friends from the bottom of my heart for all of their love, support, and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A contract records, orally or in writing, a legally binding agreement between two or more parties [48]. It specifies obligations, permissions, and prohibitions for the parties involved in the contract. Although contracts are considered to be legally binding and can be very complex, they are usually expressed in a natural language such as English. Since the natural language does not have a precise semantics, it is often not clear what a contract is intending to say.

A major goal of our research is to help people write contracts that have a clear meaning and can be readily analyzed and simplified. One way of achieving this goal is by using formal methods, which are mathematically based languages, techniques, and tools to mechanically process complex contracts for various activities such as specifying, verifying, and analyzing contracts.

This thesis presents FCL, a formal language for writing general contracts. It is based on the joint research with my supervisor, William M. Farmer, published in [20] and [21]. The language has a formal syntax and a precise semantics and thus can be readable by computers, too. The main purpose of this language is to specify contracts

as unambiguously, completely, and succinctly as possible. This chapter introduces the problem domain of this thesis and argues the need for writing contracts in a formal language. More precisely, Section 1.1 motivates the need for a formal language with a precise semantics for expressing general contracts. Section 1.2 gives an overview of FCL and discusses the key idea behind the design of FCL. Section 1.3 summarizes the contributions of this thesis. Section 1.4 outlines the structure of the remainder of this thesis.

1.1 Motivation for the Research

1.1.1 What is a Contract?

A contract is an artifact with certain properties. There is not a clear consensus of which of these properties are necessary and which are optional. We favor the definition of a contract given by Brian Blum in [11, p. 2]. He says a contract must have each of the following properties:

P1 Is an oral or written agreement.

P2 Involves at least two parties.

P3 Includes at least one promise made by the parties.

P4 Establishes an exchange relationship between the parties.

P5 Is legally enforceable.

A contract is created only because the parties reach agreement on the terms of the contract (P1). The *parties* are the people or entities that have mutually agreed

to the contract and are bound by its terms and conditions. For any contract to be valid, there must be at least two parties (P2). Typically, one party makes an offer and the other party accepts it. In addition, to be valid a contract must involve the parties in an *exchange* of something of value such as services, goods, or a promise to perform some action (P4). Note that the exchange of money is not necessary.

A contract involves a *promise* (P3) which Blum defines as an "undertaking to act or refrain from acting in a specified way at some future time" [11, p. 5]. We think of "undertaking to act" as the deontic notion of *obligation*. Similarly, we understand "refrain from acting" as the deontic notion of *prohibition*. Obligation and prohibition are concepts studied in deontic logic [42].

We will use an expanded definition of a contract that includes "degenerate contracts" that would not be considered contracts according to Blum's definition but are convenient to include in the space of all possible contracts. For example, a contract is *void* if it violates the law [7]. Void contracts are not legally enforceable agreements, so by Blum's definition they are not genuine contracts (P5). We will consider them to be contracts, but we will designate them as being degenerate. Similarly, we will consider an agreement between two parties that does not include a promise or establish an exchange relationship between the parties as a degenerate contract.

1.1.2 Example: a Contract with Observables and Conditions

A contract records a legally binding agreement between parties. The meaning of a contract — that is, what the agreement is — often depends on certain things that can be observed, called *observables*, such as actions taken by the parties of the contract, events that happen, and values (like a stock price) that fluctuate with respect to time.

To illustrate the role of observables and conditions that depend on them in a contract, we will consider the following simple example.

Example 1.1.1. Consider an American call option for purchasing one share of a certain kind of stock on June 30, 2015 for \$5. The expiration date of the option is December 17, 2015 (and so the option may be exercised on any date from June 30, 2015 to December 17, 2015). The strike price[1] of the option is \$80. The transaction of the sale of the stock must be finished within 30 days of payment.

An *American option* is a contract that gives the owner the right, but not the obligation, to buy or sell a specified asset at a specified price on or before a specified date [23]. This example describes the conditions that are required for the sale of one share of stock. It shows the role that observables and conditions commonly play in contracts. If a payment of \$5 on June 30, 2015 is made to the option seller to buy the option (first condition), the option contract will become effective. If the option buyer exercises the option by paying \$80 to the option seller on or before December 17, 2015 (the second condition), the option seller will transfer one share of the stock to the option buyer within 30 days after the option is exercised. The payments of \$5 and \$80 are both observables on which the first and second conditions respectively depend. The transference of the stock is also an observable.

This American call option can be deconstructed into three components:

(1) **Condition 1**: The option buyer gives the option seller \$5 on June 30, 2015 to buy an American call option consisting of a conditional agreement composed of the following two components.

[1]The price at which the asset may be bought or sold in an option contract (also called the exercise price) [33, p. 7]

(2) **Condition 2**: The option buyer chooses to exercise the option by paying $80 to the option seller on or before December 17, 2015.

(3) **Agreement**: The option seller is obligated to transfer one share of stock to the option buyer no later than 30 days after the option is exercised.

The contract thus has the following form:

```
if Condition 1
then
      if Condition 2
      then Agreement
```

A *conditional agreement* is an agreement that comes into effect only when a certain condition is met. Both if-then parts of the contract are conditional agreements. The second conditional agreement consists of **Condition 2** and **Agreement**, and the first conditional agreement consists of **Condition 1** and the second conditional agreement.

The *offer time* of the American call option is the time the option contract is offered by the option seller to possible buyers. At the offer time, the option contract is not a legally binding agreement. It becomes a legally binding agreement only when the option contract is purchased by the option buyer (i.e., the time when **Condition 1** is satisfied).

1.1.3 The Role of Observables and Conditions

A contract is an agreement that is often composed of various sub-agreements, and some of these sub-agreements may be conditional agreements like the American call option. When the values of the observables in the condition make the condition true,

the agreement of the conditional agreement becomes effective. Different agreements may be generated when the observables have different values. Conditional agreements, like either of the two in the American call option, can be viewed as "rules" that are applied when certain conditions become true.

As time goes on, and more and more conditions become true as the result of rules being applied, more and more agreements come into effect. The *state* of a contract at a time t is the set of agreements and rules that are in effect at t. The meaning of the agreements and rules of a contract depend on the values of observables mentioned in them. The state of a contract as a set of agreements and rules can evolve overtime.

A contract like the American Call Option is *dynamic* in the sense that it contains temporally based conditions that trigger changes to the state of the contract when the conditions become true.

A *trace* of a contract is the sequence of states of the contract that are generated over time starting at the offer time of the contract. A "model" of a contract gives each observable in the model at each time a value. A model of observables determines a trace of a contract. The *meaning* of a contract is thus a mapping from models of observables to traces of the contract. Contracts are *equivalent* if they produce the same traces from the same observables. Thus the meaning of a contract and how the state of the contract evolves over time depend on the values of observables mentioned in the contract.

1.1.4 Why do We Need a Precise Description?

Since the structure of the conditions in a contract can be very complex, it can be difficult to write dynamic contracts in a natural language in a clear and unambiguous

way. Ambiguous contract specifications can be a major source of conflict between the parties involved in the contract. For example, the contract parties may think they agree on a contract, but later disagree on what the contract requires. Writing a contract in a formal language can help to make the meaning of the contract understandable. Precision of language and careful specification of procedures can help to reduce the potential confusion and disagreement in the contracting process and execution, and can help to improve communication between parties.

In order to address these inherent complexities in contracts, there is a need to have a consistent, precise, and complete description mechanism for complex dynamic contracts to improve communication between parties. A formal definition of a contract is presented in Section 7.4.

1.1.5 Why do We Need a Formal Representation?

Another important issue is to understand why we need a formal representation of a contract. In [2], K.A. Adams describes the different kinds of uncertain meaning in contract language: ambiguity, failure to be sufficiently specific, mistakes, conflicts, failure to address an issue, and vagueness. If there are uncertain clauses in a contract, and all options to determine the meaning fail, it may be possible to sever and void just those affected clauses. The use of formal methods can greatly increase our understanding of contracts by revealing the above uncertain meanings that might otherwise go undetected.

Contract professionals that deal with complex contracts have to process them for various activities. Those activities include generating legal documents, monitoring their execution for performance, analyzing their ramifications for planning, valuation, scheduling, supply chain management, production planning, tax reporting, decision

support, reasoning about violations of obligations, and other back-end activities. Contracts — in particular, dynamic contracts — are naturally becoming more complex as the complexity of the world and human interaction grows. Continuing to write complex contracts in natural language is not sustainable if we want the contracts to be understandable and analyzable.

A precise formal language for writing contracts can be seen as a tool to help people understand and manage contracts. If a contract is written in a formal language with a precise semantics then it becomes a formal object that has a mathematically precise meaning and that can be manipulated by software. A *formal contract* of this kind can be written, analyzed, and manipulated in various ways with the help of sophisticated software tools.

1.2 Our Approach

This thesis presents FCL, a Formal Contract Language with a precise semantics for expressing general contracts that may depend on temporally based conditions. We think of a contract in two ways. Syntactically, a contract is a written expression satisfying certain syntactic conditions. Semantically, a contract is a set of temporally based promises between parties. Writing a contract in a formal language can help to make the meaning of the contract understandable.

The underlying logic of FCL is a version of simple type theory called STT. Simple type theory is a natural extension of first-order logic which has a very simple syntax. In simple type theory there are types built up from atomic types using type constructors. FCL consists of several kinds of syntactic objects that include the types and expressions from STT. We use a theory of STT as our base theory. On top of the STT theory, we

build FCL.

1.3 Contributions

We summarize our contributions below.

- Derivation of a set of desirable requirements that FCL should support, motivated by carefully selected examples of contracts. The requirements of FCL are presented in Chapter 3.
- Provision of a more precise notion of contract: We produce a clear definition of what a contract is (see Section 1.1.3 for more details).
- Development of a formal language for writing a wide range of contracts: We present FCL, an expressive formal contract language for writing general contracts. FCL has a formal syntax (see Chapter 6) and a precise semantics (see Chapter 7).
- Demonstration of the expressivity of our language: We formalize in Chapter 8 three kinds of contracts, an option contract, a sale of printer contract, and a lease contract in FCL. The formalizations of these three examples explain how contract concepts can be written in FCL.
- Development of methods for analyzing contracts written in the language: We present a reasoning system for FCL in Chapter 9. As we will show, our reasoning system can address what it means to fulfill or breach a contract.

The work presented in this thesis was done in collaboration with my supervisor, William M. Farmer. We developed the underlying ideas of FCL and designed its

syntax and semantics together as well as the reasoning system for FCL. I formulated our definition of a contract, derived the requirements for FCL, worked out the details of FCL, developed the examples that illustrate the use of FCL, and conducted an extensive review of the literature related to this thesis. STT, the underlying logic of FCL, is previous work done by Dr. Farmer.

1.4 Structure of the Thesis

This thesis is based on our research published in [20] and [21]. It provides more details and explanations of our contract language FCL. The remainder of this thesis is organized as follows.

Chapter 2 introduces the required background. This chapter aims to familiarize the reader with the domain of investigation and to clarify the concepts mentioned in the thesis.

Chapter 3 presents a set of requirements for an unambiguous formal language for writing contracts.

Chapter 4 provides a survey of the current state-of-the-art in the literature with respect to the formalisms aiming at defining a language for writing contracts.

Chapter 5 presents the underlying logic of FCL.

Chapter 6 gives a full presentation of the syntax of FCL.

Chapter 7 presents the formal semantics of FCL.

Chapter 8 shows how the examples of contracts from Chapter 3 are expressed in FCL.

Chapter 9 introduces a system for reasoning about contracts written in FCL. This

chapter also shows how this reasoning system can be used to simulate a contract over time.

Chapter 10 highlights and assesses the contributions made by this thesis. This chapter also draws conclusions and suggests avenues for future work.

Chapter 2

Background

This chapter aims to familiarize the reader with the domain of investigation and to clarify the concepts that are used throughout this thesis. Section 2.1 gives definitions relevant to contracts. Section 2.2 introduces the standard deontic logic. Section 2.3 introduces simple type theory.

2.1 Contract Concepts

In this section, we review some selected literature on contracts [7, 11, 48, 58] and give a list of relevant definitions.

Definition 2.1 (Party)**.** *Parties* are the people or entities that have mutually agreed to the contract and are bound by its terms and conditions. In the case of written agreements, the parties are typically identified as the people or entities that signed the agreement.

Definition 2.2 (Term)**.** *Term* refers to the time period during which a contract is in force. For example, a lease with a one-year term lasts for a year.

Definition 2.3 (Obligation)**.** An *obligation* expresses what an individual ought to do.

Definition 2.4 (Permission)**.** A *permission* expresses what an individual may do.

Definition 2.5 (Prohibition)**.** A *prohibition* expresses what an individual ought not do.

Definition 2.6 (Internal Event)**.** An *internal event* refers to actions that can be performed by the parties of the contract.

Definition 2.7 (External Event)**.** An *external event* refers to events that cannot be controlled by the parties of the contract.

Definition 2.8 (Valuation of a contract)**.** A *valuation of a contract* is an estimation of the total economic value of the goods or services exchanged over the term of the contract.

Definition 2.9 (Breach of contract)**.** One of the contract parties' failure to fulfill his or her obligations as described in the contract is known as a *breach of the contract.*

Definition 2.10 (Blame assignment)**.** If a contract is breached, a *blame assignment* is a determination of who among the contract parties will be held responsible for the breach of the contract.

Definition 2.11 (Void contract)**.** A *void contract* is a contract that violates the law—such as a contract to engage in wagering, money laundering, or restraint of trade. The most famous example is a hit man's contract to murder someone. A void contract is not enforceable, and courts will treat it as if it never existed.

2.2 Deontic Aspects of Contracts

Deontic logic is usually regarded as an applied form of modal logic. There are three common modal operators $\mathbb{O}$, $\mathbb{P}$, and $\mathbb{F}$ of deontic logic. The interpretations of the formulas $\mathbb{O}A$, $\mathbb{P}A$, and $\mathbb{F}A$ are, respectively, that A is obligatory, A is permitted, and A is forbidden, where A specifies an *assertion*. The modal operators obey the usual mutual relationships

$$\mathbb{O}A \equiv \neg\mathbb{P}\neg A \quad \mathbb{P}A \equiv \neg\mathbb{O}\neg A \quad \mathbb{F}A \equiv \neg\mathbb{O}A \quad \mathbb{F}A \equiv \neg\mathbb{P}A,$$

are closed under logical equivalence (i.e., if $A \equiv B$ then $\mathbb{O}A \equiv \mathbb{O}B$), and satisfy the axiom $\mathbb{O}A \implies \mathbb{P}A$ (i.e., if A is obligatory, then A is permitted). The latter implies the internal coherency of the obligations in a contract, or, in other words, something forbidden will not be done by executing an obligation.

Some contracts also specify the rights and permissions that the parties have. One may want to make a distinction between "rights" and "permissions", between having the power to do something and the permission to do so, but for certain kinds of analysis these distinctions are not needed.

Unfortunately, deontic logic in general suffers from a number of well-known paradoxes:

- **Ross's paradox** [52] arises from the theorem $\mathbb{O}\varphi \to \mathbb{O}(\varphi \vee \psi)$ which has an instance: "if it is obligatory that one mails the letter then it is obligatory that one mails it or burns it". It is typically viewed as a side effect of the interpretation of 'or' in natural language.

- The **paradox of commitment** [32] is based on the instance $\mathbb{F}\varphi \to \mathbb{O}(\varphi \to \psi)$

which is interpreted as saying that "if it is forbidden to steal a car, then you are obliged to run over a pedestrian if you steal a car".

- The **Good Samaritan's paradox** [38] arises from the rule that if $\vdash (\varphi \wedge \psi) \rightarrow \psi$, then $\vdash \mathbb{O}(\varphi \wedge \psi) \rightarrow \mathbb{O}\psi$. If we substitute φ for "the good Samaritan helps the victim who has been hurt" and ψ for "the victim has been hurt", and given that the good Samaritan ought to help the victim who has been hurt, then it follows that the victim ought to be hurt.

- **Free choice permission paradox** [43] is based on the instance $\mathbb{P}(\varphi \vee \psi) \rightarrow \mathbb{P}\varphi \wedge \mathbb{P}\psi$ which is interpreted as saying that "if it is permitted that one mails a letter or burns it, then one is allowed to both mail a letter and burn it".

- **Gentle murderer's paradox** [43] arises when a contradiction is derived. Gentle murderer's paradox consists of the following sentences:

  (1) John is obligated not to kill his mother ($\mathbb{O}\neg\varphi$).

  (2) If John kills his mother, then John is obligated to kill her gently ($\varphi \rightarrow \mathbb{O}\psi$).

  (3) John kills his mother (φ).

  (4) If Jone kills his mother gently, John kills his mother ($\psi \rightarrow \varphi$).

  (5) Therefore, John is obligated to kill his mother ($\mathbb{O}\varphi$, contradicting (1)).

- **Chisholm's paradox** [12] is arises from the theorem $(\mathbb{O}\varphi \wedge \mathbb{O}(\varphi \rightarrow \psi) \wedge (\neg\varphi \rightarrow \mathbb{O}\neg\psi) \wedge \neg\varphi) \rightarrow \bot$. The impact of this theorem is that certain intuitively consistent formula are in fact inconsistent in standard deontic logic. Chisholm's paradox consists of the following four sentences:

  (1) It ought to be that a certain man goes to the assistance of his neighbors ($\mathbb{O}\varphi$).

(2) It ought to be that if he does go, he tell them he is coming ($\mathbb{O}(\varphi \to \psi)$).

(3) If he does not go then he ought not to tell them he is coming ($\neg\varphi \to \mathbb{O}\neg\psi$).

(4) He does not go ($\neg\varphi$).

These paradoxes are not actually logical paradoxes in the normal sense. They are mismatches between intuition and what the logic actually captures. Thus, a contract that is represented in standard deontic logic may contain a mismatch between intention of the contract drafter and the actual logical consequences. Unlike many approaches that are based on the deontic logic to design a contract formalism [31, 45, 49, 50, 51], we design our language based on the simple type theory, thus avoiding these kinds of paradoxes (see Section 8.1.4 for details).

FCL can express agreements inspired by the obligations, prohibitions, and permissions of deontic logic. We interpret an agreement in a contract inspired by the terms of deontic *obligation* and *prohibition*.

2.3 Simple Type Theory

FCL can be built up based on any chosen high-order logic. We will assume in this thesis that the underlying logic of FCL is a version of simple type theory [19].

Simple type theory is a natural extension of first-order logic which has several advantages over it [19]. In particular, (1) types denote nonempty sets of values; they are used to organize entities. All values have types and operations are restricted to entities of a certain type. Thus the types provide a means to distinguish entities by their values, for example, between numbers, set of numbers, functions from numbers

to sets of numbers, sets of such functions, etc. (2) Functions can be applied to higher-order values such as sets, relations, and functions. (3) Functions can be used to represent other kinds of values such as sets and relations. For example, a set can be represented by a function which maps elements to truth values, and so if an element is in the set, then the function that represents the set maps that element to true. (4) Quantification can be applied not only to individual variables as in first-order logic, but also to higher-order variables. (5) Functions can be defined by lambda abstraction, for example, $\lambda x \,.\, x^2$ defines the function $x \mapsto x^2$. (6) Definite description builds an expression of the form $\mathrm{I}x : \alpha \,.\, P$ that denotes the unique value x of type α that satisfies the property P.

Simple type theory has a very simple and highly uniform syntax, and it is based on the same semantic principles as first-order logic. It is very well understood and easy to work with. Simple type theory is more expressive than first-order logic in a practical sense. Because it includes strong support for specifying a hierarchy of higher-order functions and is equipped with full higher-order quantification and definite description, mathematical ideas can be expressed much more directly in it than in first-order logic [19].

Chapter 3

Guiding Ideas

This chapter aims to present a set of desirable requirements for an unambiguous and rigorous formal language of contracts. We first presents the desirable properties of a formal language for contracts in Section 3.1. Section 3.2 introduces three reference examples in the thesis and further discusses the requirements of FCL based on these examples.

3.1 Requirements of a Language for Contracts

A contract records, orally or in writing, a legally binding agreement between two or more parties [48]. It must include the time when the contract was entered into, the contracting parties, the object of the contract (eg. the item to be purchased or the amount to be borrowed), the material value (eg. the price of the item), and sanctions in the event of a breach of contract and applicable laws. A precise formal language intended to represent contracts can be seen as a tool to help people to understand and manage contracts.

A business normally wants to sign a good contract. From the view of our work, a good contract formalism should be both an "internally good" and an "externally good" contract. The meaning of "internally" and "externally" good contracts can be explored by the perspectives and features given in Table 3.1. Here the "external' means an external monitoring tool for analyzing the performance of contracts.

Table 3.1: Contract Perspectives and Features

	Perspective	Features	Descriptions
Contract Internal	**Expressiveness**	1. Capability to capture contract content (the time when the contract was entered into, the item to be purchased, the amount to be borrowed, etc.). 2. Capability to represent three types of contract commitment (obligations, permissions, and prohibitions). 3. Capability to express two or more parties.	1. Formal semantics. 2. Be capable of representing contract commitments such as obligations, permissions, and prohibitions.
Contract External	**Monitoring**	1. Capability to express and monitor actions of a contract. 2. Capability to detect a violation when executing a contract. 3. Capability to identify which party is responsible for the breach of a contract.	The contract language must be capable of verifying who violated a contract agreement. For all actions of a contract execution, it must be capable of monitoring and detecting faults.
	Enforcement	1. Capability to express laws and regulation. 2. Capability to express reparational agreements.	If a violation of a contract is detected, what enforcement will be applied to responsible parties.

As described in Section 1.1.1, Brian Blum in [11, p. 2] gives a list of properties that a contract must have. A compact summary of the issues involved in contract management is provided in [34, 35]. The author of [36] includes a list of features that in

his opinion a contract language should provide. We present here the requirements that our language should meet. These requirements are derived from the definition and properties of contracts provided by Brian Blum, our analysis of the contract examples introduced in 1.1.2 and 3.2, the perspectives and features of contracts covered in Table 3.1, our investigations of the related work presented in Chapter 4, and the requirements given in [36].

We think our language should have a formal syntax and a precise semantics (R1) and should be able to do the following things.

R2 Distinguish between absolute and relative times.

R3 Specify contract commitments (obligations, permissions, or prohibitions).

R4 Express agreements that depend on the actions by the parties to the contract.

R5 Express reparational agreements.

R6 Enable the dynamic calculation of values.

R7 Specify contracts involving multiple parties.

R8 Express agreements that depend on events that cannot be controlled by the parties of the contract.

R9 Express both instantaneous and continuous actions.

R10 Describe the laws and regulations that will be used in case of disputes.

In the next section, we provide three contract examples to illustrate the requirements that a formal language for writing contracts should satisfy.

3.2 Contract Examples

The primary purpose of having a formal language for writing contracts is to help people have a good understanding of a contract. In order to guarantee that the parties of a contract share a common understanding of the contract, we require that the contract uses terms defined in a shared grammar and has a precise semantics (R1).

We consider first Example 1.1.1, an American Call Option contract. The contract is a bilateral agreement between the contract buyer and seller. This example contract describes how and when the sale of one share of stock, in return for $80, is to be carried out. An agreement is an understanding between parties. It creates obligations to do or not do the specific things that are the subject of that agreement. The modeling of contract commitments (R3) is a self-nominated requirement, given our previous, informal definition of contracts. Agreements can be conditional (R4) on what actions are taken during the execution of a contract, as illustrated in clause 2, where the permission for Party A to buy one share of IBM stock at $80 from Party B is contingent upon Party A exercising the option before the expiration date. A proper formalization of time and temporal information is required. We are not only interested in the actual time points at which an action happens but also in deadlines. Clause 1 is an example of an absolute deadline and clause 3, on the other hand, exemplifies a relative deadline, where the deadline of transfer depends on the time of payment. Thus, our contract language needs to capture the dynamic obligations, permissions, and prohibitions of contracts, provide explicit parties and thus it provides the possibility of specifying multi-party contracts, and provide explicit time to capture the notions of absolute and relative times (R2).

Consider next an extended example of a sale of a laser printer contract in Example 3.2.1 from [35, p. 5].

Example 3.2.1. The contract consists of five clauses:

(1) Seller agrees to transfer and deliver to Buyer one laser printer within 22 days after an order is made.

(2) Buyer agrees to accept the goods and to pay a total of $200 for them according to the terms further set out below.

(3) Buyer agrees to pay for the goods half upon receipt, with the remainder due within 30 days of delivery.

(4) If Buyer fails to pay the second half within 30 days, an additional fine of 10% has to be paid within 14 days.

(5) Upon receipt, Buyer has 14 days to return the goods to Seller in the original, unopened packaging. Within 7 days thereafter, Seller has to repay the total amount to Buyer.

Although this example contract is very simple, two points should be noticed. First, a contract usually specifies actions to be taken in case of the violation of a part of the contract. In the deontic literature [42], this is known as *contrary-to-duty obligations (CTDs)* or *reparational obligations*. Clause 4 of this contract is an example in which a potential violation can generate obligations to "repair" the violation (R5). Second, consider the total amount specified in clause 5. Taken literally, it would imply that the total amount the seller must repay to the buyer in case of the printer is returned should be $200 as stated in clause 2. Actually, this is not the intention of this contract. In fact, the total amount to repay is the amount that the buyer has

already paid the seller. Also notice that the deadline for returning the total amount to the buyer depends on the time the buyer returns the printer to the buyer. The time at which the buyer makes a return depends on the time that the delivery of a printer is made. Thus, a language for writing contracts should be able to define and use procedures that enable the dynamic calculation of values (R6) such as those deadlines based on relative time, or amounts of money for payment.

The buyer and seller in this example are parties of this contract. The carrier that delivers the goods is an example of a third party (R7) who will be called on to join the contract when the Buyer places an order to purchase a laser printer from the Seller. Parties are the people or entities that have mutually agreed to the contract and are bound by its terms and conditions. In the case of written agreements, the parties are typically identified as the people (may on behalf of a company or entity) that signed the agreement. A *third-party* is bound in a contract by a relation with the parties in contract.

Now consider next an example lease contract [35, p. 6]:

Example 3.2.2. The contract consists of five clauses:

(1) The term of this lease is for 3 months, beginning on January 1, 2011.

(2) At the expiration of said term, the lease will automatically be renewed for a period of one month unless either party (Landlord or Tenant) notifies the other of its intention to terminate the lease at least one month before its expiration date.

(3) The landlord has the obligation to provide one apartment to the tenant throughout the term.

(4) The tenant has the obligation to pay the amount of $1000 per month to the landlord, with payment due on the 7th day of each month.

(5) The rent is adjusted semi-annually according to the Consumer Price Index (CPI).[1]

Clauses 2 and 4 are examples of instantaneous and continuous actions (R9), respectively. Clause 5 illustrates how contracts may depend on external, time-varying observables (R8).

Thus, our contract language needs to capture the dynamic obligations, permissions, and prohibitions of contracts, provide explicit parties as well as the possibility of specifying multi-party contracts, and provide explicit time to capture the notions of absolute and relative times.

The list of requirements of a language for contracts not only allows us to compare existing formal approaches to contracts, but it also provides a guideline for constructing new formalisms.

[1]The CPI in the United States is defined by the Bureau of Labor Statistics as "a measure of the average change over time in the prices paid by urban consumers for a market basket of consumer goods and services." For example, if the tenant is paying $1000 a month and the CPI jumps 5% in one year, the monthly rent will jump to $1050.

Chapter 4

Related Work

Currently, several different approaches aiming at defining a formal language for writing contracts have been proposed. According to T. Hvitved [35] there are three main categories of contract formalisms: logic based formalisms (e.g., classical [17], modal, deontic [31, 45, 49, 50, 51], and defeasible logic [27, 53]), event-condition-action (ECA) based formalisms [26, 41], and action/trace based formalisms [3, 39]. Also, other contract formalisms worth mentioning use functional programming ([46, 47]) or models of computation (e.g., FSMs [39], Petri Nets [14, 40]).

Most formalisms are applicable specifically to contracts in general. Some researchers [8, 46, 47] have given extensive attention to automating the calculations of the monetary values of the financial contracts. Other researchers [5, 6] have sought to define a rule interchange language for the legal domain. There are also efforts to build common taxonomies [1, 9, 10, 55] of financial terms and events that can be used to specify events and actions in the contract automation process.

This chapter surveys the literature and discusses the existing research relevant to our work conducted by various academic and industry research groups in the area

of formalized contracts. Specifically, Section 4.1 examines the current research that aims at defining a formal language for representing contracts. In particular, it looks at existing classifications of different approaches and discusses the advantages and disadvantages of different approaches. The remaining sections go through the approaches from the different categories of contract formalisms.

4.1 Contract Formalisms

Our language FCL is most closely related to the languages used by the ten approaches below. Table 4.1 shows the main advantages and disadvantages of the kinds of contract formalisms that these approaches are based on.

- R. Lee (Lee) [40].
- A. Goodchild, C. Herring, and Z. Milosevic (GHM) [26].
- S. L. Peyton Jones and J. M. Eber (PJ&E) [46, 47].
- G. Governatori and Z. Milosevic (G&M) [28, 31, 41].
- J. Andersen, E. Elsborg, F. Henglein, J. G. Simonsen, and C. Stefansen (AEHSS) [3].
- A. Daskalopulu and M. Sergot (D&S) [14, 16, 24, 25].
- C. Prisacariu and G. Schneider (P&S) [22, 49, 50, 51].
- Wyner (Wyner) [58].
- P. Bahr, J. Berthold, M. Elsman (BBE) [8].

Table 4.1: Main Advantages and Disadvantages of Contract Formalisms

Contract Language	**Approach**	**Advantage**	**Disadvantage**
Models of computation based languages	1 Lee 2. D&S	1. Widely used in business world and industry. 2. Intuitive for technical and business people to write and understand	Not a strictly rigorous approach.
Functional programming based language	PJ&E	Rigorous formal approach.	Not easily understood by technical and business people.
ECA-based languages	1. GHM 2. G&M	1. Widely used in business world and industry. 2. Intuitive for technical and business people to write and understand.	Not a strictly rigorous approach.
Action/trace based languages	1. AEHSS 2. BBE	Rigorous formal approach.	Not easily understood by technical and business people.
(Deontic) Logic based languages	1. P&S 2. Wyner 3. OASIS	Rigorous formal approach	1. Not easily understood by technical and business people. 2. Not widely used in business world and industry.

- OASIS LegalRuleML Technical Committee (OASIS) [5, 6].

Table 4.2 shows a comparison of FCL and those approaches based on the contract design requirements introduced in Chapter 3. We notice that none of those approaches has reached enough maturity to be considered a solution to the problem of the formal definition of a contract. Only a few models come with a precise, formal semantics, but these are too simple. For example, since PJ&E [46, 47] focuses almost entirely on finding the value of a contract; they consider the semantic meaning of a contract to

Table 4.2: Requirements comparison matrix

	Lee	D&S	PJ&E	G&M	GHM	AEHSS	BBE	P & S	Wyner	OASIS	FCL
R1				√		√	√	√	√	√	√
R2	√	√	√	√		√	√			√	√
R3	√	√	√	√	√	√	√	√	√	√	√
R4	√	√	√	√	√	√	√	√	√	√	√
R5	√	√		√		√	√	√	√	√	√
R6			√		√	√	√				√
R7	√			√		√	√		√	√	√
R8		√	√	√		√	√		√	√	√
R9	√									√	√
R10										√	

be its value. Most approaches [26, 40, 41, 46] lack a formal semantics and a reasoning system even though they provide a good framework for monitoring contracts. We find this surprising since one of the purposes of defining a contract language is to have a formal, unambiguous semantics. More detailed comparisons of the related works will be given later in Section 10.2.

In the following sections we will examine the approaches with respect to the categories in the first column of Table 4.1. Each paper in this Chapter includes its own related work section which lists works that are more closely related to the specific approach. We present here some more generally related work aiming at designing a formal language for writing contracts. Notice that, the fonts of each section follow the fonts that are used in the paper.

4.2 Models of Computation Based Approach

Many contracts can be well modeled as processes. A contract can change from one state to another in response to some events that are relevant to the contract. A contract starts off in an initial state which describes each party's obligations, permissions, or prohibitions, and then transitions occur to move the contract from one state to the next until finally the contract ends up in a terminal state, if one exists.

4.2.1 Lee's Approach

R. Lee employs Petri Nets to support the automated design of trade procedures in electronic contracting [40]. Contracts are viewed as Petri net transition systems, where a set of states can be active at any point in time. The events that happen and actions performed by contract parties are specified as attributes of transitions that

can trigger new states to become active and to deactivate some. The idea at the core of Lee's approach is to model the transition system with logic programming and to model contract concepts such as obligations in terms of transitions. Lee introduces the predicate 'trans':

$$\mathsf{trans}([A_1, \ldots, A_m], [B_1, \ldots, B_n], E).$$

This indicates a transition from a state satisfying the precondition $[A_1, \ldots, A_m]$ of the transition, to a state satisfying the postcondition $[B_1, \ldots, B_n]$ where $[A_1, \ldots, A_m]$ is a predicate on the current active states and $[B_1, \ldots, B_n]$ is a predicate on the new active state. The parameter E is an action taken by one of the contracting parties that causes the transition.

Lee's approach on applying deontic logic to the case of contracting is based on a suggestion by A.R. Anderson and O.K. Moore [4] relating deontic logic to *alethic modal logic* using the definition

$$O\varphi \leftrightarrow \Box(\neg\varphi \rightarrow S)$$

where S is a propositional constant indicating the occurring from the violation of one's obligations. The representation of obligation (and other deontic concepts) in the contracting model is based on an interpretation of contingent actions leading to sanctions.

As an example of how contract aspects are modelled, consider the obligation for party X to perform action A:

$$\mathsf{trans}([s(0)], [s(1)], X : A).$$

$$\mathsf{trans}([s(0)], [\mathsf{default}(X)], \neg X : A).$$

Here the notation $default(X)$ indicating the state where party X has defaulted on the contract. The action is obligatory, so therefore it results in the state of "default" if it is not done. Hence, if X does action A, a transition is made from state $s(0)$ to state $s(1)$. If X does not perform A, the contract goes to a default state, which is a state indicates a breach of the contract.

We now show how Example 1.1.1 is encoded in Lee's language.

Example 4.2.1. Let $S(1), S(2), S(3)$, and $S(4)$ be states of the contract. The contract encoding is presented below:

```
C(buyer, seller, [S(1),S(2),S(3),S(4)]) :: =
trans([S(1)],[S(2)],buyer:rd(30−jun−2015):pay(seller,$5),
trans([S(2)],[S(3)],buyer:rb(17−dec−2015):pay(seller,$80),
trans([S(3)],[S(4)],seller:rb(30days):deliver(buyer,stock),
trans([S(3)],[default(seller)],∼seller:rb(30
   days):deliver(buyer,stock)
```

Lee's approach succeeds in capturing most of the aspects of a contract. It includes the aspects of absolute and relative times (R2), contract commitments (R3), action-dependent agreements (R4), reparational agreements (R5), multiple parties (R7), and instantaneous and continuous actions (R9). The problem of Lee's contracts and modeling of deontic modalities lacks a clear language definition as well as a clear semantics of contracts (R1). There is no account to external events (R8) nor description of laws and regulations (R10). Another disadvantage of the Petri nets formalization provided by Lee is that it does not handle well the dynamic calculation of values such as monetary values (R6).

4.2.2 D&S's Approach

A. Daskalopulu and M. Sergot (D&S) [16, 14] proposed a contract language based on the Event Calculus [57]. In Daskalopulu's PhD thesis [13], she mainly views contracts from a legal perspective and translates them to a Petri-Net representation for reasoning about violations. Her most current research [15, 24, 25] differs from her thesis but focuses on evidence-based monitoring. D&S also propose a mapping from a contract representation in event calculus to default logic [56]. The resulting representation allows for defeasible reasoning [44] with contracts.

We now show how Example 1.1.1 is encoded in D&S's language.

Example 4.2.2. We introduce the following notational conveniences:

b = buyer

s = seller

a = buys contract for $5

p = pays $80

g = gives stock to buyer

The contract encoding is presented below:

initially $\{\mathbb{O}_b(a), \langle b : a\rangle \mathtt{T}, \langle not\ \ b : a\rangle \mathtt{T}\}$

1 [ ]$(\mathbb{O}_b(a) \rightarrow [b : a]\mathbb{O}_b p)$

2 [ ]$(\mathbb{O}_b(a) \rightarrow [not\ \ b : a]\neg\mathbb{O}_b a)$

3 [ ]$(\mathbb{O}_b(a) \rightarrow [not\ \ b : a]$ terminated)

4 [ ]$(\mathbb{O}_b(p) \rightarrow [b : p]\mathbb{O}_s g)$

5 [ ]$(\mathbb{O}_b(p) \rightarrow [not\ \ b : p]$ terminated)

6 [ ]$(\mathbb{O}_b(p) \rightarrow [not\ \ b : p]\neg\mathbb{O}_s g)$

7 [ ]($\mathbb{O}_s(g) \rightarrow [s : g]$ terminated)

8 [ ]($\mathbb{O}_s(g) \rightarrow [not\ \ s : g]$ terminated)

In Example 4.2.2, the keyword `initially` means that the following formula is true at the start of contract execution. The [ ] symbol is the box from modal logic and means "necessarily" (i.e. in every state) when there is nothing inside the box. $[b : a]$ is from dynamic logic and means "always after b has brought about a". The $< b : a >$ means that it is possible that b can bring about a.

The approach provide by D&S includes the aspects of absolute and relative times (R2), obligations and prohibitions (R3), action-dependent agreements (R4), reparational agreements (R5), and external events (R8). There is no account to dynamic calculation of values (R6), multiple parties (R7), instantaneous and continuous actions (R9), nor description of laws and regulations (R10). D&S does not give a formal account of actions that are not performed. In this approach, action that are not performed are treated as "negative actions". A negative action is informally described with the *not* prefix. A precise formal account of these negative actions as well as a clear semantics (R1) of contracts is needed.

4.3 Functional Programming Based Approach

S.L. Peyton Jones and J-M. Eber (PJ&E) present in the paper "Composing contracts: An adventure in financial engineering" [46] a functional combinator language for modeling the complex financial contracts traded in derivative markets. The design of PJ&E's language can be seen as a declarative, domain-specific language embedded in Haskell. Let us consider an example [47] of a financial contract below.

C: The permission to choose on 30 June 2000 between:

D_1 Both of

D_{11} Receive \$100 on 29 Jan 2001.

D_{12} Pay \$105 on 1 Feb 2002.

D_2 An option exercisable on 15 Dec 2000 to choose one of:

D_{21} Both of

D_{211} Receive \$100 on 29 Jan 2001.

D_{212} Pay \$106 on 1 Feb 2002.

D_{22} Both of

D_{221} Receive \$100 on 29 Jan 2001.

D_{222} Pay \$112 on 1 Feb 2003.

Each D_i represents a contract. That is, C is a contract formed by combining together simpler contracts, such as D_1, which in turn is formed from simpler contracts D_{11} and D_{12}.

The compositional structure of financial contracts is appropriate for formalization in functional programming, which is compositional in nature. PJ&E define the set of following contract constructors:

```
c ::= zero                                   (no rights/obligations)
    | one(k)                  (permission to transfer one unit of currency k)
    | give(c)                 (reverse the permissions and obligations of c)
    | and(c1,c2)                  (immediately acquire both c1 and c2)
    | or(c1,c2)          (immediately acquire either c1 or c2, but not both)
```

| `cond(o,`c_1`,`c_2`)` (immediately acquire c_1 if o holds, otherwise c_2)

| `scale(`o, c`)` (immediately acquire c where all amounts are scaled by o)

| `when(`o, c`)` (immediately acquire c as soon as o holds)

| `anytime(`o, c`)` (acquire c once, anytime o holds)

| `until(`o, c`)` (immediately acquire c, but abandon c once o holds)

where k ranges over currencies and o ranges over time-varying values.

PJ&E demonstrate how the compact library suffices for describing standard financial contracts such as zero-coupon bonds, options, swaps, and futures. The language presented by PJ&E also provides a decomposition of these standard financial contracts into individual payment commitments that are combined declaratively using a small set of contract combinators.

We now show how Example 1.1.1 is encoded in PJ&E's language.

Example 4.3.1. Let t be a time, n be an amount, and USD denote the USA currency. We introduce the following notational conveniences:

```
receive_on(d,n) = when(d,scale(n,one(USD))
pay_on(d,n) = give(receive_on(d,n))
```

With these definitions the contract can be expressed as follows:

```
c = when(2015−6−30,
        and(pay_on(2015−6−30,5),
            when(2015−12−17,
                or(and(pay_on(2015−12−17,80),
                       receive_on(add(2015−12−17,30),IBM)),
                   zero)
```

The combinator library of PJ&E supports absolute and relative times (R2), obligations and prohibitions (R3), action-dependent agreements (R4), dynamic calculation of values (R6), and external events (R8). There is no account to reparational agreements (R5), accounting for multiple parties (R7), instantaneous and continuous actions (R9), nor the description of law and regulations (R10).

The strength of this approach is its ability to perform compositional analysis of contracts expressed in the combinator library. For instance, they show how to give an abstract valuation semantics to combinators to process a contract. But compared to all other work, theirs is a special purpose language for expressing only the financial contracts.

4.4 Event-Condition-Action Based Approach

4.4.1 G&M's approach

G. Governatori's approach is based on the business contract language (BCL) proposed in [28, 29, 31, 41] to represent contracts. BCL is a domain specific language, designed with the purpose of enabling event-based monitoring of business activities. It was developed by taking into account several policy and community frameworks and an expressive event language for the specification of event-based behavior as part of policy expressions. The first presentations of BCL [41] used English to express contracts and thus lacked a formal language.

The motivation for BCL is to enable monitoring of contract execution in an event-based manner. A single event can be used to signify any one of the following:

- An action performed by one of the signatories to the contract, or any other

party mentioned in the contract.

- A temporal occurrence such as the passing of a deadline.
- A change in the contract state associated with a contract variable.
- A contract violation and other conditions associated with contract execution.

The grammar for BCL is as follows (keywords are in bold face, [·] denotes optionality):

`Contract` ::= `Policy`*

`Policy` ::= **Policy**: `Name`
Role: `Role`
Modality: `Modality`
Trigger: `EP`$^+$
`[Guard]`
Behavior: `EP`

`Modality` ::= **Obligation** | **Permission** | **Prohibition**

`EP` ::= **not** `E` | `E` **and** `E` | `E` **or** `E` | `E` **before** `E` | `ET` **before** `T`

`Guard` ::= **Guard**: `StateExp` | **violated**`(Name)`

An event pattern is a means for describing a state of affairs that introduces specific relationships between events of relevance to business contracts. Examples of event patterns are logical relationships between events, temporal relationships between events, and temporal constraints on event patterns. Whereas events are atomic, event patterns are used to describe complex events.

In terms of a semantic model of contracts, Governatori and Milosevic [28] introduce the formal contract logic (FCL). FCL is restricted to ought-to-do statements. The grammar for FCL is as follows:

$$
\begin{aligned}
l \quad & ::= p \mid \neg p && \text{(literals)} \\
ml \quad & ::= O_s l \mid \neg O_s l \mid P_s l \mid \neg P_s l && \text{(modal literals)} \\
\otimes - exp \quad & ::= ml \mid O_{s_1} l_1 \otimes \cdots \otimes O_{s_n} l_n && \text{(}\otimes\text{-expressions)} \\
& \quad \mid O_{s_1} l_1 \otimes \cdots \otimes O_{s_n} l_n \otimes P_{s_{n+1}} l_{n+1} \\
\varphi \quad & ::= l \to \varphi \mid ml \to \varphi \mid \otimes - exp && \text{(policies)}
\end{aligned}
$$

where literals l are propositional atoms and events, and the connective $\otimes$ is used to represent contrary-to-duty structures.

We now show how Example 1.1.1 is encoded in G&M's language.

Example 4.4.1. Let P_i be the policies, where i refers to the clauses in the example contract. The following events are used:

`init: Initial the state of the contract`

`buy_option: Buy an option`

`buy_stock: Buy a stock`

The contract encoding is presented below:

Policy: P1

Role: Buyer

Modality: **Permission**

Trigger: init

Behavior: buy_option **before** 2015−06−30

Policy: P2

Role: Buyer

Modality: **Permission**

Trigger: buy_option

Behavior: buy_stock **before** 2015−12−17

Policy: P3

Role: Seller

Modality: **Obligation**

Trigger: buy_stock

Behavior: deliver **before** 30

G&M sketches how to map a subset of BCL to FCL and provides a Gentzen systems for reasoning with deontic concepts. Besides a formal semantics (R1), this approach also includes the aspects of absolute and relative times (R2), contract commitments (R3), action-dependent agreements (R4), reparational agreements (R5), multiple parties (R7), and external events (R8). There is no account to dynamic calculation of values (R6), instantaneous and continuous actions (R9), and description of laws and regulations (R10).

4.4.2 GHM's Approach

A. Goodchild, C. Herring, and Z. Milosevic (GHM) [26] model contracts as sets of policies. A policy specifies that a legal entity is either forbidden or obliged to perform an action under certain event-based conditions. The grammar for policies is as follows:

Policy ::= VariableDeclaration*
When Condition
Action
must [**not**] **occur where** Condition
otherwise Trigger

```
Action ::= action(ActionName, Actor, Audience, Time)
Trigger::= trigger(ActionName, Audience)
```

where $[\cdot]$ denotes optionality and $*$ denotes zero or more occurrences.

We now show how Example 1.1.1 is encoded in GHM's language.

Example 4.4.2. Let S and B be the seller and buyer of the option defined below:

```
S = Contract.Seller;
B = Contract.Buyer;
```

The contract is encoded as follows:

```
when Contract.State == 'initial'
  action(buy option,B,S,t)
must occur where
  t ≤ 2015−6−30
otherwise
  trigger(send_notice_of_breach,∗,''Buyer failed to buy contract'')

when {delivery made and choose to exercise the option}
  action(exercise option,B,S,t)
must occur where
  t ≤ 2015−12−17
otherwise
  trigger(send_notice_of_breach,∗,''Buyer failed to exercise the
     option'')

when buy share made
  action(transfer stock,S,B,t)
```

```
must occur where
  {t is within 30 days of payment}
otherwise
  trigger(send_notice_of_breach,*,‘‘Seller failed to transfer stock’’)
```

This approach includes the aspects of contract commitments (R3), action-dependent agreements (R4), dynamic calculation of values (R6), and explains how ECA can be used to model contracts. Unfortunately, the policy language of GHM does not have a semantics (R1). It also lacks a detailed description of the syntax; there is only the incomplete grammar above. There is no account to absolute and relative times (R2), reparational agreements (R5), dynamic calculation of values (R6), multiple parties (R7), external events (R8), instantaneous and continuous actions (R9), nor description of laws and regulations (R10).

4.5 Action/Trace Based Approach

4.5.1 AEHSS'S Approach

J. Andersen, E. Elsborg, F. Henglein, J.G. Simonsen and C. Stefansen (AEHSS) [3] consider commercial contracts that govern the exchange of resources between multiple parties. The approach models transaction patterns of companies (agents) as transfers of resources, referred to as events. This approach is inspired partly by the compositional approach of Peyton Jones and Eber.

The grammar of AEHSS's contract calculus is as follows:

k ::= **letrec**$\{f_i[X_i] = c_i\}_{i=1}^{m}$ **in** c

c ::= **Success**

| **Failure**

| $c_1 + c_2$

| $c_1 || c_2$

| $c_1 ; c_2$

| $f(\mathbf{a})$

| **transmit**$(A_1, A_2, R, T|P)$.c

where X_i is a vector of formal parameters for use in the embedded contract c_i, A denotes an agent, R denotes a resource, T denotes time, and P is a predicate.

AEHSS provides a trace-based semantic model for contracts. A trace is a finite sequence of events, and events have the form

$$\mathsf{transmit}(A_1, A_2, R, T)$$

denoting the transmission of resource R from agent A_1 to agent A_2 at time T. A denotational semantics maps contract specifications compositionally into sets of traces.

We now show how Example 1.1.1 is encoded in AEHSS's language.

Example 4.5.1. Let t be a time. The contract encoding is presented below:

```
letrec
  sale(b,s,payment,goods,t,days) =
    (transmit(b,s,payment,T|T ≤ t).
    (transmit(s,b,goods,T'|T' ≤ T+days)).

  option[b,s,fee,payment,goods,days,t1,t2] =
      (transmit(b,s,fee,T|T ≤ t1).
      Success
```

```
  +
  sale (b,s,payment,goods,t2,days)).
in
option(``buyer'',``seller'',5,80,IBM,30,2015−06−30,2015−12−17)
```

Besides a formal semantics (R1), absolute and relative times (R2), contract commitments (R3), action-dependent agreements (R4), reparational agreements (R5), dynamic calculation of values (R6), the contract language has support for multiple parties (R7), and external events (R8). There is no account to instantaneous and continuous actions (R9), nor description of laws and regulations (R10). T. Hvitved [35] points out that in AEHSS's calculus it is impossible to assign blame.

4.5.2 BBE's Approach

P. Bahr, J. Berthold, and M. Elsman (BBE) present in [8] a multi-party contract language for modeling financial contracts that is capable of expressing common FX and other derivatives. The language includes a simple denotational cash-flow semantics independent of any stochastic aspects. Besides, a reduction semantics for the language that evolves contracts over time in accordance with the denotational semantics is provided.

The syntax of BBE's language is as follows:

$$
\begin{array}{rrl}
\text{types} & \tau ::= & \texttt{Real} \mid \texttt{Bool} \\
\text{expressions} & e ::= & x \mid r \mid b \mid \texttt{obs}(l,t) \mid op(e_1,\cdot,e_n) \mid \texttt{acc}(\lambda\, x\, .\, e_1, d, e_2) \\
\text{contracts} & c ::= & \emptyset \mid \texttt{let}\ x = e\ \texttt{in}\ c \mid dc \mid c_1 \& c_2 \mid e \times c \mid a(p \to q) \mid \\
 & & \texttt{if}\ e\ \texttt{within}\ d\ \texttt{then}\ c_1\ \texttt{else}\ c_2
\end{array}
$$

where $x \in \texttt{Var}, r \in \mathbb{R}, b \in \mathbb{B}, l \in \texttt{Label}, t \in \mathbb{Z}, d \in \mathbb{N}, a \in \texttt{Asset}, p, q \in \texttt{Party}, op \in \texttt{Op}$

BBE formalizes their contract language in the Coq proof assistant [54]. Using the code extraction functionality of the Coq system, they automatically extract an implementation of an embedded contract domain-specific language in Haskell. They also provide machine-checkable proofs (in Coq) of the key properties of the contract language.

We now show how the Example 1.1.1 is encoded in BBE's language.

Example 4.5.2.

```
if obs(buyer defaults,0) within 0
then 5 × USD(buyer → seller) &
     if obs(buyer exercise option,0) within 170
     then 80 × (USD(buyer → seller)) &
          30 ↑ 1 × (Stock(seller → buyer))
     else ∅
else ∅
```

Besides a formal semantics (R1), BBE's cash-flow trace based approach [8] has support for absolute and relative times (R2), contract commitments (R3), action-dependent agreements (R4), reparational agreements (R5), dynamic calculation of values (R6), multiple parties (R7), and external events (R8). There is no account to instantaneous and continuous actions (R9), nor description of laws and regulations (R10).

4.6 Deontic Logic Based Approach

4.6.1 P&S's Approach

C. Prisacariu and G. Schneider (P&S) [49, 50, 51] introduce a contract language $\mathcal{CL}$ for expressing electronic contracts based on a combination of deontic, dynamic, and temporal logics. The concepts that $\mathcal{CL}$ captures are drawn from legal contracts. As in the logic of Governatori and Milosevic [30], $\mathcal{CL}$ restricts deontic modalities to ought-to-do statements, that is deontic modalities can only be applied to actions that are performed. On top, they add the modalities of dynamic logic to be able to reason about what happens after an action is performed. $\mathcal{CL}$ also incorporates the notions of contrary-to-duty and contrary-to-prohibition by explicitly attaching to the deontic modalities a reparation that is to be enforced in case of violations.

The grammar of $\mathcal{CL}$ is as follows:

$$\mathit{Contract} \ ::= \ D; C$$

$$C \ ::= \ \varphi \mid O_C(\alpha) \mid P(\alpha) \mid F_C(\alpha) \mid C \to C \mid [\delta]C \mid \bot \qquad \text{(CL expressions)}$$

$$\alpha \ ::= \ a \mid 0 \mid 1 \mid \alpha \times \alpha \mid \alpha \cdot \alpha \mid \alpha + \alpha \qquad \text{(deontic actions)}$$

$$\delta \ ::= \ a \mid 0 \mid 1 \mid \delta \times \delta \mid \delta \cdot \delta \mid \delta + \delta \mid \delta^* \mid \varphi? \qquad \text{(dynamic actions)}$$

$$\varphi \ ::= \ \varphi \mid 0 \mid 1 \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \qquad \text{(tests)}$$

Actions are propositional atoms in $\mathcal{CL}$. Deontic actions α are the actions used inside the deontic modalities. The dynamic condition $[\delta]C$ means that if δ happens, then C must be fulfilled. Dynamic actions δ are the actions used inside the dynamic box modality $[\delta]C$ of $\mathcal{CL}$. Dynamic actions extend deontic actions with a set of Boolean tests denoted $\varphi?$ and the Kleene $*$ operator. The $*$ operator enables the clauses of the form $[\delta^*]C$, that is, whenever δ happens then C should be fulfilled.

Unlike the logic of G&M [28], P&S [50] presents a formal semantics (R1) in terms of an extended fragment of the propositional μ-calculus. P&S [51] maps $\mathcal{CL}$ to an algebra of actions for modeling concurrent contract actions and reasoning about violation. This tool seems more effective than that of G&M [27, 30] and Daskalopulu [13, 14].

We now show how the Example 1.1.1 is encoded in P&S's language.

Example 4.6.1. We introduce the following notational conveniences:

```
b = buyer buys contract for $5
p = buyer pays $80
g = seller gives stock to buyer
```

The contract encoding is presented below:

1 $\Box\mathbb{P}(b)$

2 $\Box[b]\mathbb{P}(p)$

3 $\Box[p]\mathbb{O}(g)$

Besides a formal semantics (R1), $\mathcal{CL}$ supports contract commitments (R3), action-dependent agreement (R4), and reparational agreements (R5). There is no account to absolute times (R2), dynamic calculation of values (R6), multiple parties (R7), external events (R8), instantaneous and continuous actions (R9), nor description of laws and regulations (R10).

4.6.2 Wyner's Approach

A. Wyner introduces in [58] the Abstract Contract Calculator — a flexible, open framework, and implemented tool to express the contractual notions of permissions, obligations, and prohibitions over abstract complex actions. The tool provides simple

and complex actions as state-transitions. In this work, deontically specified actions are reduced to fine-grained markers of violation and fulfillment. A contract is given as a contract state, which is a finite list of deontically specified actions relative to a party of the contract, along with a set of rules. The rules of a contract can modify a contract state relative to violation or fulfillment markers.

The main strength of the approach is that it allows alternative definitions of actions and deontic notions to be represented and animated. By using a dynamic language with violation and fulfillment markers, this approach can avoid many deontic paradoxes. Besides a formal semantics (R1), this approach also includes the aspects of contract commitments (R3), action-dependent agreements (R4), reparational agreements (R5), multiple parties (R7), and external events (R8). There is no account to absolute times (R2), dynamic calculation of values (R6), instantaneous and continuous actions (R9), nor the description of laws and regulations (R10).

4.6.3 OASIS LegalRuleML TC's Approach

OASIS has many technical committees working on domain-specific components for business documents. One of particular interest is the LegalRuleML Technical Committee (TC) [5, 6] who focuses on the creation of machine-readable forms of the content of legal texts, such as legislation, regulations, contracts, and case law, for different concrete Web applications.

OASIS LegalRuleML TC extends RuleML to provide a rule interchange language with formal features specific for the legal domain. The work enables implementers to structure the contents of the legal texts in a machine-readable format by using the representation tools provided.

Head/body relationship	Notation	Strength
body always head	body $\rightarrow$ head	strict
body sometimes head	body $\Longrightarrow$ head	defeasible
body not complement head	body head	defeater
body no relationship head		
body always complement head	body head	strict
body sometimes complement head	body head	defeasible
body not head	body head	defeater

RuleML is an XML-based family of language for expressing two categories of rules: reaction rules and derivation rules. The former are ECA based specifications, and the latter are used for describing facts. LegalRuleML reuses and extends concepts and syntax of RuleML. A rule of LegalRuleML is considered as a binary relationship between the pre-conditions (or body) of the rule, and the effect (head) of the rule. There are seven possible relationships between the body. The above table summaries the relationships and the strengths of the relationships.

Strict rules are used for concluding new facts. For a strict rule body $\rightarrow$ head the interpretation is that every time the body holds then the head holds. Defeasible rules are rules that may be overruled by some other rule. For a defeasible rule, it is possible to have exceptions to the rule, and it is possible to have prescriptions for the opposite conclusion. Defeater rules are used to specify that the opposite conclusion does not hold.

LegalRuleML introduces in [5] a suborder list that is a list of deontic formulas to model penalties. It utilizes the defeasible deontic logic to reason about violations of obligations.

We now show how the Example 1.1.1 is encoded in OASIS LegalRuleML.

Example 4.6.2. We have blocks for the specification of time:

```
<lrml:TimeInstants>
  <ruleml:Time key="t1">
    <ruleml:Data xsi:type="dateTime">
      2015-06-30-01T00:00:00
    </ruleml:Data>
  </rulem:Time>
</lrml:TimeInstants>
<lrml:TimeInstants>
  <ruleml:Time key="t2">
    <ruleml:Data xsi:type="dateTime">
      2015-12-17-01T00:00:00
    </ruleml:Data>
  </rulem:Time>
</lrml:TimeInstants>
```

The contract encoding is presented below:

```
<lrml:Statements key="option">
  <lrml:PrescriptiveStatement key="#ps1">
    <ruleml:Rule key=":rule1" closure="universal">
      <lrml:hasStrength>
        <lrml:Defeasible/>
      <lrml:hasStrength>
      <ruleml:if timsBlock="#t1">
        <ruleml:Atom>
          <ruleml:Rel iri="#buyOption"/>
          <ruleml:Var>buyer</ruleml:Var>
```

```
      </ruleml:Atom>
    </ruleml:if>
    <ruleml:then timesBlock="#t2">
      <lrml:Suborderlist>
        <lrml:Permission>
          <ruleml:Atom>
            <ruleml:Rel iri="#exerciseOption"/>
            <ruleml:Var>buyer</ruleml:Var>
          </ruleml:Atom>
        </lrml:Permission>
      </lrml:Suborderlist>
    </ruleml:then>
  </ruleml:Rule>
<lrml:PrescriptiveStatement key="#ps2">
  <ruleml:Rule key=":rule1" closure="universal">
    <lrml:hasStrength>
      <lrml:Defeasible/>
    <lrml:hasStrength>
    <ruleml:if timesBlock="#t2">
      <ruleml:Atom>
        <ruleml:Rel iri="#exerciseOption"/>
        <ruleml:Var>buyer</ruleml:Var>
      </ruleml:Atom>
    </ruleml:if>
    <ruleml:then>
```

```
      <lrml:Suborderlist>
        <lrml:Obligation>
          <ruleml:Atom timesBlock="#t3">
            <ruleml:Rel iri="#buyStock"/>
            <ruleml:Var>buyer</ruleml:Var>
            <ruleml:Ind>80</ruleml:Ind>
          </ruleml:Atom>
        </lrml:Obligation>
      </lrml:Suborderlist>
    </ruleml:then>
  </ruleml:Rule>
<lrml:PrescriptiveStatement key="#ps3">
  <ruleml:Rule key=":rule1" closure="universal">
    <lrml:hasStrength>
      <lrml:Defeasible/>
    <lrml:hasStrength>
    <ruleml:if>
      <ruleml:Atom>
        <ruleml:Rel iri="#buyStock"/>
        <ruleml:Var>buyer</ruleml:Var>
      </ruleml:Atom>
    </ruleml:if>
    <ruleml:then>
      <lrml:Suborderlist>
        <lrml:Obligation>
```

```
        <ruleml:Atom>
          <ruleml:Rel iri="#transferStock"/>
          <ruleml:Var>seller</ruleml:Var>
        </ruleml:Atom>
      </lrml:Obligation>
    </lrml:Suborderlist>
  </ruleml:then>
</ruleml:Rule>
```

Besides a formal semantics (R1), OASIS LegalRuleML supports absolute and relative times (R2), contract commitments (R3), action-dependent agreement (R4), reparational agreements (R5), multiple parties (R7), external events (R8), instantaneous and continuous actions (R9), and the description of laws and regulations (R10). There is no account to dynamic calculation of values (R6).

Chapter 5

Simple Type Theory

This chapter presents the underlying logic of FCL, a version of simple type theory based on the system STT presented in [19] and BESTT presented in [18]. Our version of simple type theory is also called STT. It includes additional new type constructors and expression constants for reasoning with sets and tuples. Section 5.1 introduces the syntax of STT and Section 5.2 gives the semantics of STT.

5.1 Syntax

5.1.1 Types

A *type* of STT is a string of symbols defined inductively by the formation rules below. Let $\alpha, \beta, \gamma, \ldots$ be syntactical variables ranging over types.

(1) A *base type* is a type of individuals or Bool, the type of propositions or truth values.

(2) For any α and β, $(\alpha \to \beta)$ is a *type of functions*.

(3) For any α and β, $(\alpha \times \beta)$ is a *type of ordered pairs*.

(4) For any α, $\mathsf{set}[\alpha]$ is a *type of sets*.

Let Ω denote the set of types of STT. The definition of a type shows that Ω is composed of the base types and an infinite hierarchy of function types, product types, and set types built from the base types. When there is no loss of meaning, a matching pair of parentheses in a type may be omitted. The function type $\rightarrow$ is assumed to associate to the right, so that $\alpha \rightarrow \beta \rightarrow \gamma$ abbreviates $(\alpha \rightarrow (\beta \rightarrow \gamma))$.

5.1.2 Languages

Let $\mathcal{V}$ be a fixed infinite set of symbols which will be used to construct variables. A *language* of STT is a tuple $\mathcal{L} = (\mathcal{B}, \mathcal{C}, \tau)$ where:

- $\mathcal{B}$ is a set of symbols with $\mathsf{Bool} \in \mathcal{B}$ called the *base types* of $\mathcal{L}$. The types of $\mathcal{L}$ are all the types formed from $\mathcal{B}$ using the type formation rules given above. Let $\Omega_{\mathcal{L}}$ denote the set of types of $\mathcal{L}$.
- $\mathcal{C}$ is a set of symbols called the *constants* of $\mathcal{L}$. $\mathcal{C}$ includes the built-in constants given in the left column of Table 5.1 as well as possibly other constant symbols.
- $\mathcal{V}$ and $\mathcal{C}$ are disjoint.
- $\tau : \mathcal{C} \rightarrow \Omega_{\mathcal{L}}$ is a total function that assigns a type of $\mathcal{L}$ to each constant in $\mathcal{C}$. The definition of τ on the built-in constants is given in the right column of Table 5.1.

$\in_\alpha, \subset_\alpha, \subseteq_\alpha, \emptyset_\alpha, \cup_\alpha$ and $\cap_\alpha$ denote the usual set-theoretical operations for type $\alpha \in \Omega_{\mathcal{L}}$.

Table 5.1: The Built-In Constant Symbols of $\mathcal{L}$

Constant c		TYPE $\tau(c)$
$\mathsf{make\text{-}pair}_{\alpha,\beta}$	for all $\alpha, \beta \in \Omega_{\mathcal{L}}$	$(\alpha \to \beta \to (\alpha \times \beta))$
$\mathsf{first}_{\alpha,\beta}$	for all $\alpha, \beta \in \Omega_{\mathcal{L}}$	$((\alpha \times \beta) \to \alpha)$
$\mathsf{second}_{\alpha,\beta}$	for all $\alpha, \beta \in \Omega_{\mathcal{L}}$	$((\alpha \times \beta) \to \beta)$
$\mathsf{make\text{-}set}_{\alpha}$	for all $\alpha \in \Omega_{\mathcal{L}}$	$((\alpha \to \mathsf{Bool}) \to \mathsf{set}[\alpha])$
$\in_{\alpha}$	for all $\alpha \in \Omega_{\mathcal{L}}$	$((\alpha \times \mathsf{set}[\alpha]) \to \mathsf{Bool})$
$\subset_{\alpha}$	for all $\alpha \in \Omega_{\mathcal{L}}$	$((\mathsf{set}[\alpha] \times \mathsf{set}[\alpha]) \to \mathsf{Bool})$
$\subseteq_{\alpha}$	for all $\alpha \in \Omega_{\mathcal{L}}$	$((\mathsf{set}[\alpha] \times \mathsf{set}[\alpha]) \to \mathsf{Bool})$
$\emptyset_{\alpha}$	for all $\alpha \in \Omega_{\mathcal{L}}$	$\mathsf{set}[\alpha]$
$\cup_{\alpha}$	for all $\alpha \in \Omega_{\mathcal{L}}$	$((\mathsf{set}[\alpha] \times \mathsf{set}[\alpha]) \to \mathsf{set}[\alpha])$
$\cap_{\alpha}$	for all $\alpha \in \Omega_{\mathcal{L}}$	$((\mathsf{set}[\alpha] \times \mathsf{set}[\alpha]) \to \mathsf{set}[\alpha])$

5.1.3 Expressions

Let $\mathcal{L} = (\mathcal{B}, \mathcal{C}, \tau)$ be a language of STT. There are six kinds of expressions in STT. An *expression of type* of α of STT is a string of symbols defined inductively by the formation rules below. Let $x, y, z, f, g, h, \ldots$ denote members of $\mathcal{V}$ and c denote a *constant*. $\mathbf{expr}_{\mathcal{L}}[A, \alpha]$ asserts that A is an expression of type α of $\mathcal{L}$.

Variable:

$$\frac{x \in \mathcal{V}, \alpha \in \Omega_{\mathcal{L}}}{\mathbf{expr}_{\mathcal{L}}[(x : \alpha), \alpha]}$$

Constant:

$$\frac{c \in \mathcal{C}, \alpha = \tau(c)}{\mathbf{expr}_{\mathcal{L}}[c, \alpha]}$$

Function application:

$$\frac{\mathbf{expr}_{\mathcal{L}}[A, \alpha], \mathbf{expr}_{\mathcal{L}}[f, \alpha \to \beta]}{\mathbf{expr}_{\mathcal{L}}[f(A), \beta]}$$

Function abstraction:

$$\frac{x \in \mathcal{V}, \alpha \in \Omega_{\mathcal{L}}, \mathbf{expr}_{\mathcal{L}}[B, \beta]}{\mathbf{expr}_{\mathcal{L}}[(\lambda x : \alpha \,.\, B), (\alpha \to \beta)]}$$

Equality:

$$\frac{\mathbf{expr}_{\mathcal{L}}[A, \alpha], \mathbf{expr}_{\mathcal{L}}[B, \alpha]}{\mathbf{expr}_{\mathcal{L}}[(A = B), \mathsf{Bool}]}$$

Definite description:

$$\frac{x \in \mathcal{V}, \alpha \in \Omega_{\mathcal{L}}, \mathbf{expr}_{\mathcal{L}}[A, \mathsf{Bool}]}{\mathbf{expr}_{\mathcal{L}}[(\mathrm{I}\, x : \alpha \,.\, A), \alpha]}$$

Expressions of type Bool are called *propositions* or *formulas*. A *predicate* is an expression of STT of type $(\alpha \to \mathsf{Bool})$ for any $\alpha \in \Omega_{\mathcal{L}}$. An *equation* is an expression of the form $A = B$ (where A and B are expressions of the same type). An occurrence of a variable $(x : \alpha)$ is called *bound* if it occurs within the scope of a textually enclosing $\lambda\, x : \alpha$ or $\mathrm{I}\, x : \alpha$; otherwise the occurrence is called *free*. An expression in which all occurrences of variables are bound is called a *closed expression*. A *sentence* of $\mathcal{L}$ is a closed formula of $\mathcal{L}$.

A *substitution* is a function from a finite set of variables to expressions. Let $v_1, v_2, \ldots, v_n$ denote variables of the form $(x_1 : \alpha), (x_2 : \alpha), \ldots, (x_n : \alpha)$, respectively. We write a substitution σ on $\{v_1, v_2, \ldots, v_n\}$ as $[v_1 \mapsto A_1, v_2 \mapsto A_2, \ldots, v_n \mapsto A_n]$ where $\sigma(v_i) = A_i$ for $i = 1, 2, \ldots n$. The application of σ to an expression e, written as $e\sigma$, is the result of simultaneously replacing all free occurrences of the variables $v_1, v_2, \ldots, v_n$ in the expression e with the expressions $\sigma(v_1), \sigma(v_2), \ldots, \sigma(v_n)$, respectively.

We will often use the following abbreviation rules to write expressions in a more compact form:

(1) A variable $(x : \alpha)$ occurring in the body B of $(\Box\, x : \alpha\, .\, B)$, where $\Box \in \{\lambda, \forall, \exists, \mathrm{I}\}$ may be written as x if there is no resulting ambiguity.

(2) A variable $(x : \alpha)$ occurring freely in an expression may be written as x if it is clear from the rest of expression what its type must be.

(3) A matching pair of parentheses in an expression may be omitted if there is no loss of meaning.

5.1.4 Definitions and Abbreviations

Let $A_\alpha, B_\alpha, C_\alpha, \ldots$ denote syntactic variables ranging over expressions of type α of $\mathcal{L}$. We will assume that $\varphi, \psi, \ldots$ denote formulas, i.e., expressions of type Bool. We introduce the following definitions and abbreviations:

true	means	$(\lambda\, x : \mathsf{Bool}\, .\, x) = (\lambda\, x : \mathsf{Bool}\, .\, x)$.
false	means	$(\lambda\, x : \mathsf{Bool}\, .\, \mathsf{true}) = (\lambda\, x : \mathsf{Bool}\, .\, x)$.
$\neg\varphi$	means	$(\varphi = \mathsf{false})$.
$(A_\alpha \neq B_\alpha)$	means	$\neg(A_\alpha = B_\alpha)$.
$(\varphi \wedge \psi)$	means	$(\lambda\, f : \alpha\, .\, f(\mathsf{true})(\mathsf{true})) = (\lambda\, f : \alpha\, .\, f(\varphi)(\psi))$ where $\alpha = (\mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool})$.
$(\varphi \vee \psi)$	means	$\neg(\neg\varphi \wedge \neg\psi)$.
$(\varphi \supset \psi)$	means	$(\neg\varphi \vee \psi)$.
$(\varphi \Leftrightarrow \psi)$	means	$(\varphi = \psi)$.
$(\forall\, x : \alpha\, .\, \varphi)$	means	$(\lambda\, x : \alpha\, .\, \varphi) = (\lambda\, x : \alpha\, .\, \mathsf{true})$.
$(\exists\, x : \alpha\, .\, \varphi)$	means	$\neg(\forall\, x : \alpha\, .\, \neg\varphi)$.
$\Box\, x_1 : \alpha_1, \ldots, x_n : \alpha_n\, .\, \varphi$	means	$(\Box\, x_1 : \alpha_1\, .\, (\Box\, x_2 : \alpha_2, \ldots, x_n : \alpha_n\, .\, \varphi))$ where $\Box \in \{\lambda, \forall, \exists\}$ and $n \geq 2$.

(A_α, B_β)	means	$(\mathsf{make\text{-}pair}_{\alpha,\beta}(A_\alpha)(B_\beta))$.
$(A_{\alpha_1}, \ldots, A_{\alpha_n})$	means	$(A_{\alpha_1}, (A_{\alpha_2}, \ldots, A_{\alpha_n}))$
		where $n \geq 3$.
$(\alpha_1 \times \cdots \times \alpha_n)$	means	$(\alpha_1 \times (\alpha_2 \times \cdots \times \alpha_n))$
		where $n \geq 3$.
$f_\beta(A^1_{\alpha_1}, \ldots, A^n_{\alpha_n})$	means	$f_\beta((A^1_{\alpha_1}, \ldots, A^n_{\alpha_n}))$
		where $n \geq 2$ and $\beta = ((\alpha_1 \times \cdots \times \alpha_n) \to \gamma)$.
$\#1(A^1_{\alpha_1}, \ldots, A^n_{\alpha_n})$	means	$(\mathsf{first}_{\alpha_1,\beta}((A^1_{\alpha_1}, \ldots, A^n_{\alpha_n}))$
		where $n \geq 2$ and $\beta = (\alpha_2 \times \cdots \times \alpha_n)$.
$\#m(A^1_{\alpha_1}, \ldots, A^n_{\alpha_n})$	means	$\#(m-1)(\mathsf{second}_{\alpha_1,\beta}((A^1_{\alpha_1}, \ldots, A^n_{\alpha_n})))$
		where $2 \leq m \leq n$, and
		$\beta = (\alpha_2 \times \cdots \times \alpha_n)$.
$\mathsf{if}(\varphi, A_\alpha, B_\alpha)$	means	$(\mathrm{I}\, x : \alpha \,.\, ((\varphi \supset ((x : \alpha) = A_\alpha)) \wedge$
		$(\neg\varphi \supset ((x : \alpha) = B_\alpha))))$
		where $(x : \alpha)$ does not occur in φ, A_α or B_α.
$\{(x : \alpha) \mid \varphi\}$	means	$(\mathsf{make\text{-}set}_\alpha((\lambda\, x : \alpha \,.\, \varphi)))$.
$(A_\alpha \in B_\beta)$	means	$(\in_\alpha (A_\alpha, B_\beta))$
		where $\beta = \mathsf{set}[\alpha]$.

5.2 Semantics of STT

5.2.1 Frame

A *frame* for a language $\mathcal{L} = (\mathcal{B}, \mathcal{C}, \tau)$ of STT is a collection $\{D_\alpha : \alpha \in \Omega_\mathcal{L}\}$ of nonempty domains such that:

(1) $D_{\mathsf{Bool}} = \{\mathsf{T}, \mathsf{F}\}$.

(2) For $\alpha, \beta \in \Omega_\mathcal{L}$, $D_{\alpha\to\beta}$ is the set of all *total* functions mapping D_α to D_β.

(3) For $\alpha, \beta \in \Omega_\mathcal{L}$, $D_{\alpha\times\beta}$ is the set of all ordered pairs (a, b) such that $a \in D_\alpha$ and $b \in D_\beta$.

(4) For $\alpha \in \Omega_\mathcal{L}$, $D_{\mathsf{set}\{\alpha\}}$ is the power set of D_α

D_{Bool} is the *domain of truth values*; for $\alpha \in \mathcal{B}\setminus\mathsf{Bool}$, D_α is a *domain of individuals*; and, for $\alpha, \beta \in \Omega_\mathcal{L}$, $D_{\alpha\to\beta}$ is a *function domain*, $D_{\alpha\times\beta}$ is an *ordered pair domain*, and $D_{\mathsf{set}\{\alpha\}}$ is a *set domain*.

5.2.2 Interpretations

A model[1] $\mathcal{M}$ of STT for a language $\mathcal{L} = (\mathcal{B}, \mathcal{C}, \tau)$ is a triple $\mathcal{M} = (\{D_\alpha : \alpha \in \Omega_\mathcal{L}\}, I, e)$ consists of a frame, a function e that maps each $\alpha \in \Omega$ to a member of D_α, and an interpretation function I that maps each constant c in $\mathcal{C}$ to an element of $D_{\tau(c)}$, which is called the *denotation of* c, such that:

(1) For all $\alpha, \beta \in \Omega_\mathcal{L}$, $I(\mathsf{make\text{-}pair}_{\alpha,\beta})$ is the function $f \in D_{\alpha\to\beta\to(\alpha\times\beta)}$ such that, for all $d_1 \in D_\alpha$ and $d_2 \in D_\beta$, $f(d_1)(d_2) = (d_1, d_2)$.

[1] We are defining here the notion of a *standard model* as opposed to a general model.

(2) For all $\alpha, \beta \in \Omega_{\mathcal{L}}$, $I(\mathsf{first}_{\alpha,\beta})$ is the function $f \in D_{(\alpha\times\beta)\to\alpha}$ such that, for all $d_1 \in D_\alpha$ and $d_2 \in D_\beta$, $f((d_1, d_2)) = d_1$.

(3) For all $\alpha, \beta \in \Omega_{\mathcal{L}}$, $I(\mathsf{second}_{\alpha,\beta})$ is the function $f \in D_{(\alpha\times\beta)\to\beta}$ such that, for all $d_1 \in D_\alpha$ and $d_2 \in D_\beta$, $f((d_1, d_2)) = d_2$.

(4) For all $\alpha \in \Omega_{\mathcal{L}}$, $I(\mathsf{make\text{-}set}_\alpha)$ is the function $f \in D_{(\alpha\to\mathsf{Bool})\to\mathsf{set}[\alpha]}$ such that, for all $d_1 \in D_{\alpha\to\mathsf{Bool}}$, $f(d_1) = \{d_2 \in D_\alpha \mid d_1(d_2) = \mathsf{T}\}$.

(5) For all $\alpha \in \Omega_{\mathcal{L}}$, $I(\in)$ is the function $f \in D_{(\alpha\times\mathsf{set}[\alpha])\to\mathsf{Bool}}$ such that, for all $d_1 \in D_\alpha$ and $d_2 \in D_{\mathsf{set}[\alpha]}$, $f((d_1, d_2)) = \mathsf{T}$ if d_1 is a member of d_2.

(6) For all $\alpha \in \Omega_{\mathcal{L}}$, $I(\subset_\alpha)$ is the function $f \in D_{(\mathsf{set}[\alpha]\times\mathsf{set}[\alpha])\to\mathsf{Bool}}$ such that, for all $d_1 \in D_{\mathsf{set}[\alpha]}$ and $d_2 \in D_{\mathsf{set}[\alpha]}$, $f((d_1, d_2)) = \mathsf{T}$ if d_1 is a proper subset of d_2.

(7) For all $\alpha \in \Omega_{\mathcal{L}}$, $I(\subseteq_\alpha)$ is the function $f \in D_{(\mathsf{set}[\alpha]\times\mathsf{set}[\alpha])\to\mathsf{Bool}}$ such that, for all $d_1 \in D_{\mathsf{set}[\alpha]}$ and $d_2 \in D_{\mathsf{set}[\alpha]}$, $f((d_1, d_2)) = \mathsf{T}$ if d_1 is a subset of d_2.

(8) For all $\alpha \in \Omega_{\mathcal{L}}$, $I(\emptyset_\alpha) = \{\}$, the empty set.

(9) For all $\alpha \in \Omega_{\mathcal{L}}$, $I(\cup_\alpha)$ is the function $f \in D_{(\mathsf{set}[\alpha]\times\mathsf{set}[\alpha])\to\mathsf{set}[\alpha]}$ such that, for all $d_1, d_2 \in D_{\mathsf{set}[\alpha]}$, $f((d_1, d_2)) = \{d_3 \in D_\alpha \mid d_3 \in d_1 \vee d_3 \in d_2\}$.

(10) For all $\alpha \in \Omega_{\mathcal{L}}$, $I(\cap_\alpha)$ is the function $f \in D_{(\mathsf{set}[\alpha]\times\mathsf{set}[\alpha])\to\mathsf{set}[\alpha]}$ such that, for all $d_1, d_2 \in D_{\mathsf{set}[\alpha]}$, $f((d_1, d_2)) = \{d_3 \in D_\alpha \mid d_3 \in d_1 \wedge d_3 \in d_2\}$.

An *assignment* into a frame $\{D_\alpha : \alpha \in \Omega_{\mathcal{L}}\}$ for $\mathcal{L}$ is a function θ whose domain is the set of variables of $\mathcal{L}$ such that, for each variable $(x : \alpha)$ for $\mathcal{L}$, $\theta(x : \alpha) \in D_\alpha$. Given an assignment θ, a variable $(x : \alpha)$, and $d \in D_\alpha$, let $\theta[(x : \alpha) \mapsto d]$ be the variable assignment θ' such that $\theta'((x : \alpha)) = d$ and $\theta'(X) = \varphi(X)$ for all variables $X \neq (x : \alpha)$.

The *valuation function* for $\mathcal{M}$ is the binary function V that takes as arguments an expression E of $\mathcal{L}$ and a variable assignment θ into the frame of $\mathcal{M}$ and that satisfies the six conditions below:

(1) Let E be a variable (i.e., $E \in \mathcal{V}$ of the form $(x : \alpha)$) of STT. Then $V_\theta^{\mathcal{M}}(E) = \theta(E)$.

(2) Let E be a constant (i.e., $E \in \mathcal{C}$) of STT. Then $V_\theta^{\mathcal{M}}(E) = I(E)$.

(3) Let E be of the form $f(A)$. Then $V_\theta^{\mathcal{M}}(E) = V_\theta^{\mathcal{M}}(f)(V_\theta^{\mathcal{M}}(A))$, the result of applying the function $V_\theta^{\mathcal{M}}(f)$ to the argument $V_\theta^{\mathcal{M}}(A)$.

(4) Let E be of the form $(\lambda\, x : \alpha\ .\ B)$ where B is of type β. Then $V_\theta^{\mathcal{M}}(E)$ is the function $f : D_\alpha \to D_\beta$ such that, for each $d \in D_\alpha$, $f(d) = V^{\mathcal{M}}_{\theta[(x:\alpha)\mapsto d]}(B)$.

(5) Let E be of the form $A = B$. If $V_\theta^{\mathcal{M}}(A) = V_\theta^{\mathcal{M}}(B)$, then $V_\theta^{\mathcal{M}}(E) = \mathsf{T}$; otherwise $V_\theta^{\mathcal{M}}(E) = \mathsf{F}$.

(6) Let E be of the form $(\mathrm{I}\, x : \alpha : A)$. If there is a unique $d \in D_\alpha$ such that $V^{\mathcal{M}}_{\theta[(x:\alpha)\mapsto d]}(A) = \mathsf{T}$, then $V_\theta^{\mathcal{M}}(E) = d$; otherwise $V_\theta^{\mathcal{M}}(E) = e(\alpha)$, the canonical error element for type α.

Let E be an expression of type α of $\mathcal{L}$ and φ be a formula. $V_\theta^{\mathcal{M}}(E)$ is called the *value* of E in $\mathcal{M}$ with respect to θ. For a closed expression E, $V_\theta(E)$ does not depend on θ. A formula φ of $\mathcal{L}$ is *valid* in $\mathcal{M}$ if $V_\theta^{\mathcal{M}}(\varphi) = \mathsf{T}$ for all variable assignments θ into $\mathcal{M}$.

A *theory* of STT is a pair $T = (\mathcal{L}, \Gamma)$ where $\mathcal{L}$ is a language of STT and Γ is a set of sentences of $\mathcal{L}$ called the *axioms* of T. T_0, T_1, T_2, etc. denote theories.

A *model* of T is a model $\mathcal{M}$ for $\mathcal{L}$ in which each member of Γ is valid in $\mathcal{M}$. T is *satisfiable* if there is some model for T. A formula φ of $\mathcal{L}$ is a *semantic consequence* of T, written as $T \models \varphi$, if φ is valid in every model for T.

In the rest of the thesis, let $\mathcal{L} = (\mathcal{B}, \mathcal{C}, \tau)$ be a language of STT and $T = (\mathcal{L}, \Gamma)$ be a theory of STT.

Chapter 6

FCL: The Syntax

In Chapter 5 we introduced STT, the underlying logic of FCL. FCL consists of several kinds of syntactic objects that include types and expressions from STT. In this chapter we give a full presentation of the syntax of FCL. We use a theory of STT as our base theory. On top of the STT theory, we build FCL. We assume that each concept and notation defined in Chapter 5 for STT is defined in FCL in the obvious way if it is not explicitly defined in this chapter.

FCL has a parameter T that is a theory of STT that extends the base theory. T includes any needed vocabulary and assumptions about the contracts that are being considered. We consider a contract in FCL to be a set of components (definitions, agreements, and rules) that can refer to "observables" and can include conditions that depend on "observables". Section 6.1 introduces $\mathcal{L}_0$, the base language of FCL. Section 6.2 gives the concepts of observables and actions of FCL. The last part of this chapter will focus on the presentation of FCL, a language based on $\mathcal{L}_0$ for expressing contracts.

6.1 The Language $\mathcal{L}_0$

Let $\mathcal{L}_0 = (\mathcal{B}_0, \mathcal{C}_0, \tau_0)$ be a language of FCL of STT where:

- $\mathcal{B}_0$ includes the following base types:

  (1) Int denotes a type of integers.

  (2) Time is an alias for Int. Hence, we assume that time is represented as a discrete linearly ordered set of values such that each value has a predecessor and a successor. The values many denote any convenient measure of time such as days, hours, seconds, etc.

  (3) String denotes a type of strings of a sequence of characters.

  (4) Event, Act, Party, Currency, and Goods are unspecified types. Event is a type of events. These can be actions performed by the parties of a contract as well as events that the parties have no control over. Party is a type of the parties of contracts. Currency is a type of the chosen unit of monetary exchange. Goods is a type of products that are shipped between parties. Act is a type of acts. An act performed by parties is called an *action*.

- $\mathcal{C}_0$ includes the constants given in the left column of Table 5.1, the constants of the integers operations given in Table 6.1, and the constants for writing contracts given in Table 6.2.

- The definition of τ on the constants in $\mathcal{C}_0$ is given in the right columns of Table 5.1, Table 6.1, and Table 6.2.

We assume $\mathcal{V}$ includes X_{time}. The variable $(X_{\text{time}} : \mathsf{Time})$ is used to instantiate expressions with the current time of a contract. Other base types and constants can be added to $\mathcal{B}_0$ and $\mathcal{C}_0$ as needed.

Table 6.1: The constants for Int of $\mathcal{L}_0$

CONSTANT c	TYPE $\tau(c)$
0	Int
1	Int
$+$	$((\mathsf{Int} \times \mathsf{Int}) \to \mathsf{Int})$
$-$	$((\mathsf{Int} \times \mathsf{Int}) \to \mathsf{Int})$
$*$	$((\mathsf{Int} \times \mathsf{Int}) \to \mathsf{Int})$
$<$	$((\mathsf{Int} \times \mathsf{Int}) \to \mathsf{Bool})$
$\leq$	$((\mathsf{Int} \times \mathsf{Int}) \to \mathsf{Bool})$

Table 6.2: The constants for contracts of $\mathcal{L}_0$

CONSTANT c	TYPE $\tau(c)$
t_{offer}	Time
obs-event	$((\mathsf{Time} \times \mathsf{Event}) \to \mathsf{Bool})$
transfer	$((\mathsf{Currency} \times \mathsf{Int}) \to \mathsf{Act})$
deliver	$((\mathsf{Goods} \to \mathsf{Act})$

The constants given in the left column of Table 6.1 denote the usual integer operations. t_{offer} is a time constant that is intended to represent the time a contract is offered. obs-event is used to express the observation of events as described in section 6.2. transfer is an unspecified constant that represents the transaction of monetary values. transfer(\$,5) means the action of transferring \$5. deliver is an unspecified constant that represents the delivery of goods or products.

We define $\mathsf{sub}(\varphi, t)$ to be the set of substitutions σ that map the free variables in the formula φ to appropriate closed expressions such that $\sigma(X_{\text{time}}) = t$.

Let $T_0 = (\mathcal{L}_0, \Gamma_0)$ be the theory of FCL of STT, where $\mathcal{L}_0 = (\mathcal{B}_0, \mathcal{C}_0, \tau_0)$ and Γ_0 is a set of sentences that include the usual axioms for integer constants given in Table 6.1. The constants t_{offer}, obs-event, and transfer are unspecified by T_0.

6.2 Observables

An *observable* is something that has a variable value that can be observed at a particular time [46, 47]. Let us look at a couple of examples. The temperature of a room is an observable. Its value at a given time t is the temperature measured in the room at t. An event is an observable whose value is either true or false. Its value at a given time t is true [false] if the event occurs [does not occur] at t.

Definition 6.1 (Observable)**.** In FCL, an *observable* is an application of the form

$$f(t, a_1, \ldots, a_n)$$

where f is a constant of type

$$\mathsf{Time} \times \alpha_1 \times \cdots \times \alpha_n \to \beta$$

with $n \geq 0$.

Thus the value of the observable $f(t, a_1, \ldots, a_n)$ depends on time in the sense that it depends on the value of its first argument which is of type Time. The value of $f(t, a_1, \ldots, a_n)$ can also depend on the parameters $a_1, \ldots, a_n$. Notice that the observable of FCL is closer to a Lisp quotation, or a PROLOG fact, than other things.

Definition 6.2 (Observation)**.** An *observation* of FCL is an equation

$$o = v$$

where:

- o is an observable $f(t, a_1, \ldots, a_n)$ of type $\mathsf{Time} \times \alpha_1 \times \cdots \times \alpha_n \to \beta$.

- v is an expression of type β that denotes a possible value of the observable o.

When the output type of f is Bool, $o = \mathsf{true}$ and $o = \mathsf{false}$ can be written as o and $\neg o$, respectively. An *observational statement* of FCL is a formula of STT constructed from observations using the machinery of STT.

The two examples of observables mentioned above can be expressed in FCL as Examples 6.2.1 and 6.2.2.

Example 6.2.1. Let obs-temp be a constant of type $\mathsf{Time} \to \mathsf{Int}$. Then $\mathsf{obs\text{-}temp}(t) = a$ represents the observation that the temperature in a particular room is a at time t.

Example 6.2.2. $\mathsf{obs\text{-}event}(t, e)$ represents the observation that the event e occurs at time t.

6.3 Actions

An *action* is an event that can be performed by the parties of a contract. There are two sorts of entities involved in an action: *subjects* and *objects*. The former are the entities who perform the action, while the latter are the entities that are acted upon by the subjects. The subjects and objects are parties of the contract.

Definition 6.3 (Action)**.** An *action* a of type Event is defined as a tuple of the form

$$(L, \mathsf{act}, \mathcal{S}, \mathcal{O})$$

where:

- L, of the type String, is the *label* of an action. Each action has a unique label.

- act, of the type Act, is the *act* of the action (i.e., the thing that is performed).
- $\mathcal{S}$, of the type set[Party], is the set of *subjects* of the action.
- $\mathcal{O}$, of the type set[Party], is the set of *objects* of the action.

Contracts typically include actions that specify the transfer of resources (money, goods, services, and even pieces of information) between parties. The act of the action would be the transfer of resources from one party (the subject) to another party (the object). Thus an action of this kind encodes both what is transferred and what parties are involved in the transference. Notice that there will be predicates defined on Action.

6.4 FCL

FCL is a language with a parameter $T = (\mathcal{L}, \Gamma)$ that is a theory of STT that extends the base theory T_0 such that $\mathcal{L}_0 \subseteq \mathcal{L}$ and $\Gamma_0 \subseteq \Gamma$. T is a theory of STT that serves as a basis for defining the following components of FCL:

(1) Constant definitions.

(2) Agreements.

(3) Rules.

(4) Contracts.

T includes the machinery of T_0 as well as any additional vocabulary and assumptions about the contracts that are being considered. A type of FCL (with parameter

T) is any type of the base language $\mathcal{L}$ and an expression of FCL (with parameter T) is any expression of $\mathcal{L}$.

In the following sections we will explain the components of FCL and conclude the syntax of FCL.

6.4.1 Constant Definitions

Before presenting the syntax of FCL, we introduce the concepts of constant definition as the definitions below. Let α and A be a type and an expression of FCL, respectively.

Definition 6.4 (Simple Constant Definition)**.** A *simple constant definition* of FCL is an expression **D** of the form

$$c = A_\alpha$$

where:

- c is a new constant (i.e., a constant not in $\mathcal{C}$).
- $\tau(c) = \alpha$.
- A_α is an expression of FCL (with possible new constants that are previously defined) that defines the value of c.

Definition 6.5 (Constant Definition with Time Parameter)**.** A *constant definition with a time parameter* of FCL is an expression **D** of the form

$$f(t) = A_\alpha$$

where:

- f is a new constant.

- $\tau(f) = \mathsf{Time} \to \alpha$.
- t is an expression of type Time.
- A_α is an expression of FCL (with possible new constants) that defines the value of f.
- $f(t)$ is treated as a new constant in the semantics for FCL.

Note: This last kind of definition is not a standard definition of a function. It defines the value of f at just a single value, namely, t. Other definitions can define f at other values. Thus f will only particularly defined in a particular state of a contract.

Constant definitions are used, among other things, to define temporally based values. The expressions in a contract may contain the new constants given in constant definitions.

6.4.2 Agreements

An *agreement* is a promise to do or not do a specific action. It creates obligations or prohibitions for the subjects of the agreement.

Definition 6.6 (Agreement)**.** An *agreement* **A** of FCL is an expression of FCL (with possible new constants) of either the form

$$\mathbb{O}(a, \mathcal{T}) \quad \text{or} \quad \mathbb{F}(a, \mathcal{T})$$

where:

- a is an action, i.e., an expression denoting an action.

- $\mathcal{T}$ is a set of times, i.e., an expression of type $\mathsf{set}[\mathsf{Time}]$.
- The operators $\mathbb{O}$ and $\mathbb{F}$ are inspired by the deontic operators for *obligation* and *prohibition* [42].

$\mathbb{O}(a, \mathcal{T})$ is called an *obligation*; it represents the promise that the action a will be observed at some time in $\mathcal{T}$. $\mathbb{F}(a, \mathcal{T})$ is called a *prohibition*; it represents the promise that the action a will not be observed at any time in $\mathcal{T}$. $\mathbb{O}(a, \mathcal{T})$ and $\mathbb{F}(a, \mathcal{T})$ are considered to be *duals* of each other.

We interpret an agreement in a contract inspired by the terms of deontic *obligation* and *prohibition*. These concepts are utilized in expressions involving actions that are executed by the parties of the contract. Thus, the concepts express what a party *ought to do* or *ought not do* [37]. Permissions will be handled differently from obligations and prohibitions. See Chapter 8 for details of how permissions are handled in our language.

Obligations and prohibitions have the distinctive characteristic of being violable. When a promise made in a contract is honored, we say the promise has been *satisfied*. If it has not been honored, we say it has been *violated*. A promise may be restricted by a temporal bound, that is, a period of time during which an obligation or prohibition is in force. For example, a tenant may be obliged to pay rent on the first day of each month. Thus violation of an obligation may arise in two ways [7]: a party obliged to perform an action may fail to do so or the party performs a temporally bounded obligatory action but not within the time period specified. Violation of a prohibition may arise in just one way: a party may perform a temporally bounded prohibited action during the time period specified. A permission will never be violated but may expire: a party may fail to perform a temporally bounded permitted action within the time period specified.

Notice that a promise may not be restricted by a temporal bound. In this case, an obligation can never be violated, and a prohibition can never be satisfied.

6.4.3 Rules

Definition 6.7 (Rule)**.** A *rule* **R** of FCL is inductively defined as an expression of the form

$$\varphi \mapsto \mathcal{Q}$$

where:

- φ is a formula of FCL (with possible new constants).
- $\mathcal{Q}$ is a set of constant definitions, agreements, and rules.

We assume that each free variable occurring in a constant definition or an agreement in $\mathcal{Q}$ also occurs in φ. Note that this assumption is not made for the constant definitions and agreements in the rules of $\mathcal{Q}$.

The condition φ of a rule is an assertion that depends on the values of observables. It can be an assertion that something has happened. We will see in Chapter 7 that, if φ is satisfied at time t, the members of $\mathcal{Q}$ are added to the state of a contract at time $t + 1$. Thus a rule can dynamically change the state of a contract.

Definition 6.8 (Conditional Agreement)**.** A *conditional agreement* of FCL is a rule of the form

$$\varphi \mapsto \{\mathbf{A}\}$$

where **A** is an agreement.

Thus, a *conditional agreement* is an agreement that comes into effect only when a certain condition is satisfied.

6.4.4 Contracts

We think of a contract in two ways. Syntactically, a contract is a written expression satisfying certain syntactic conditions. Semantically, a contract is a set of temporally based promises between parties.

Definition 6.9 (Contract)**.** A *contract* **C** of FCL is a pair

$$(t_0, \mathcal{Q})$$

where:

- t_0 is an expression of type Time representing the time the contract is offered to the parties.
- $\mathcal{Q}$ is a set of constant definitions, agreements, and rules.

The *parties* of **C** are the parties mentioned in the rules in $\mathcal{Q}$.

6.4.5 The Syntax of FCL

Definition 6.10 (FCL Syntax)**.** The syntax of FCL is defined by the following grammar:

$\mathbf{D}$	:=	$c = A_\alpha \mid f(t) = A_\alpha$	
$\mathbf{A}$	:=	$\mathbb{O}(a, \mathcal{T}) \mid \mathbb{F}(a, \mathcal{T})$	where a denotes an action and $\mathcal{T}$ denotes a set of times.
$\mathbf{R}$	:=	$\varphi \mapsto \mathcal{Q}$	where $\mathcal{Q} = \{\mathbf{D}_1, \ldots, \mathbf{D}_k, \mathbf{A}_1, \ldots, \mathbf{A}_m, \mathbf{R}_1, \ldots, \mathbf{R}_n\}$ with $k, m, n \geq 0$.
$\mathbf{C}$	:=	$(t, \mathcal{Q})$	where t is an expression of type Time.

We use agreements to describe agreements that are independent of time and events and use rules to describe how to add new agreements to a contract over time. Conditions are assertions that depend on events. As we will see in the next chapter, a contract has a "state" consisting of a set of constant definitions, agreements, and rules. The state evolves over time like the state of a program evolves over time. A contract is "fulfilled" when all the agreements in its state are satisfied and all the rules in its state are no longer applicable. A contract is "breached" when some agreement in its state is violated.

Chapter 7

FCL: The Semantics

In FCL, a contract is a set of components (definitions, agreements, and rules) that can refer to "observables" and can include conditions that depend on "observables". The meaning of these components can change when the values of observables mentioned in them change, and new components can be added when triggering conditions become true. Hence the state of a contract as a set of components can evolve over time in much the same way as the state of a computer program evolves over time.

In Chapter 6 we introduced the syntax of FCL. This chapter presents the formal semantics of FCL. Section 7.1 introduces a model of FCL. Section 7.2 and 7.3 introduce the semantics of agreements and rules in contracts. Section 7.4 explains how the state of a contract changes.

7.1 Models

A *model* of FCL with parameter $T = (\mathcal{L}, \Gamma)$ is a model of T as an STT theory. Fix a model $\mathcal{M} = (\{D_\alpha : \alpha \in \Omega_\mathcal{L}\}, I, e)$ for T. Let $V^\mathcal{M}$ be the valuation function of $\mathcal{M}$ that

assigns each (closed) expression of $\mathcal{L}$ a value in $\mathcal{M}$. In particular, $V^{\mathcal{M}}$ assigns each observable $f(t, a_1, \ldots, a_n)$ a value for all times t (and parameters $a_1, \ldots, a_n$). Thus a model includes the values for all observables at all times.

7.2 Agreements

Let t be an expression of type Time. The *value* of an obligation $\mathbb{O}(a, \mathcal{T})$ in $\mathcal{M}$ at time t is $V^{\mathcal{M}}(\varphi)$ where φ is the formula

$$\exists\, u : \mathsf{Time}\ .\ u \in \mathcal{T} \wedge u \leq t \wedge \mathsf{obs\text{-}event}(u, a).$$

The *value of a prohibition* $\mathbb{F}(a, \mathcal{T})$ *in* $\mathcal{M}$ *at time* t is $V^{\mathcal{M}}(\psi)$ where ψ is the formula

$$\forall\, u : \mathsf{Time}\ .\ u \in \mathcal{T} \supset (u \leq t \wedge \neg \mathsf{obs\text{-}event}(u, a)).$$

An agreement is *satisfied* in $\mathcal{M}$ at time t if its value in $\mathcal{M}$ at t is T. An agreement is *violated* in $\mathcal{M}$ at time t if the value of its dual in $\mathcal{M}$ at t is T. We will occasionally use an agreement $\mathbb{O}(a, \mathcal{T})$ or $\mathbb{F}(a, \mathcal{T})$ as a formula of FCL whose meaning is φ or ψ, respectively.

7.3 Rules

Let $\mathbf{R} = \varphi \mapsto \mathcal{Q}$ be a rule of FCL, t be an expression of type Time, $\mathsf{sub}(\varphi, t)$ be the set of substitutions defined in Section 6.1. The variable X_{time} is used to instantiate a rule with the current time of a contract. For any $Q \in \mathcal{Q}$ and substitution $\sigma \in \mathsf{sub}(\varphi, t)$, let $Q|\sigma$ be $Q\sigma$ if Q is a constant definition or agreement and be $Q\sigma'$ if Q is a rule where

$\sigma' = \sigma[X_{\text{time}} \to X_{\text{time}}]$. Then define $\mathsf{new\text{-}items}(R, \mathcal{M}, t)$ to be

$$\{Q|\sigma \,|\, \sigma \in \mathsf{sub}(\varphi, t) \wedge V^{\mathcal{M}}(\varphi\sigma) = \mathsf{T} \wedge Q \in \mathcal{Q}\}.$$

R is *active* in $\mathcal{M}$ at t if $V^{\mathcal{M}}(\varphi\sigma) = \mathsf{T}$ for some $\sigma \in \mathsf{sub}(\varphi, t)$. **R** is *defunct* in $\mathcal{M}$ at t if $V^{\mathcal{M}}(\varphi\sigma) = \mathsf{F}$ for all $u \geq t$ and all $\sigma \in \mathsf{sub}(\varphi, u)$. If **R** is defunct in $\mathcal{M}$ at t, then **R** is not active in $\mathcal{M}$ at u and $\mathsf{new\text{-}items}(\mathbf{R}, \mathcal{M}, u) = \emptyset$ for all $u \geq t$.

7.4 Contracts

7.4.1 Expansion of Models

Before we present the semantics of contracts, we introduce the concept of *expansion* as the definition given below.

Definition 7.1 (Sub-language)**.** Let $\mathcal{L} = (\mathcal{B}, \mathcal{C}, \tau)$ and $\mathcal{L}' = (\mathcal{B}, \mathcal{C}', \tau')$. $\mathcal{L}$ is a *sub-language* of $\mathcal{L}'$ written as $\mathcal{L} \subseteq \mathcal{L}'$ if $\mathcal{C} \subseteq \mathcal{C}'$ and τ is a sub-function of τ'.

Definition 7.2 (Expansion)**.** If $\mathcal{L} \subseteq \mathcal{L}'$ and $\mathcal{M} = (D, I)$ is a model of $\mathcal{L}$, then a model $\mathcal{M}' = (D, I')$ of $\mathcal{L}'$ is an *expansion* of $\mathcal{M}$ to $\mathcal{L}'$ if I is a sub-function of I'.

7.4.2 State of Contracts

In FCL, the state of a contract is a set of constant definitions, agreements, and rules. A contract could involve dynamic processes. As certain events happened, more and more components of a contract are put into effect. The contract is progressively realized in this way.

Definition 7.3 (State of a Contract)**.** Let $\mathbf{C} = (t_0, \mathcal{Q})$ be a contract. The *state* of $\mathbf{C}$ in $\mathcal{M}$ at time t, written $\mathsf{state}(\mathbf{C}, \mathcal{M}, t)$, is the set of constant definitions, agreements, and rules defined inductively as follows:

(1) $\mathsf{state}(\mathbf{C}, \mathcal{M}, t_0) = \mathcal{Q} \cup \{\mathbf{D}\}$, where $\mathbf{D}$ is the constant definition $t_{\text{offer}} = t_0$.

(2) If $t \geq t_0$, then $\mathsf{state}(\mathbf{C}, \mathcal{M}, t+1) =$

$$(\mathsf{state}(\mathbf{C}, \mathcal{M}, t)) \cup \bigcup_{\mathbf{R} \in \mathcal{Q}} \mathsf{new\text{-}items}(\mathbf{R}, \mathcal{M}', t)$$

where $\mathcal{M}'$ is the smallest expansion of $\mathcal{M}$ such that $V^{\mathcal{M}'}(\mathbf{D}) = \mathsf{T}$ for each constant definition $\mathbf{D} \in \mathsf{state}(\mathbf{C}, \mathcal{M}, t)$.

The model $\mathcal{M}'$ in clause 2 is called the $\mathbf{C}$-*expansion of* $\mathcal{M}$ *at time* t. A *model of the contract* $\mathbf{C}$ *at time* t is any $\mathbf{C}$-expansion of a model of FCL at time t.

Note: We are employing a very simple model of concurrency in our semantics for contracts: At each time t, all active rules are applied simultaneously and the resulting new components – constant definitions, agreements, and rules — are added to the state of the contract at the next point in time. There is no opportunity for the application of rules to interfere with each other. In particular, a component can never be removed from the state once it is added to it.

Definition 7.4 (Fulfilled Contract)**.** A contract $\mathbf{C}$ is *fulfilled* in $\mathcal{M}$ at time t where $V^{\mathcal{M}}(t) \geq V^{\mathcal{M}}(t_0)$ if every agreement in $\mathsf{state}(\mathbf{C}, \mathcal{M}, t)$ is satisfied in the $\mathbf{C}$-expansion of $\mathcal{M}$ at t and every rule in $\mathsf{state}(\mathbf{C}, \mathcal{M}, t)$ is defunct in the $\mathbf{C}$-expansion of $\mathcal{M}$ at t.

Definition 7.5 (Breached Contract)**.** A contract $\mathbf{C}$ is *breached* in $\mathcal{M}$ at time t where $V^{\mathcal{M}}(t) \geq V^{\mathcal{M}}(t_0)$ if there is an agreement in $\mathsf{state}(C, \mathcal{M}, t)$ that is violated in the $\mathbf{C}$-expansion of $\mathcal{M}$ at t.

Definition 7.6 (Null Contract)**.** A contract $\mathbf{C}$ is *null* in $\mathcal{M}$ at time t where $V^{\mathcal{M}}(t) \geq V^{\mathcal{M}}(t_0)$ if $\mathsf{state}(\mathbf{C}, \mathcal{M}, t)$ contains no agreements and every rule in $\mathsf{state}(\mathbf{C}, \mathcal{M}, t)$ is defunct in the $\mathbf{C}$-expansion of $\mathcal{M}$ at t.

Notice that once a contract is fulfilled [breached], it is fulfilled [breached] forever. In some cases, a contract may have agreements that can never be fulfilled, and then we call it an *unfulfillable* contract. Of course, breached contracts are unfulfillable.

Chapter 8

Examples expressed in FCL

In this chapter, we will first explain the concepts of permissions and reparations of contracts and how these concepts can be expressed in FCL, and then formalize in FCL the examples introduced in Chapter 3.

8.1 FCL: Expressivity Issues

8.1.1 Permissions

An agreement is an understanding between parties. It creates obligations, permissions, or prohibitions to do the specific things that are the subject of that agreement. Agreements of the form $\mathbb{O}(a, \mathcal{T})$ and $\mathbb{F}(a, \mathcal{T})$ are used in FCL to represent promises in the form of obligations and prohibitions. Notice that agreements in FCL do not include expressions formed using an operator corresponding to the deontic operator $\mathbb{P}$ for *permission*.

Unlike an obligation or a prohibition, a permission is not a promise. We think a

permission given to one party can be expressed as an obligation (or prohibition) given to a second party. We recall Example 1.1.1 to illustrate this. The clause 2 of this example is a permission, saying that the buyer is permitted to exercise the option on December 17, 2015. This permission can be considered as a conditional agreement, saying that if the option buyer exercise the option on time, then the option seller is obligated to transfer one share of stock to the option buyer within the deadline.

In FCL, one party's permission to perform an action a will often be expressed by:

(1) The absence of an obligation to perform a;

(2) The absence of a prohibition to perform a; and

(3) A conditional agreement that says, if a is performed, then another party is obligated to perform some action b.

Thus, a permitted action by one party will be expressed as a rule that can create a second party's obligation in response to the performance of this action.

In this case, some kinds of permissions can be expressed in FCL. Consider Example 1.1.1 in which the option buyer is permitted to exercise the call option. This permission is expressed by adding a rule $\varphi \mapsto \mathcal{B}$ to the contract's state where the condition φ holds if it is observed that the buyer of the option exercises the option and $\mathcal{B}$ includes an obligation that the seller of option sells to the buyer the goods specified by the option. See Section 8.2 for details.

8.1.2 Reparations

A contract usually specifies actions to be taken in case of the violation of a part of the contract. A conditional obligation arising in response to a violated agreement is

considered as a *reparational agreement*. To illustrate how a violation that arises in a contract can be "repaired", we recall Example 3.2.1.

Note that clause 3 of this example is a primary obligation, saying that the buyer is obligated to pay the second half within 30 days of delivery. Clause 4 is an example of *reparational obligation* in which an unfulfilled obligation can generate obligations to "repair" this violation. It says what the buyer is obligated to do if he or she violates the primary obligation.

Similar to the reparational obligation, a *reparational prohibition* is a conditional agreement arising in response to a violated prohibition. Both the reparational obligations and reparational prohibitions are reparational agreements. In FCL, a reparational agreement will be expressed as a rule that can create other agreements or rules in response to the violation of the primary agreement. To express the potential violations, we introduce $\mathsf{obs\text{-}event\text{-}during}(t, e, \mathcal{T})$ to represent the observation at time t that the event e occurred during the time period $\mathcal{T}$.

In FCL, a reparational obligation of $\mathbb{O}(a, \mathcal{T})$ is expressed as a rule of the form

$$\neg\mathsf{obs\text{-}event\text{-}during}(X_{\text{time}}, a, \mathcal{T}) \mapsto \mathcal{B}.$$

The expression $\neg\mathsf{obs\text{-}event\text{-}during}(t, a, \mathcal{T})$ represents a potential violation of agreement $\mathbb{O}(a, \mathcal{T})$. If an obligation is satisfied, the rule to "repair" this obligation will never be active. Similarly, a reparational prohibition of $\mathbb{F}(a, \mathcal{T})$ is expressed as a rule of the form

$$\mathsf{obs\text{-}event\text{-}during}(X_{\text{time}}, a, \mathcal{T}) \mapsto \mathcal{B}.$$

A rule that represents a reparational prohibition of $\mathbb{F}(a, \mathcal{T})$ will always be defunct if the agreement $\mathbb{F}(a, \mathcal{T})$ is satisfied.

Consider clause 4 of Example 3.2.1, the reparational obligation of the primary obligation given in clause 3 is the conditional agreement that, if the second half of payment has not been observed within 30 days of delivery, then the buyer has to pay an additional fine of 10% within 14 days. How this conditional agreement is expressed in FCL is shown in Section 8.3.

Notice that the violation of an agreement will make a breach of a contract. If a contract is reparable, then every related agreement that can "repair" the breach will be considered as a reparational agreement and represented as a rule.

8.1.3 Continuous Actions

We need a proper formalization of time and temporal information of events as noted in Section 3.1. In FCL, a continuous action that is performed throughout time $\mathcal{T}$ is expressed as a repeated action that is performed at each time point during $\mathcal{T}$. Consider clause 1 of Example 3.2.2, the providing of an apartment to the tenant is a continuous action that should be performed by the landlord throughout the term of the lease. How this action is expressed in FCL is shown in Section 8.4.

8.1.4 Avoid Paradoxes

As described in Chapter 2 the standard deontic logic suffers from a number of well-known paradoxes. These paradoxes are not actually logical paradoxes in the normal sense. They are more or less confusions between the formal deontic concepts and natural language. Although the operators $\mathbb{O}$ and $\mathbb{F}$ of FCL are inspired by the deontic operators for obligation and prohibition, they are not deontic operators. Thus the paradoxes in deontic logic that arise from these operators do not occur in FCL. For

example, the logical or is usually understood as a choice in the standard deontic logic. This interpretation may produce the free choice permission paradoxes like "if it is permitted that one mails a letter or burns it, then one is allowed to both mail a letter and burn it". In FCL, we can add two rules to avoid these paradoxes: 1) if one mails a letter, then one is prohibited to burn it; 2) if one burns a letter, then one is prohibited to mail it. It is possible, however, for a contract state to be produced that contains contradictions, but this would be caused by a flaw in the contract, not a flaw in the conceptual framework.

8.2 An American Call Option

We formalize here Example 1.1.1, an American Call Option, as a contract $\mathbf{C}$ of FCL. Let $T = (\mathcal{L}, \Gamma)$ be an extension of T_0, the base theory of FCL, such that:

- $\mathcal{L} = (\mathcal{B}, \mathcal{C}, \tau)$ where:
  - $\mathcal{B} = \mathcal{B}_0$.
  - $\mathcal{C}$ includes the members of $\mathcal{C}_0$ and the constants given in the left column of Table 8.1.
  - The definitions of τ on the new constants are given in the right column of Table 8.1.
- $\Gamma = \Gamma_0$.

$\mathbf{C}$ has two parties: a seller and a buyer. The unit of time is one day. Let the offer time t_0 of the contract, the time the seller offered the contract to the buyer, be some day before June 30, 2015. $\mathbf{C}$ is defined as the pair $(t_0, \{\mathbf{D}_1, \mathbf{D}_2, \mathbf{R}_1\})$ where:

Table 8.1: Constants for an American Call Option

CONSTANT c	TYPE $\tau(c)$
seller, buyer	Party
"Buy Option"	String
"Exercise Option"	String
"Transfer Stock"	String

$\mathbf{D}_1 : t_{\text{buy}} = 0$ (June 30, 2015).

$\mathbf{D}_2 : t_{\text{expire}} = 170$ (December 17, 2015).

$\mathbf{R}_1$ is defined below.

$\mathbf{C}$ is constructed from two rules $\mathbf{R_1}$ and $\mathbf{R_2}$:

(1) *Rule for Buying the Option:*

$\mathbf{R}_1 = \varphi_1 \mapsto \{\mathbf{R}_2\}$ where:

$\varphi_1 = \mathsf{obs\text{-}event}(t_{\text{buy}}, e_1)$.

$e_1 = (\text{"Buy Option"}, \mathsf{transfer}(\$5), \{\mathsf{buyer}\}, \{\mathsf{seller}\})$.

$\mathbf{R}_2$ is defined below.

(2) *Rule for Exercising the Option:*

$\mathbf{R}_2 = \varphi_2 \mapsto \{\mathbf{D}_3, \mathbf{A}\}$ where:

$\varphi_2 = \mathsf{obs\text{-}event}(X_{\text{time}}, e_2) \wedge X_{\text{time}} \in [t_{\text{buy}}, t_{\text{expire}}]$.

$e_2 = (\text{"Exercise Option"}, \mathsf{transfer}(\$80), \{\mathsf{buyer}\}, \{\mathsf{seller}\})$.

$\mathbf{D}_3 : t_{\text{exercise}} = X_{\text{time}}$.

$\mathbf{A} = \mathbb{O}(e_3, [t_{\text{exercise}}, t_{\text{exercise}} + 30])$.

$e_3 = (\text{"Transfer Stock"}, \mathsf{deliver}(\mathsf{stock}), \{\mathsf{seller}\}, \{\mathsf{buyer}\})$.

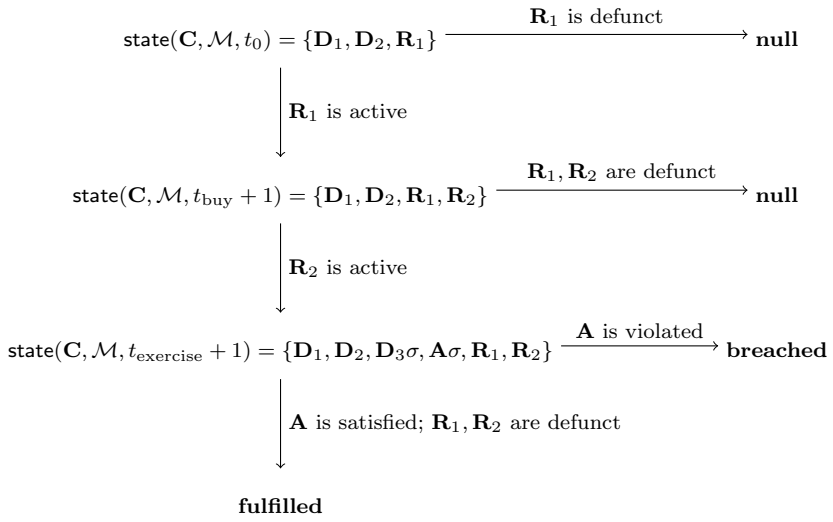

Figure 8.1: Execution of the American Call Option $\mathbf{C}$

t_{buy}, t_{expire}, and t_{exercise} are new constants of type Time. $[t_{\text{buy}}, t_{\text{expire}}]$ is the interval representing the set of times $\{t_{\text{buy}}, t_{\text{buy}} + 1, \ldots, t_{\text{expire}} - 1, t_{\text{expire}}\}$. $[t_{\text{exercise}}, t_{\text{exercise}}+30]$ is the interval representing the set of times $\{t_{\text{exercise}}, t_{\text{exercise}} + 1, \ldots, t_{\text{exercise}} + 30\}$.

e_1, e_2, and e_3 denote expressions of type Event. Each of these three events are actions by one of the two parties. "Buy Option", "Exercise Option", and "Transfer Stock" are constants of type String. They are the labels of the three events e_1, e_2, and e_3, respectively. The three events are tied to the contract. For example, a more exact name for "Buy Option" would be "Buy Option Described by Contract C". We assume that each of the three events can happen as most once. φ_1 asserts the option is bought on June 30, 2015, and φ_2 asserts the option is exercised at a time after the option is bought and before the option expires.

The *state* of $\mathbf{C}$ in a model $\mathcal{M}$ at time $t \geq t_0$, written as $\mathsf{state}(\mathbf{C}, \mathcal{M}, t)$, evolves over time as indicated in Figure 8.1. How the state of $\mathbf{C}$ evolves depends on the observables specified by $\mathcal{M}$. In the figure, let u be the time that R_2 becomes active, i.e, when the buyer exercises the option. Let σ be the substitution that maps X_{time}

to u. Applying σ has the effect of replacing X_{time} with u, whose value is the time u. $\mathbf{D_3}\sigma$ is thus the equation $t_{\text{exercise}} = u$, but $\mathbf{A}\sigma$ is $\mathbf{A}$ since X_{time} does not occur in $\mathbf{A}$.

8.3 A Sale of Goods Contract

We formalize here Example 3.2.1, the sale of a printer contract, as a contract $\mathbf{C}$ of FCL. Let $T = (\mathcal{L}, \Gamma)$ be an extension of T_0, the base theory of FCL, such that:

- $\mathcal{L} = (\mathcal{B}, \mathcal{C}, \tau)$ where:
  - $\mathcal{B} = \mathcal{B}_0$.
  - $\mathcal{C}$ includes the members of $\mathcal{C}_0$ and the constants given in the left column of Table 8.2.
  - The definitions of τ on the new constants are given in the right column of Table 8.2.
- Γ includes Γ_0 and the two sentences given below.
  (1) $\forall t, t' : \mathsf{Time},\ e : \mathsf{Event}\ .\ (\exists\, u : \mathsf{Time}\ .\ u \leq t \wedge u \leq t' \wedge \mathsf{obs\text{-}event}(u, e)) \iff \mathsf{obs\text{-}event\text{-}before}(t, t', e)$.
  (2) $\forall t : \mathsf{Time},\ \mathcal{T} : \mathsf{set}[\mathsf{Time}],\ e : \mathsf{Event}\ .\ (\exists\, u : \mathsf{Time}\ .\ u \leq t \wedge u \in \mathcal{T} \wedge \mathsf{obs\text{-}event}(u, e)) \iff \mathsf{obs\text{-}event\text{-}during}(t, \mathcal{T}, e)$.

$\mathbf{C}$ has two parties: a seller and a buyer. The unit of time is one day. Let the offer time t_0 of the contract, the time the seller offered the contract to the buyer, be some day before the buyer make an order. $\mathbf{C}$ is defined as the pair $(t_0, \{\mathbf{R}_1\})$ where $\mathbf{R}_1$ is defined below. $\mathbf{C}$ is constructed from the following nine rules:

Table 8.2: Constants for a Sale of Goods Contract

CONSTANT c	TYPE $\tau(c)$
seller, buyer	Party
"Order Printer"	String
"Deliver Printer"	String
"Return Printer"	String
"Return Payment"	String
"Pay Seller"	String
obs-event-before	$(\mathsf{Time} \times \mathsf{Time} \times \mathsf{Event} \to \mathsf{Bool})$
obs-event-during	$(\mathsf{Time} \times \mathsf{set}[\mathsf{Time}] \times \mathsf{Event} \to \mathsf{Bool})$
%	$((\mathsf{Int} \times \mathsf{Int}) \to \mathsf{Int})$

(1) *Rule for Ordering a Printer:*

$\mathbf{R}_1 = \varphi_1 \mapsto \{\mathbf{D}_1, \mathbf{A}_1, \mathbf{R}_2\}$.

$\varphi_1 = \mathsf{obs\text{-}event}(X_{\text{time}}, e_1) \wedge t_{\text{offer}} \leq X_{\text{time}}$.

$e_1 = (\text{"Order Printer"}, \mathsf{deliver}(\mathsf{order}), \{\mathsf{buyer}\}, \{\mathsf{seller}\})$.

$\mathbf{D}_1 : t_{\text{order}} = X_{\text{time}}$.

$\mathbf{A}_1 = \mathbb{O}(e_2, [t_{\text{order}}, t_{\text{order}} + 22])$.

$e_2 = (\text{"Deliver Printer"}, \mathsf{deliver}(\mathsf{printer}), \{\mathsf{seller}\}, \{\mathsf{buyer}\})$.

$\mathbf{R}_2$ is defined below.

(2) *Rule for Delivering the Printer:*

$\mathbf{R}_2 = \varphi_2 \mapsto \{\mathbf{D}_2, \mathbf{D}_3, \mathbf{R}_3, \mathbf{R}_4, \mathbf{R}_5, \mathbf{R}_6\}$.

$\varphi_2 = \mathsf{obs\text{-}event}(X_{\text{time}}, e_2) \wedge X_{\text{time}} \in [t_{\text{order}}, t_{\text{order}} + 22]$.

$\mathbf{D}_2 : t_{\text{deliver}} = X_{\text{time}}$.

$\mathbf{D}_3 : \mathsf{total\text{-}paid}(X_{\text{time}}) = 0$.

$\mathbf{R}_3, \mathbf{R}_4, \mathbf{R}_5$ and $\mathbf{R}_6$ are defined below.

(3) *Rule for Returning the Printer:*

$\mathbf{R_3} = \varphi_3 \mapsto \{\mathbf{D_4}, \mathbf{A_2}\}$.

$\varphi_3 = \textsf{obs-event}(X_{\text{time}}, e_3) \wedge X_{\text{time}} \in [t_{\text{deliver}}, t_{\text{deliver}} + 14]$.

$e_3 = (\text{“Return Printer”}, \textsf{deliver}(\textsf{printer}), \{\textsf{buyer}\}, \{\textsf{seller}\})$.

$\mathbf{D_4} : t_{\text{return}} = X_{\text{time}}$.

$\mathbf{A_2} = \mathbb{O}(e_4(\textsf{total-paid}(X_{\text{time}})), [t_{\text{return}}, t_{\text{return}} + 7])$.

$e_4 = \lambda X_{\text{total}}.(\text{“Return Payment”}, \textsf{transfer}(X_{\text{total}}), \{\textsf{seller}\}, \{\textsf{buyer}\})$.

(4) *Rules for Recording the Payment:*

$\mathbf{R_4} = \varphi_4 \wedge \neg\varphi_5 \mapsto \{\mathbf{D_5}\}$.

$\mathbf{R_5} = \neg\varphi_4 \wedge \neg\varphi_5 \mapsto \{\mathbf{D_6}\}$.

$\varphi_4 = \textsf{obs-event}(X_{\text{time}}, e_5(X_{\text{payment}})) \wedge t_{\text{deliver}} \leq X_{\text{time}}$.

$e_5 = \lambda X_{\text{payment}}.(\text{“Pay Seller”}, \textsf{transfer}(X_{\text{payment}}), \{\textsf{buyer}\}, \{\textsf{seller}\})$.

$\varphi_5 = \textsf{obs-event-before}(X_{\text{time}}, X_{\text{time}}, e_3)$.

$\mathbf{D_5} : \textsf{total-paid}(X_{\text{time}}) = \textsf{total-paid}(X_{\text{time}} - 1) + X_{\text{payment}}$.

$\mathbf{D_6} : \textsf{total-paid}(X_{\text{time}}) = \textsf{total-paid}(X_{\text{time}} - 1)$.

(5) *Rules for Making Payments:*

$\mathbf{R}_6 = \neg\varphi_6 \mapsto \{\mathbf{A}_3, \mathbf{R}_7\}$.

$\varphi_6 = \textsf{obs-event-before}(X_{\text{time}}, t_{\text{deliver}} + 1, e_3)$.

$\mathbf{A}_3 = \mathbb{O}(e_5(200 * 0.5), [t_{\text{deliver}}, t_{\text{deliver}} + 1])$.

$\mathbf{R}_7 = \varphi_7 \wedge \neg\varphi_5 \mapsto \{\mathbf{D}_7, \mathbf{R}_8\}$.

$\varphi_7 = \textsf{obs-event}(X_{\text{time}}, e_5(200 * 0.5)) \wedge X_{\text{time}} \in [t_{\text{deliver}}, t_{\text{deliver}} + 1]$.

$\mathbf{D}_7 : t_{\text{first}} = X_{\text{time}}$.

$\mathbf{R}_8 = \neg\varphi_8 \wedge \neg\varphi_5 \mapsto \{\mathbf{R_9}\}$.

$\varphi_8 = \mathsf{obs\text{-}event\text{-}during}(X_{\text{time}}, [t_{\text{first}} + 1, t_{\text{deliver}} + 14], e_5(200 * 0.5))$.

$\mathbf{R}_9$ is defined below.

(6) *Rules for Paying Fine for a Late Payment:*

$\mathbf{R}_9 = \neg\varphi_9 \mapsto \{\mathbf{A}_4, \mathbf{A}_5\}$.

$\varphi_9 = \mathsf{obs\text{-}event\text{-}during}(X_{\text{time}}, [t_{\text{deliver}} + 15, t_{\text{deliver}} + 30], e_5(200 * 0.5))$.

$\mathbf{A}_4 = \mathbb{O}(e_5(200 * 0.5), [t_{\text{deliver}} + 31, t_{\text{deliver}} + 44])$.

$\mathbf{A}_5 = \mathbb{O}(e_5(0.1 * 200 * 0.5), [t_{\text{deliver}} + 31, t_{\text{deliver}} + 44])$.

t_{order}, t_{deliver}, t_{return}, and t_{first} are new constants of type Time. Each of the five events e_1, e_2, e_3, e_4, and e_5 are actions by one of the two parties. We assume that the events e_1, e_2, e_3, e_4 can happen at most once and the "Pay Seller" event e_5 can happen at most twice.

$\mathsf{total\text{-}paid}(t)$ represents the total amount that the buyer has been observed to have paid the seller at time t. When rule $\mathbf{R_4}$ is active, $\mathbf{D_5}$ is generated. $\mathbf{D}_5$ is used to add a payment to the total amount paid at the previous time point. $\mathbf{D_5}$ and $\mathbf{D_6}$ work together to record the happenings of the "Pay Seller" event e_5 in the timeline. $\mathsf{obs\text{-}event\text{-}before}(t, t', e)$, a constant of type $\mathsf{Time} \times \mathsf{Time} \times \mathsf{Event} \to \mathsf{Bool}$, represents the observation that at time t the event e occurred on or before time t'.

We identify that the buyer has the following options to choose from after he has accepted the printer and made the first payment:

(1) Buyer makes a return within 14 days after the delivery is made.

(2) Buyer makes the second payment within 14 days after the delivery made.

(3) Buyer makes the second payment between 15 to 30 days after the delivery made.

(4) Buyer makes the second payment with an additional fine between 31 to 44 days after the delivery made.

$\mathbf{R_8}$ and $\mathbf{R_9}$ work together as a reparation if the second payment is not made on time. Within 14 days the buyer has the first and second options to choose from. $\mathbf{R_8}$, says if the first two options have not been chosen, then between 15 to 44 days the buyer is obligated to make the second payment. If it is paid late, which means $\mathbf{R_9}$ is active, then an additional fine must be paid.

The *state* of C in a model $\mathcal{M}$ at time $t \geq t_{\text{offer}}$, written as $\mathsf{state}(C, \mathcal{M}, t)$, evolves over time as indicated in Figure 8.2.

8.4 A Lease Contract

We formalize here Example 3.2.2, a lease contract, as a contract $\mathbf{C}$ of FCL. Let $T = (\mathcal{L}, \Gamma)$ be an extension of T_0, the base theory of FCL, such that:

- $\mathcal{L} = (\mathcal{B}, \mathcal{C}, \tau)$ where:
  - $\mathcal{B} = \mathcal{B}_0 \cup \{\mathbb{Q}, \mathsf{Notice}, \mathsf{Service}\}$. $\mathbb{Q}$ denotes a type of rational numbers. Notice and $\mathsf{Service}$ are unspecified types.
  - $\mathcal{C}$ includes the members of $\mathcal{C}_0$ and the constants given in the left column of Table 8.3.
  - The definitions of τ on the new constants are given in the right column of Table 8.3.

Figure 8.2: Execution of the Sale of a Printer contract **C**

Table 8.3: Constants for a Lease Contact

CONSTANT c	TYPE $\tau(c)$
landlord, tenant	Party
"Contract Signed"	String
"Tenant End Lease"	String
"Landlord End Lease"	String
"Provide Apartment"	String
"Pay Rent"	String
obs-cpi	(Time $\to \mathbb{Q}$)
obs-event-before	(Time $\times$ Time $\times$ Event $\to$ Bool)
provide	(Service $\to$ Act)
send	(Notice $\to$ Act)
max	($\mathbb{Q} \times \mathbb{Q} \to \mathbb{Q}$)

- Γ includes Γ_0 and the two sentences given below.

  (1) $\forall\, t, t' : \mathsf{Time},\ e : \mathsf{Event}\ .\ (\exists\, u : \mathsf{Time}\ .\ u \leq t \wedge u \leq t' \wedge \mathsf{obs\text{-}event}(u, e)) \iff \mathsf{obs\text{-}event\text{-}before}(t, t', e)$.

  (2) $\forall\, x, y : \mathbb{Q}\ .\ \mathsf{max}(x, y) = \mathsf{if}(x \leq y, y, x)$.

C has two parties: a landlord and a tenant. The unit of time is one month. A month m is divided into days so that $m.n$ is the nth day in month m. Let the offer time t_0 of the contract, the time the landlord offered the contract to the tenant, be some day before January 01, 2016. **C** is defined as the pair $(t_0, \{\mathbf{D}_1, \mathbf{D}_2, \mathbf{R}_1\})$ where:

$\mathbf{D}_1 : t_{\text{start}} = 1.1$ (2016-01-01).

$\mathbf{D}_2 : t_{\text{expire}} = 6.30$ (2016-06-30).

$\mathbf{R}_1$ is defined below.

C is constructed from the following seven rules:

(1) *Rule for Signing the Contract:*

$\mathbf{R}_1 = \varphi_1 \mapsto \{\mathbf{R}_2, \mathbf{R}_3, \mathbf{R}_4, \mathbf{R}_5, \mathbf{R}_6\}$ where

$\varphi_1 = \mathsf{obs\text{-}event}(X_{\text{time}}, e_1) \wedge t_{\text{offer}} \leq X_{\text{time}} \leq t_{\text{start}}$.

$e_1 = (\text{“Contract Signed”}, \mathsf{deliver}(\mathsf{signature}), \{\mathsf{tenant}\}, \{\mathsf{landlord}\})$.

$\mathbf{R}_2, \mathbf{R}_3, \mathbf{R}_4, \mathbf{R}_5$ and $\mathbf{R_6}$ are defined below.

(2) *Rule for Notifying the Termination of the Lease:*

$\mathbf{R_2} = \varphi_2 \wedge \varphi_3 \mapsto \{\mathbf{D}_3, \mathbf{D}_4, \mathbf{R}_7\}$.

$\varphi_2 = t_{\text{start}} \leq X_{\text{time}}$.

$\varphi_3 = \mathsf{obs\text{-}event}(X_{\text{time}}, e_2(X_{\text{month}})) \vee \mathsf{obs\text{-}event}(X_{\text{time}}, e_3(X_{\text{month}}))$.

$e_2 = \lambda X_{\text{month}}.(\text{“Tenant End Lease”}, \mathsf{send}(X_{\text{month}}), \{\mathsf{tenant}\}, \{\mathsf{landlord}\})$.

$e_3 = \lambda X_{\text{month}}.(\text{“Landlord End Lease”}, \mathsf{send}(X_{\text{month}}), \{\mathsf{landlord}\}, \{\mathsf{tenant}\})$.

$\mathbf{D}_3 : t_{\text{notice}} = X_{\text{time}}$.

$\mathbf{D_4} : \mathsf{termination\text{-}month}(X_{\text{time}}) = X_{\text{month}}$.

(3) *Rule for Paying Rent before Notifying the Termination of the Lease:*

$\mathbf{R_3} = \varphi_2 \wedge (\varphi_4 \vee \varphi_5) \mapsto \{\mathbf{A}_1, \mathbf{A}_2\}$.

$\varphi_4 = \neg(\mathsf{obs\text{-}event\text{-}before}(X_{\text{time}}, X_{\text{time}}, e_2(X_{\text{month}}))$
$\vee \mathsf{obs\text{-}event\text{-}before}(X_{\text{time}}, X_{\text{time}}, e_3(X_{\text{month}})))$.

$\varphi_5 = (\mathsf{obs\text{-}event\text{-}before}(X_{\text{time}}, X_{\text{time}}, e_2(X_{\text{month}}))$
$\vee \mathsf{obs\text{-}event\text{-}before}(X_{\text{time}}, X_{\text{time}}, e_3(X_{\text{month}})))$
$\wedge \neg(1 \leq X_{\text{month}} \wedge X_{\text{time}} \leq t_{\text{notice}} + X_{\text{month}})$.

$\mathbf{A_1} = \mathbb{O}(e_4, [X_{\text{time}}, X_{\text{time}}])$.

$e_4 = (\text{“Provide Apartment”}, \mathsf{provide}(\mathsf{appartment}), \{\mathsf{landlord}\}, \{\mathsf{tenant}\})$.

$\mathbf{A_2} = \mathbb{O}(e_5(\textsf{rent-required}(X_{\text{time}})), [X_{\text{time}}.1, X_{\text{time}}.7])$

$e_5 = \lambda X_{\text{rent}}.(\text{“Pay Rent”}, \textsf{transfer}(X_{\text{rent}}), \{\textsf{tenant}\}, \{\textsf{landlord}\})$.

(4) *Rules for Annually Changing Rent:*

$\mathbf{R_4} = \varphi_6 \mapsto \{\mathbf{D_5}\}$

$\mathbf{R_5} = \varphi_7 \wedge \varphi_8 \mapsto \{\mathbf{D_6}\}$.

$\mathbf{R_6} = \varphi_7 \wedge \neg\varphi_8 \mapsto \{\mathbf{D_7}\}$.

$\varphi_6 = (X_{\text{time}} = t_{\text{start}})$.

$\mathbf{D_5} : \textsf{rent-required}(X_{\text{time}}) = 1000$.

$\varphi_7 = \textsf{obs-event}(X_{\text{time}}, e_4) \wedge t_{\text{start}} + 1 \leq X_{\text{time}}$.

$\varphi_8 = (X_{\text{time}} \% 12 = 1)$.

$\mathbf{D_6} : \textsf{rent-required}(X_{\text{time}}) = \textsf{rent-required}(X_{\text{time}} - 1) * \textsf{obs-cpi}(X_{\text{time}})$.

$\mathbf{D_7} : \textsf{rent-required}(X_{\text{time}}) = \textsf{rent-required}(X_{\text{time}} - 1)$.

(5) *Rule for Paying Rent after Notifying the Termination of the Lease:*

$\mathbf{R_7} = \varphi_9 \mapsto \{\mathbf{A_1}, \mathbf{A_2}\}$.

$\varphi_9 = \textsf{max}(t_{\text{expire}}, t_{\text{notice}}) \leq X_{\text{time}} \wedge X_{\text{time}} \leq t_{\text{notice}} + \textsf{termination-notice}(t_{\text{notice}})$.

t_{start}, t_{expire}, and t_{notice} are new constants of type Time. Each of the four events e_1, e_2, e_3, and e_4 are actions by one of the two parties.

rent-required(t) represents the rent that the tenant has to pay to the landlord at time t. When rule $\mathbf{R}_4$ is active, $\mathbf{D}_5$ is generated. $\mathbf{D}_5$ is used to record the amount of rent required for the first month. $\mathbf{D}_6$ and $\mathbf{D}_7$ work together to calculate the required amount of rent based on the observation of the CPI changes.

send(X_{month}) represents the length of time the tenant will stay in the apartment. For example, if either the tenant or landlord sends to the other party a notice says the tenant will move out of the apartment after 2 months, then $X_{\text{month}} = 2$.

Chapter 9

A Reasoning System

In Chapter 7 we have explained how a rule can change the state of a contract over time. We think the state of a contract evolves over time like the state of a program evolves over time. A contract is fulfilled when all the agreements in its state are satisfied and all the rules in its state are no longer applicable. A contract is breached when some agreement in its state is violated.

This chapter presents a reasoning system for FCL that is an extension of a proof system for STT. The reasoning system of FCL can reason about whether a contract is fulfilled or breached over time. Section 9.1 gives the definitions that are needed to understand the rest of this chapter. Section 9.2 lists the additional judgments that our reasoning system should include. The rules of inference for the reasoning system are introduced in Section 9.3. An example of a simulation of a contract over time is given in Section 9.4.

9.1 Definitions

Let $T_0 = (\mathcal{L}_0, \Gamma_0)$ be the base theory of FCL, t be an expression of type Time, φ be a formula, Γ be a set of formulas, $\mathbf{A}$ be an agreement, $\mathbf{R}$ be a rule, and $\mathbf{C}$ be a contract. For a model $\mathcal{M}$ of FCL, $\mathcal{M} \models \varphi$ means $V^{\mathcal{M}}(\varphi) = \mathsf{T}$ and $\mathcal{M} \models \Gamma$ means $\mathcal{M} \models \psi$ for all $\psi \in \Gamma$.

9.2 Judgments

A reasoning system for STT has a judgment of the form $\Gamma \vdash \varphi$ that asserts φ logically follows from Γ, i.e., $\mathcal{M} \models \Gamma$ implies $\mathcal{M} \models \varphi$ for all models $\mathcal{M}$ of FCL. Since constant definitions, agreements, and rules are not expressions of STT, we need the following additional judgments in a reasoning system for FCL:

(1) $\Gamma \vdash_{\mathbf{C},t} \varphi$ asserts that φ logically follows from Γ and $\mathbf{C}$ at t, i.e., $\mathcal{M} \vDash \Gamma$ implies $\mathcal{M} \vDash \varphi$ for all models $\mathcal{M}$ of $\mathbf{C}$ at t.

(2) $\mathsf{Agreement}[\Gamma, \mathbf{A}, \mathbf{C}, t]$ asserts that $\mathbf{A}$ is in the state of $\mathbf{C}$ at t with respect to Γ, i.e., $\mathcal{M} \vDash \Gamma$ implies $\mathbf{A} \in \mathsf{state}(\mathbf{C}, \mathcal{M}, t)$ for all models $\mathcal{M}$ of FCL.

(3) $\mathsf{Rule}[\Gamma, \mathbf{R}, \mathbf{C}, t]$ asserts that $\mathbf{R}$ is in the state of $\mathbf{C}$ at t with respect to Γ, i.e., $\mathcal{M} \vDash \Gamma$ implies $\mathbf{R} \in \mathsf{state}(\mathbf{C}, \mathcal{M}, t)$ for all models $\mathcal{M}$ of FCL.

(4) $\mathsf{Satisfied}[\Gamma, \mathbf{A}, \mathbf{C}, t]$ asserts that $\mathbf{A}$ is satisfied at t with respect to Γ, i.e., $\mathcal{M} \vDash \Gamma$ implies that $\mathbf{A}$ is satisfied in $\mathcal{M}$ at t for all models $\mathcal{M}$ of $\mathbf{C}$ at t.

(5) $\mathsf{Violated}[\Gamma, \mathbf{A}, \mathbf{C}, t]$ asserts that $\mathbf{A}$ is violated at t with respect to Γ, i.e., $\mathcal{M} \vDash \Gamma$ implies $\mathbf{A}$ is violated in $\mathcal{M}$ at t for all models $\mathcal{M}$ of $\mathbf{C}$ at t.

(6) $\mathsf{Defunct}[\Gamma, \mathbf{R}, \mathbf{C}, t]$ asserts that $\mathbf{R}$ is defunct at t with respect to Γ, i.e., $\mathcal{M} \vDash \Gamma$ implies $\mathbf{R}$ is defunct in $\mathcal{M}$ at t for all models $\mathcal{M}$ of $\mathbf{C}$ at t.

(7) $\mathsf{Fulfilled}[\Gamma, \mathbf{C}, t]$ asserts that $\mathbf{C}$ is fulfilled at t with respect to Γ, i.e., $\mathcal{M} \vDash \Gamma$ implies $\mathbf{C}$ is fulfilled in $\mathcal{M}$ at t for all models $\mathcal{M}$ of $\mathbf{C}$ at t,

(8) $\mathsf{Breached}[\Gamma, \mathbf{C}, t]$ asserts that $\mathbf{C}$ is breached at t with respect to Γ, i.e., $\mathcal{M} \vDash \Gamma$ implies $\mathbf{C}$ is breached in $\mathcal{M}$ at t for all models $\mathcal{M}$ of $\mathbf{C}$ at t

The role of Γ is to specify the models in which $\mathbf{C}$ will be considered.

9.3 Rules of Inference

The reasoning system of FCL has the 16 rules of inference below including the usual rules of inference for STT. The first rule of inference shows $\Gamma \vdash_{\mathbf{C},t} \varphi$ extends $\Gamma \vdash \varphi$.

Inference Rule 1.

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash_{\mathbf{C},t} \varphi}.$$

Inference Rule 1 says that if φ follows from Γ, then φ follows from Γ and $\mathbf{C}$ at t for any contract $\mathbf{C}$ and time t.

Let t_0 be an expression of type Time representing the time that a contract is offered. The following four rules of inference show how the initial state of a contract is established:

Inference Rule 2.

$$\frac{\mathbf{C} = (t_0, \mathcal{Q})}{\Gamma \vdash_{\mathbf{C},t_0} t_{\text{offer}} = t_0}$$

Inference Rule 3.

$$\frac{\mathbf{C} = (t_0, \mathcal{Q}) \qquad \mathbf{D} \in \mathcal{Q}}{\Gamma \vdash_{\mathbf{C},t_0} \mathbf{D}}$$

Inference Rule 4.

$$\frac{\mathbf{C} = (t_0, \mathcal{Q}) \qquad \mathbf{A} \in \mathcal{Q}}{\mathsf{Agreement}[\Gamma, \mathbf{A}, \mathbf{C}, t_0]}$$

Inference Rule 5.

$$\frac{\mathbf{C} = (t_0, \mathcal{Q}) \qquad \mathbf{R} \in \mathcal{Q}}{\mathsf{Rule}[\Gamma, \mathbf{R}, \mathbf{C}, t_0]}$$

In FCL, once a constant definition, an agreement, or a rule has been added into the state of a contract, it will be always in the state until the contract is terminated. The following three rules of inference establish this property:

Inference Rule 6.

$$\frac{\Gamma \vdash_{\mathbf{C},t} \mathbf{D}}{\Gamma \vdash_{\mathbf{C},t+1} \mathbf{D}}$$

Inference Rule 7.

$$\frac{\mathsf{Agreement}[\Gamma, \mathbf{A}, \mathbf{C}, t]}{\mathsf{Agreement}[\Gamma, \mathbf{A}, \mathbf{C}, t+1]}$$

Inference Rule 8.

$$\frac{\mathsf{Rule}[\Gamma, \mathbf{A}, \mathbf{C}, t]}{\mathsf{Rule}[\Gamma, \mathbf{A}, \mathbf{C}, t+1]}$$

Let $\mathbf{R} = \varphi \mapsto \{\mathbf{D}_1, \ldots, \mathbf{D}_k, \mathbf{A}_1, \ldots, \mathbf{A}_m, \mathbf{R}_1, \ldots, \mathbf{R}_n\}$ be a rule where $\mathbf{D}_1, \ldots, \mathbf{D}_k$ are constant definitions, $\mathbf{A}_1, \ldots, \mathbf{A}_m$ are agreements, $\mathbf{R}_1, \ldots, \mathbf{R}_n$ are rules, and $\sigma \in \mathsf{sub}(\varphi, t)$. The following three rules of inference show how a rule changes the state of a contract:

Inference Rule 9.

$$\frac{\mathsf{Rule}[\Gamma, \mathbf{R}, \mathbf{C}, t], \Gamma \vdash_{\mathbf{C},t} \varphi\sigma}{\Gamma \vdash_{\mathbf{C},t+1} \mathbf{D}_1\sigma, \ldots, \Gamma \vdash_{\mathbf{C},t+1} \mathbf{D}_k\sigma}$$

Inference Rule 10.

$$\frac{\mathsf{Rule}[\Gamma, \mathbf{R}, \mathbf{C}, t], \Gamma \vdash_{\mathbf{C},t} \varphi\sigma}{\mathsf{Agreement}[\Gamma, \mathbf{A}_1\sigma, \mathbf{C}, t+1], \ldots, \mathsf{Agreement}[\Gamma, \mathbf{A}_m\sigma, \mathbf{C}, t+1]}$$

Inference Rule 11.

$$\frac{\mathsf{Rule}[\Gamma, \mathbf{R}, \mathbf{C}, t], \Gamma \vdash_{\mathbf{C},t} \varphi\sigma}{\mathsf{Rule}[\Gamma, \mathbf{R}_1\sigma', \mathbf{C}, t+1], \ldots, \mathsf{Rule}[\Gamma, \mathbf{R}_n\sigma', \mathbf{C}, t+1]}$$

where $\sigma' = \sigma[X_{\text{time}} \mapsto X_{\text{time}}]$.

An *agreement* of FCL is a formula of FCL that asserts that something will be true at a specified time. There is a rule of inference for satisfied agreements and a rule of inference for violated agreements:

Inference Rule 12 (Satisfied Agreement)**.**

$$\frac{\mathsf{Agreement}[\Gamma, \mathbf{A}, \mathbf{C}, t], \Gamma \vdash_{\mathbf{C},t} \mathbf{A}}{\mathsf{Satisfied}[\Gamma, \mathbf{A}, \mathbf{C}, t]}.$$

Inference Rule 13 (Violated Agreement)**.**

$$\frac{\mathsf{Agreement}[\Gamma, \mathbf{A}, \mathbf{C}, t], \Gamma \vdash_{\mathbf{C},t} \mathbf{A}^*}{\mathsf{Violated}[\Gamma, \mathbf{A}, \mathbf{C}, t]}$$

where A^* *is the dual of* A *(See Section 6.4.2 for the definition).*

Let n be an integer with an expression of type Time representing n. There is a rule of inference for defunct rules:

Inference Rule 14 (Defunct Rule)**.**

$$\frac{\mathsf{Rule}[\Gamma, \mathbf{R}, \mathbf{C}, t], (\Gamma \vdash_{\mathbf{C},t} \neg\varphi\sigma \textit{ for all } n \geq 0 \textit{ and all } \sigma \textit{ with } \sigma(X_{\text{time}}) = t + n)}{\mathsf{Defunct}[\Gamma, \mathbf{R}, \mathbf{C}, t]}.$$

The following two rules of inference are for fulfilled and breached contract:

Inference Rule 15 (Fulfilled Contract)**.**

$$\mathsf{Defunct}[\Gamma, \mathbf{R}, \mathbf{C}, t] \textit{ for all rules } \mathbf{R} \textit{ such that } \mathsf{Rule}[\Gamma, \mathbf{R}, \mathbf{C}, t] \textit{ holds}$$

$$\frac{\mathsf{Satisfied}[\Gamma, \mathbf{A}, \mathbf{C}, t] \textit{ for all agreements } \mathbf{A} \textit{ such that } \mathsf{Agreement}[\Gamma, \mathbf{A}, \mathbf{C}, t] \textit{ holds}}{\mathsf{Fulfilled}[\Gamma, \mathbf{C}, t]}.$$

Inference Rule 16 (Breached Contract)**.**

$$\frac{\mathsf{Violated}[\Gamma, \mathbf{A}, \mathbf{C}, t] \textit{ for some agreement } \mathbf{A} \textit{ such that } \mathsf{Agreement}[\Gamma, \mathbf{A}, \mathbf{C}, t] \textit{ holds}}{\mathsf{Breached}[\Gamma, \mathbf{C}, t]}.$$

9.4 Example

Our reasoning system can be used to both prove statements about a contract and to simulate the unfolding of a contract over time. The latter is done by using Γ to specify the observations that are expected over the course of the contract. The reasoning system can be strengthened by introducing temporal operators that enable one to say, for example, that it follows from Γ that $\mathbf{C}$ will *eventually* be fulfilled.

We formalized Example 1.1.1 in FCL in Chapter 8. We now demonstrate how to simulate the unfolding of this American Call Option over time by using the reasoning system of FCL. Section 9.4.1 gives an example of a fulfilled option and Section 9.4.2 presents the simulation of a breached option. The simulations of this American Call Option correspond to its executions as shown in Figure 8.1 of Section 8.2.

9.4.1 Case 1: Contract is Fulfilled

The example contract $\mathbf{C}$ is defined as the pair $(t_0, \{\mathbf{D}_1, \mathbf{D}_2, \mathbf{R}_1\})$. Let $T_1 = (\mathcal{L}, \Gamma_1)$ be a theory of $\mathbf{C}$ and $\mathcal{M} = (\{D_\alpha : \alpha \in \Omega_\mathcal{L}\}, I, e)$ be a model of T_1. Let $\psi_1, \psi_2, \psi_3, \psi_4, \psi_5 \in \Gamma_1$ where:

ψ_1: $t_{\text{offer}} \leq t_{\text{buy}}$.

ψ_2: $\forall\, t : \mathsf{Time} \,.\, \mathsf{obs\text{-}event}(t, e_1) \iff t = t_{\text{buy}}$.

ψ_3: $\exists !\, t : \mathsf{Time} \,.\, \mathsf{obs\text{-}event}(t, e_2) \wedge t \in [t_{\text{buy}}, t_{\text{expire}}]$.

ψ_4: $\exists !\, t : \mathsf{Time} \,.\, \mathsf{obs\text{-}event}(t, e_3) \wedge t \in [t_{\text{exercise}}, t_{\text{exercise}} + 30]$.

ψ_5: $\forall\, t : \mathsf{Time} \,.\, t > t_{\text{exercise}} \implies \neg \mathsf{obs\text{-}event}(t, e_2)$.

Then the unfolding of this fulfilled option is simulated as shown below.

$\Gamma_1 \vdash_{\mathbf{C}, t_0} t_{\text{offer}} = t_0$ follows from $\mathbf{C}$ by Inference Rule 2. (9.1)

$\Gamma_1 \vdash_{\mathbf{C}, t_0} \mathbf{D}_1$ follows from $\mathbf{C}$ by Inference Rule 3. (9.2)

$\Gamma_1 \vdash_{\mathbf{C}, t_0} \mathbf{D}_2$ follows from $\mathbf{C}$ by Inference Rule 3. (9.3)

$\mathsf{Rule}[\Gamma_1, \mathbf{R}_1, \mathbf{C}, t_0]$ follows from $\mathbf{C}$ by Inference Rule 5. (9.4)

$\Gamma_1 \vdash \psi_1$ by STT logic and Inference Rule 1. (9.5)

$\Gamma_1 \vdash \psi_2$ by STT logic and Inference Rule 1. (9.6)

$\Gamma_1 \vdash_{\mathbf{C}, t_{\text{buy}}} \varphi_1$ follows from 9.1, 9.2, 9.5 and 9.6 by Inference Rule 1. (9.7)

$\mathsf{Rule}[\Gamma_1, \mathbf{R}_1, \mathbf{C}, t_{\text{buy}}]$ follows from 9.4 and 9.7 by Inference Rule 8. (9.8)

$\mathsf{Rule}[\Gamma_1, \mathbf{R}_2, \mathbf{C}, t_{\text{buy}} + 1]$ follows from 9.7 and 9.8 by Inference Rule 11. (9.9)

$\Gamma_1 \vdash \psi_3$ by STT logic and Inference Rule 1. (9.10)

$\Gamma_1 \vdash \mathsf{obs\text{-}event}(u_1, e_2) \wedge u_1 \in [t_{\text{buy}}, t_{\text{expire}}]$ follows from 9.3, 9.10 by Inference Rule 1. (9.11)

$\Gamma_1 \vdash_{\mathbf{C}, u_1} \varphi_2[X_{\text{time}} \mapsto u_1]$ follows from 9.11 by Inference Rule 1. (9.12)

$\mathsf{Rule}[\Gamma_1, \mathbf{R}_2, \mathbf{C}, u_1]$ follows from 9.9 and 9.11 by Inference Rule 8. (9.13)

$\Gamma_1 \vdash_{\mathbf{C}, u_1+1} \mathbf{D}_3[X_{\text{time}} \mapsto u_1]$ follows from 9.12 and 9.13 by Inference Rule 9. (9.14)

$\mathsf{Agreement}[\Gamma_1, \mathbf{A}, \mathbf{C}, u_1 + 1]$ follows from 9.12 and 9.13 by Inference Rule 10. (9.15)

$\Gamma_1 \vdash \psi_4$ by STT logic and Inference Rule 1. (9.16)

$\Gamma_1 \vdash \mathsf{obs\text{-}event}(u_2, e_3) \wedge u_2 \in [t_{\text{exercise}}, t_{\text{exercise}} + 30]$ follows from 9.16 by Inference Rule 1. (9.17)

$\mathsf{Agreement}[\Gamma_1, \mathbf{A}, \mathbf{C}, u_2]$ follows from 9.14, 9.15, and 9.17 by Inference Rule 7. (9.18)

$\Gamma_1 \vdash_{\mathbf{C}, u_2} \mathbf{A}$ follows from 9.17 and 9.18 by the definition of $\mathbf{A}$. (9.19)

$\mathsf{Satisfied}[\Gamma_1, \mathbf{A}, \mathbf{C}, u_2]$ follows from 9.18 and 9.19 by Inference Rule 12. (9.20)

$\mathsf{Rule}[\Gamma_1, \mathbf{R}_1, \mathbf{C}, u_2]$ follows from 9.4 and 9.17 by Inference Rule 8. (9.21)

$\mathsf{Defunct}[\Gamma_1, \mathbf{R}_1, \mathbf{C}, u_2]$ follows from 9.6 and 9.21 by Inference Rule 14. (9.22)

$\Gamma_1 \vdash \psi_5$ by STT logic and Inference Rule 1. (9.23)

$\mathsf{Rule}[\Gamma_1, \mathbf{R}_2, \mathbf{C}, u_2]$ follows from 9.9 and 9.17 by Inference Rule 8. (9.24)

$\mathsf{Defunct}[\Gamma_1, \mathbf{R}_2, \mathbf{C}, u_2]$ follows from 9.23 and 9.24 by Inference Rule 14. (9.25)

$\mathsf{Fulfilled}[\Gamma_1, \mathbf{C}, u_2]$ (9.26)

At time u_2 there are two rules (9.21 and 9.24) in the state of the contract $\mathbf{C}$ and the rules are defunct (9.22 and 9.25). And there is one agreement (9.18) which is satisfied (9.20) in the state of $\mathbf{C}$ at u_2. Thus 9.26 follows from 9.18, 9.20, 9.21, 9.22, 9.24, and 9.25 by Rule 15.

Note that the simulation of this example contract does not end up using all inference rules that are given in Section 9.3. Indeed, selected rules of inference will be used while we simulate the unfolding of different contracts. For example, Inference Rule 4 will be applied only when we simulate a contract that contains an initial agreement.

9.4.2 Case 2: Contract is Breached

Let $T_2 = (\mathcal{L}, \Gamma_2)$ be a theory of $\mathbf{C}$ and $\mathcal{M} = (\{D_\alpha : \alpha \in \Omega_\mathcal{L}\}, I, e)$ be a model of T_2. Let $\psi_1, \psi_2, \psi_3, \psi_6 \in \Gamma_2$ where:

ψ_6: $\forall\, t : \mathsf{Time}\ .\ t \in [t_{\text{exercise}}, t_{\text{exercise}} + 30] \implies \neg\mathsf{obs\text{-}event}(t, e_3)$.

One can think of a scenario that constitute a breach of this option: the option seller fails to deliver the specified stock to the option buyer on time. Then the unfolding of this breached option is simulated as shown below. Note that the steps 9.27—9.41 are the same as the steps 9.1—9.15 in Case 1.

$$\Gamma_2 \vdash_{\mathbf{C},t_0} t_{\text{offer}} = t_0 \tag{9.27}$$

$$\Gamma_2 \vdash_{\mathbf{C},t_0} \mathbf{D}_1 \tag{9.28}$$

$$\Gamma_2 \vdash_{\mathbf{C},t_0} \mathbf{D}_2 \tag{9.29}$$

$$\mathsf{Rule}[\Gamma_2, \mathbf{R}_1, \mathbf{C}, t_0] \tag{9.30}$$

$$\Gamma_1 \vdash \psi_1 \tag{9.31}$$

$$\Gamma_2 \vdash \psi_2 \tag{9.32}$$

$$\Gamma_2 \vdash_{\mathbf{C},t_{\text{buy}}} \varphi_1 \tag{9.33}$$

$$\mathsf{Rule}[\Gamma_2, \mathbf{R}_1, \mathbf{C}, t_{\text{buy}}] \tag{9.34}$$

$$\mathsf{Rule}[\Gamma_2, \mathbf{R}_2, \mathbf{C}, t_{\text{buy}} + 1] \tag{9.35}$$

$$\Gamma_2 \vdash \psi_3 \tag{9.36}$$

$$\Gamma_2 \vdash \mathsf{obs\text{-}event}(u_1, e_2) \wedge u_1 \in [t_{\text{buy}}, t_{\text{expire}}] \tag{9.37}$$

$$\Gamma_2 \vdash_{\mathbf{C},u_1} \varphi_2[X_{\text{time}} \mapsto u_1] \tag{9.38}$$

$$\mathsf{Rule}[\Gamma_2, \mathbf{R}_2, \mathbf{C}, u_1] \tag{9.39}$$

$$\Gamma_2 \vdash_{\mathbf{C},u_1+1} \mathbf{D}_3[X_{\text{time}} \mapsto u_1] \tag{9.40}$$

$\mathsf{Agreement}[\Gamma_2, \mathbf{A}, \mathbf{C}, u_1 + 1]$ (9.41)

$\Gamma_2 \vdash \psi_6$ by STT logic and Inference Rule 1. (9.42)

$\mathsf{Agreement}[\Gamma_2, \mathbf{A}, \mathbf{C}, t_{\text{exercise}} + 31]$ follows from 9.40 and 9.41 by Inference Rule 7. (9.43)

$\Gamma_2 \vdash_{\mathbf{C}, t_{\text{exercise}}+31} \mathbf{A}^*$ follows from 9.42 and 9.43 by the definition of $\mathbf{A}$. (9.44)

$\mathsf{Violated}[\Gamma_2, \mathbf{A}, \mathbf{C}, t_{\text{exercise}} + 31]$ follows from 9.43 and 9.44 by Inference Rule 13. (9.45)

$\mathsf{Breached}[\Gamma_2, \mathbf{C}, t_{\text{exercise}} + 31]$ follows from 9.43 and 9.45 by Inference Rule 16. (9.46)

Chapter 10

Conclusion and Future Work

The main goal of this thesis is to help people write contracts that have a clear meaning and can be readily analyzed and simplified. One way of achieving this goal is by using a formal language with a precise semantics to write contracts. Writing contracts in a formal language can help understand the meaning of contracts. Precision of the language and careful specification of the procedure of contracts can improve the communication between parties. In this thesis we have presented such a language named FCL. FCL is a formal language for writing general contracts that may contain temporally based conditions. In this chapter, we review the contributions of our research in Section 10.1, compare our approach with related works in Section 10.2 and 10.3, and discuss our future research directions in Section 10.4.

10.1 Highlights of Contributions

We presented the formal contract language FCL. We think of a contract in two ways. Syntactically, a contract is a written expression satisfying certain syntactic conditions. Semantically, a contract is a set of temporally condition-based promises between parties. Changes to the states of a FCL contract are triggered when the conditions expressed in it become true.

FCL contains the attributes listed below that enable the requirements presented in Chapter 3 to be fulfilled.

(1) FCL is formal and has a precise semantics. Since the underlying logic of FCL is a version of simple type theory, the semantics of contracts written in FCL is based on very well understood ideas.

(2) The parties of a contract in FCL are explicitly indicated in the contract, thus FCL is able to specify multiple parties of a contract including third parties.

(3) Times relevant to a contract are expressed explicitly.

(4) FCL can express agreements inspired by the obligations, prohibitions, and permissions of deontic logic.

(5) FCL can express conditions that depend on events and other observables, and include condition-based rules to introduce new constants and new agreements.

(6) FCL is able to express instantaneous as well as continuous actions in discrete time.

We clearly defined what a contract is and what it means to fulfill and breach a contract. We demonstrated the usefulness of FCL by applying FCL to various kinds of

contracts and successfully formalized them. We formalize an option contract, a sale of a printer contract, and a lease. We validated FCL by developing a reasoning system for FCL. In the next section, we will compare FCL with the previous proposed formal contract languages.

10.2 Comparison of the Approaches

Table 4.1 and 4.2 that are introduced in Section 4.1, and Table 10.1,10.2, 10.3, and 10.4 show a comparison of the related work and FCL.

Several techniques are employed in the literature for developing a precise formal language for specifying contracts. Most of the techniques, such as those given in [26, 28, 41], belong to the ECA-based scheme. GHM and G&M model contracts as sets of policies. AEHSS provide an action-trace based language [3] to model contracts. J&E's functional programming based language [46, 47] and BBE's cash-flow trace based approach [8] use the idea of observables to specify events. P&S introduce in [49, 50, 51] a dynamic, deontic action based language $\mathcal{CL}$ for expressing contracts. Rather than providing a logical language for contracts, the OASIS extends RuleML to provide a rule interchange language with formal features specific for the legal domain. This enables implementers to structure the contents of the legal texts in a machine-readable format by using the representation tools. We introduce in FCL the concept of a *rule* that is (in its simplest form) a conditional agreement that depends on certain observations. The use of observables to determine both the meaning of a contract and how the state of the contract evolves over time provides a basis for monitoring the dynamic aspects of a contract.

Not all the approaches can specify reparation clauses. LegalRuleML introduces a

Table 10.1: Comparison of our approach with other approaches

Perspective		LEE	PJ&E	G&M	GHM	AEHSS	BBE	P&S	D&S	WYNER	OASIS	FCL
1. Contract Expressiveness												
1	Time and object	√	√			√	√	√			√	√
2	Contract commitments			√				√	√	√	√	√
3	External third-parties	√				√	√					√
4	Nested rules										√	√
2. Contract Monitoring												
1	Actions Executions	√	√	√	√	√	√	√	√	√	√	√
2	Violations Detection	√			√	√		√	√	√	√	√
3	Blame assignment	(√)								√	√	√
3. Contract Enforcement												
1	Applicable laws										√	
2	Sanctions in the event of a breach of contract	√		√		√	√	√	√		√	√
4. Implementation												
1	Contract language	√	√				√			√	√	
2	A reasoning system						√					

Table 10.2: Comparison of our language with other approaches

Research	Domain	Methodology	Parties
Lee	General business contract.	1. This approach makes use of a Petri net notation to model contracts and then translate the notation into logic programming. 2. Consider only one world instead of possible world semantics.	Multiple (more than two) parties can be explicitly specified in a contract. By using the logic programming query engine, it is able to identify who have outstanding commitments.
D&S	Develop a language for the specification of the legal aspects of contracts.	Contracts are represented in event calculus.	Multiple parties can be explicit specified.
PJ&E	Develop a language for pricing financial contract valuation.	1. Define a set of constructors for contracts and demonstrate how to use them to describe standard financial contracts. 2. Formalize the compositional structure of financial contracts in functional programming language Haskel.	All contracts are two-party contract. The parties are implicitly specified.
G&M	Develop a language for business contract called BCL to express contract conditions for run time contract monitoring.	1. Similar to GHM's ECA approach, a contract consists a set of roles along with a set of policies. 2. An expressive event language for the specification of event-based behavior as part of policy expression. 3. The language is then mapped to a fragment of stand deontic logic (SDL) extended with contrary-to-duty (CTD) obligations. 4. Deontic modalities are restricted to out-to-do statements. 5. The language is XML-centric, exploiting relevant XML standards.	Multiple parties can be explicitly specified in a contract, but it is not able to identify who have outstanding commitments.
GHM	Develop a domain specific language (DSL), designed to support abstractions needed for the expressing of business contract.	Based on the ECA paradigm, contracts are modeled as sets of policies which specify that forbidden or obliged actions are triggered by events when certain conditions are met.	Multiple parties can be explicitly specified in a contract, it is not able to identify who have outstanding commitments.
AEHSS	Commercial contract.	1. Provide a compositional specification of contracts. 2. Provide a trace-based semantic model for contracts.	Two parties can be explicitly specified in a contract, but it is not able to identify who have outstanding commitments.
BBE	Develop a language for pricing financial contract valuation.	1. Provide a compositional specification of contracts. 2. Provide a trace-based semantic model for contracts.	Multiple parties can be explicitly specified, but it is not able to identify who have outstanding commitments.
P&S	Develop a language for the automation of contract management activity.	1. Introduce a logic for expressing contracts based on a combination of deontic, dynamic, and temporal logics. 2. Deontic modalities are restricted to out-to-do statements. 3. Present a trace semantics for contracts	Parties are implicitly specified.
Wyner	Develop a fulfillment and violation marker for general electronic contract.	Based on deontic logic to provide a rule based framework for the fulfillment and violation marker of contracts.	Multiple parties can be implicitly specified in a contract.
OASIS	Focuses on the creation of machine-readable forms of the content of legal texts, such as legislation, regulations, contracts, and case law, for different concrete Web applications.	Extends RuleML to provide a rule interchange language for the legal domain.	Multiple parties can be explicitly specified.
Ours	General contract.	Design a formal language for writing contracts.	Multiple parties can be explicitly specified.

Table 10.3: Comparison of our language with other approaches

	Time	Contract States	State Transitions
LEE	1. Adopt ideas from RU calculus, time are explicitly specified as calendar date which is an interval scale rather than a ratio scale. 2. Only consider one type of time unit, an integer multiple of days, ignoring other units such as months and years. 3. Temporal operators are introduced and thus able to specify reference time as deadlines.	Using T calculus to monitoring the transition system. The states are propositions.	The actions of parties are specified as attributes of transitions.
D&S	Based on the event calculus to specify the time relation.	Not provided.	Performance of dynamic actions trigger the transitions of contract states.
PJ&E	1. Time are explicitly specified as calendar date. 2. Reference absolute time based on the unit of days.	A compositional specification of contracts.	Observational values trigger the transitions of contact states.
G&M	Time are modeled as a set of integers with unit of days.	A set of policies.	Happening of complex events trigger the transitions of contract states.
GHM	Time line based on sequence of states.	A set of policies.	State change caused (only) by the execution of physical actions (and their indirect effect).
AEHSS	1. Time are modeled as a totally ordered set of integers. 2. It is able to monitoring the absolute and relative times.	A compositional specification of contracts. A base structure of contract is a tuple of sets of resources, agents and time.	The actions which transfer resource from one party to another at some specific time.
BBE	Time line based on sequence of states. The language operates with days as the smallest unit of time.	A compositional specification of contracts.	Observational values trigger the transitions of contact states.
P&S	Temporal operators are introduced to specify the relation of time.	Not provided.	Performance of dynamic actions trigger the transitions of contract states.
WYNER	Time are implicit specified.	Valued action specifications	1. Any boolean expression can update the contract history (actions performed and violations or fulfillments of agreements found). 2. Based on the rule function of contracts and the contract history to trigger the change of states.
OASIS	Time are explicitly specified as calendar date.	Rules are used to trigger the change of states. A rule of the work is considered as a binary relationship between the pre-conditions and the effect of the rule.	Performance of dynamic actions trigger the transitions of contract states.
FCL	Time are explicitly specified as a set of integers with chosen units.	The state of a contract is a set of constant definitions, agreements, and rules of the contract.	Any condition can trigger an update of the contract state.

Table 10.4: Comparison of our language with other approaches

	Contract Values	Contract Reparations	Contract Semantics
Lee	Not provided.	No means to repair a contract.	No formal semantics provided.
D&S	Violations of obligations.	No means to repair a contract.	No formal semantics provided.
PJ&E	Monetary value of contracts.	No means to repair a contract.	Only provide means to perform a valuation analysis of contracts.
G&M	Violations of obligations.	Contrary-to-duty (CTD) obligations are introduced to repair violated obligations.	A possible world semantics.
GHM	Not provided.	No means to repair a contract.	No formal semantics provided.
AEHSS	Fulfillment and breach of contracts.	CTD	Provide a trace-based semantics.
BBE	Monetary value of contracts.	No means to repair a contract.	Provide reduction semantics and a denotational cash-flow semantic.
P&S	Violations of obligations.	CTD	Provide a trace-based semantics.
Wyner	Fulfillment and breach of contracts.	Rule functions are used to repair a contract.	Utilize the deontic logic to reason about violations of obligations.
OASIS	Fulfillment and breach of contracts.	CTD	Utilize the defeasible deontic logic to reason about violations of obligations.
FCL	Fulfillment and breach of contracts.	Rules are used to repair a violation of an agreement.	Based on the simple type theory to provide semantics to reason about fulfillment and breach of contracts.

suborder list that is a list of deontic formulas to model penalties. Wyner introduces the concept of rule function for the contract reparation. We think the use of only contrary-to-duty obligations to recover a contract when it is breached is too limited. There is no provision provided for recovery from technical or business-related issues. In FCL, we interpret an agreement in a contract in terms of the deontic concepts of *obligation* and *prohibition*. These concepts are applied in expressions to actions that are executed by the parties of the contract. Thus, the concepts express what a party *ought to do* and or *ought not do*. FCL rules can also be used for reparational purposes when an agreement is violated (see subsection 8.1.2).

With the exception of approaches provided by AEHSS, BBE, P&S, and OASIS, all of the languages above are informal. The work of both AEHSS and P&S include a trace-based reduction semantics model for contracts. These two approaches provide a run-time monitoring of the fulfillment and breach of a contract since the state of a contract at a time is determined by the events that have happened. LegalRuleML utilizes the defeasible deontic logic to reason about violations of obligations. Both GHM's work and G&M's work lack a formal semantics and a reasoning system even though they provide a good framework for monitoring contracts. The semantics provided by J&E in is based on stochastic processes. J&E's approach provides the ability to perform compositional analysis of monetary values of contracts. This work can estimate the expected value of financial contracts. BBE's trace-based semantics allows the modification of a contract according to the passage of time and the values of observables. But since both of the approaches provided by J&E and BBE pay more attention to finding the monetary value of contracts, they consider the semantic meaning of a contract to be its cash-flow gain or loss, which is too limited for general

contracts from our point of view. We find this lack of work on formal semantics surprising since one of the main benefits of defining a contract language to be formal is to enable the language to have a precise, unambiguous semantics.

Although the languages of AEHSS and P&S provide a formal mathematical model for contracts with a formal semantics and are able to express some important features of contracts, they are not as expressive as FCL. For example, in the case where a contract is breached, the monitor should not only report a breach of contract, but also who among the contract parties is responsible (blame assignment). Except for the languages provided by BBE and OASIS, all the other contracts covered by these approaches, including the work of AEHSS and P&S, are two-party contracts in which the parties are implicit. These approaches are not able to determine who is to be blamed when a contract is breached. Our language provides explicit participants and thus provides the possibility of having contracts with both an unrestricted number of parties and with blame assignment. If an agreement is violated, the subject of the violated action is responsible for this violation.

Because time constraints are implicit in P&S's $\mathcal{CL}$ language, it only has relative deadlines where one party's commitment to do something depends on when the other party has performed an action. Our language FCL has not only relative temporal constraints, but also absolute temporal constraints.

Notice that the domains and objectives of these approaches are varied, thus the requirements of the languages are not the same. PJ&E's and BBE's work provide means for the compositional analysis of values of financial contracts. AEHSS is concerned with formalizing commercial contracts. The LegalRuleML TC focuses on the creation of machine-readable forms of the content of legal texts, such as legislation, regulations, contracts, and case law, for different concrete web applications. Most

other approaches, including our contract language FCL, consider the formalization of general contracts that are agreements written by and for humans.

In addition, we view two other works particularly as worthy of investigation. The first is the OASIS's LegalRuleML which meets almost all our requirements of contracts. An essential advantage of LegalRuleML is it is a markup language for the semantic web. The second is the BBE's work. They formalize their contract language in the Coq theorem prover and automatically extract a Haskell implementation from the Coq formalization. The resulting Haskell module can be used as a certified core of a portfolio management framework. Our language fails to express laws and regulations. And, besides that, we do not provide a full implementation of our language or a reasoning system. We primarily focus on providing a formal language that can be used to write contracts precisely.

10.3 Final Remarks

To our knowledge, none of the existing approaches captures all the intuitive properties of contracts we have described in Chapter 3. More surprising is that many approaches lack a formal semantics and a reasoning system. In contrast, this thesis introduces FCL, a formal language for writing contracts that may contain temporally based conditions. FCL has a precise semantics and a reasoning system. We believe FCL meets the first nine requirements and can be extended to meet R10 to express laws and regulations and thus identify agreements of a contract that are in conflict with the underlying laws and regulations of the contract.

Since the standard deontic logic suffers from several well-known paradoxes, a contract that is represented in such a logic may contain a mismatch between intention

of the contract drafter and the actual logical consequences. Unlike many approaches that are based on deontic logic to design a contract formalism, the underlying logic of FCL is simple type theory, thus the semantics of contracts written in FCL is based on very well understood ideas and the logical tools for writing contracts in FCL are very expressive..

Moreover, since FCL is a formal language, software-implemented formal methods can be used to assist in the writing and analysis of FCL contracts. In particular, we can use software tools to check whether an action in a contract has been performed or not, to report whether a contract has been fulfilled or violated, to compute the value of a contract, etc. We can also use software tools to reason about possible future outcomes of a contract and about the relationship between different contracts.

10.4 Future Work

We propose the following improvements as our future work:

(1) Extending the design of FCL. FCL could be extended to handle more situations. One example might be to extend FCL to express laws and regulations and to thus identify agreements of a contract that are in conflict with the underlying laws and regulations of the contract.

(2) Developing a module system for FCL with a rich language for expressing contracts and combinators for building large contracts from smaller contracts.

(3) Developing a software tool to analyze and simplify contracts that could, for example, be used to reason about the fulfillment and breach of contracts.

Appendix A

FCL: The Implementation

We present here the start of the implementation of FCL in Agda.

A.1 Basic Data Types and Expressions

We construct many of our expressions from basic Agda types. Boolean values, integers, and propositional equality are imported from the standard library and opened to make their content available. We include some useful modules from the Agda standard library:

- `Data.Bool` defines the boolean values.
- `Data.Integer` defines the integers.
- `Data.Fin` defines an inductive families to represent the type of all numbers less than a given number.
- `Data.List` defines finite lists.

- `Data.Vec` defines lists indexed by their length.
- `Relation.Nullary` defines a type for decidable relations, Dec:

```
data Dec (P :Set) : Set where
  yes : ( p :  p) → Dec P
  no  : (¬ p : ¬ p) → Dec P
```

In terms of the given types of Agda, we have a type `Time` which is an alias of type $\mathbb{Z}$.

```
Time = ℤ
```

`toffer` and `Xtime` are constants of type `Time`.

```
Postulate
  toffer : Time
  Xtime : Time
```

The declaration of `Currency` data type is given below:

```
data Currency : Set where
  GBP USD EUR ZAR KYD CHF : Currency
```

We use `Pair` to represent the product type.

```
record Pair (A B : Set) : Set where
  constructor _,_
  field
    fst : A
    snd : B
```

The membership relation is defined as follows:

```
data Either (A : Set) (B : Set) : Set where
  left  : A → Either A B
  right : B → Either A B

_isin_ : ∀ {A} → A → List A → Set
a isin [] = ⊥
a isin (x :: xs) = Either (a ≡ x) (a isin xs)
```

A.2 Action

An *action* is a record of type `Action`, which has fields for a `L` of type `string`, an `A` of type `Act`, a `S` of type `List Party`, and an `O` of type `List Party`.

```
record Action : Set where
  constructor action
  field
        L : String
        A : Act
        S : List Party
        O : List Party
```

The label `L` of an action is its name. The `S` and `O` of an action are parties of the contract where `S` is a set of subjects and `O` is a set of objects.

The act of an action is of type `Act`. The `Act` data type has two constructors: `transfer` and `deliver`.

```
data Act : Set where
  transfer : Currency → ℤ → Act
  deliver : Goods → ℤ → Act
```

The declaration of the event datatype is given below. `Event` data type has a constructor `act`.

```
data Event : Set where
  act : Action → Event
```

A.3 Agreements and Rules

Inductive data types are dependent versions of algebraic data types as they occur in functional programming. The data types `Agreement`, `Rule`, and `Q` are given by the simultaneous — as denoted by the keyword `mutual` — inductive data types.

```
mutual
  data Agreement : Set where
    FF : Action → List Time → Agreement
    OO : Action → List Time → Agreement

  data Rule : Set where
    _→_ : Bool → List Q → Rule

  data Q : Set where
    A : Agreement → Q
    R : Rule → Q
```

The constructor `OO` and `FF` are used to represent agreements. `OO` represents the obligations and `FF` represents the prohibitions.

A.4 Contracts

A *contract* is a record of type `Contract`, which has fields for a `time` of type `Time` and a `component` of type `List Q`.

```
record Contract : Set where
  constructor cntract
  field
        time : Time
        component : List Q
```

`List Q` is a type of a set of constant definition, agreements, and rules.

The state of a contract is a record of type `SOC`, which has fields for a `t` of type `Time`, and a `state` of type `List Q`.

```
record SOC : Set where
  constructor State
  field
        t : Time
        state : List Q
```

A.5 Example: An American Call Option

We implement here Example 1.1.1, an American Call Option, as a contract **C** of FCL. We first introduce the constants and their types below.

```
postulate
  buyer seller : Party
  "Buy Option" : String
  "Exercise Option" : String
  "Transfer Stock" : String
  texercise : Time
```

C has two parties: a seller and a buyer. `t0` is the time the seller offered the contract to the buyer be some day before `tbuy`.

```
postulate
 t0 : Time


tbuy = 0
texpire = 170
```

Actions are listed below:

```
a1 = action "Buy Option" (transfer USD 5) (buyer :: []) (seller :: [])
a2 = action "Exercise Option" (transfer USD 80)
                              (buyer :: []) (seller :: [])
a3 = action "Transfer Stock" (deliver stock 1)
                              (seller :: []) (buyer :: [])
```

Events are listed below:

e_1 = act a_1

e_2 = act a_2

C contain an agreement as follows:

A = OO a_3 [texercise,texercise + 30]

C is constructed from two rules R_1 and R_2:

R_1 = obs-event tbuy e_1 $\rightarrow$ (R_2 :: [])

R_2 = obs-event t e_2 $\wedge$ t isin [tbuy,texpire] $\rightarrow$ (A :: [])

Bibliography

[1] J. Ø. Aagedal. *Quality of Service Support in Development of Distributed Systems*. PhD thesis, Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Oslo, 2001.

[2] K. A. Adams. Know your enemy: Sources of uncertain meaning in contracts. *Michigan Bar Journal*, 95(10):40 – 45, October 2016.

[3] J. Andersen, E. Elsborg, F. Henglein, J. Simonsen, and C. Stefansen. Compositional specification of commercial contracts. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(6):485–516, 2006.

[4] A. R. Anderson and O. K. Moore. The formal analysis of normative concepts. *American Sociological Review*, 22:9–17, 1957.

[5] T. Athan, H. Boley, G. Governatori, M. Palmirani, A. Paschke, and A. Wyner. Oasis legalruleml. In E. Francesconi and B. Verheij, editors, *ICAIL*, pages 3–12. ACM, 2013.

[6] T. Athan, G. Governatori, M. Palmirani, A. Paschke, and A. Z. Wyner. LegalRuleML: Design principles and foundations. In Wolfgang Faber and Adrian Pashke, editor, *The 11th Reasoning Web Summer School*, pages 151–188, Berlin, Germany, July 2015. Springer.

[7] R. S. Attorney. *Contracts: The Essential Business Desk Reference*. Nolo, 2010.

[8] P. Bahr, J. Berthold, and M. Elsman. Certified symbolic management of financial multi-party contracts. In *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015*, pages 315–327, 2015.

[9] A. Beugnard, J.-M. Jézéquel, and N. Plouzeau. Contract aware components, 10 years after. *Electronic Proceedings in Theoretical Computer Science*, (37):86–100, 2010.

[10] A. Beugnard, J. M. Jézéquel, N. Plouzeau, and D. Watkins. Making components contract aware. *Computer*, 32(7):38–45, 1999.

[11] B. A. Blum. *Contracts: Examples and Explanations*. Aspen Publishers, fourth edition, 2007.

[12] R. M. Chisholm. Contrary-to-duty imperatives and deontic logic. *Analysis*, 24:33–36, 1963.

[13] A. Daskalopulu. *Logic-based Tools for the Analysis and Representation of Legal Contracts*. PhD thesis, 1999.

[14] A. Daskalopulu. Model checking contractual protocols. *Legal Knowledge and Information Systems, JURIX*, pages 35–47, 2000.

[15] A. Daskalopulu, T. Dimitrakos, and T. Maibaum. Evidence-based electronic contract performance monitoring. *Group Decision and Negotiation*, 11(6):469–485, 2002.

[16] A. Daskalopulu and M. Sergot. The representation of legal contracts. *AI and Society*, 11:6, 1997.

[17] H. Davulcu, M. Kifer, and I. Ramakrishnan. CTR-S: A logic for specifying contracts in semantic web services. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 144–153. ACM, 2004.

[18] W. Farmer. A basic extended simple type theory. SQRL Report 14, McMaster University, 2004.

[19] W. M. Farmer. The seven virtues of simple type theory. *Journal of Applied Logic*, 6:267–286, 2008.

[20] W. M. Farmer and Q. Hu. A formal language for writing contracts. In *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI 2016)*, pages 134–141. IEEE, 2016.

[21] W. M. Farmer and Q. Hu. *FCL: A Formal Language for Writing Contract*, volume Quality Software Through Reuse and Integration of *Advances in Intelligent Systems and Computing*. Springer International Publishing AG, 2018.

[22] S. Fenech, G. J. Pace, and G. Schneider. Automatic conflict detection on contracts. In *International Colloquium on Theoretical Aspects of Computing*, pages 200–214. Springer, 2009.

[23] M. B. Finan. A discussion of financial economics in actuarial models: A preparation for exam MFE/3F. Prepared for Arkansas Tech University, 2015.

[24] G. K. Giannikis and A. Daskalopulu. Defeasible reasoning with e-contracts. In *Intelligent Agent Technology, 2006. IAT '06. IEEE/WIC/ACM International Conference on*, pages 690–694, 2006.

[25] G. K. Giannikis and A. Daskalopulu. The representation of e-contracts as default theories. In H. Okuno and M. Ali, editors, *New Trends in Applied Artificial Intelligence*, volume 4570 of *Lecture Notes in Computer Science*, pages 963–973. Springer Berlin Heidelberg, 2007.

[26] A. Goodchild, C. Herring, and Z. Milosevic. Business contracts for B2B. In *Proceedings of the CAISE00 Workshop on Infrastructure for Dynamic Business-to-Business Service Outsourcing, Stockholm, Sweden*, 2000.

[27] G. Governatori. Representing business contracts in ruleml. *International Journal of Cooperative Information Systems*, 14(02n03):181–216, 2005.

[28] G. Governatori and Z. Milosevic. A formal analysis of a business contract language. *International Journal of Cooperative Information Systems*, 15(04):659–685, 2006.

[29] G. Governatori, Z. Milosevic, and S. Sadiq. Compliance checking between business processes and business contracts. In *2006 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pages 221–232, Oct 2006.

[30] G. Governatori and D. H. Pham. Dr-contract: an architecture for e-contracts in defeasible logic. *International Journal of Business Process Integration and Management*, 4(3):187–199, 2009.

[31] G. Governatori and A. Rotolo. Logic of violations: A gentzen system for reasoning with contrary-to-duty obligations. *Australasian Journal of Logic*, 4:193–215, 2005.

[32] S. O. Hansson. Semantics for more plausible deontic logics. *Journal of Applied Logic*, 2:3 – 18, 2004.

[33] J. C. Hull. *Options, Futures, and Other Derivatives*. Prentice-Hall, 8th edition edition, 2009.

[34] T. Hvitved. A survey of formal languages for contracts. In *Fourth Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS 2010)*, 2010.

[35] T. Hvitved. *Contract formalisation and modular implementation of domain-specific languages*. PhD thesis, Department of Computer Science, University of Copenhagen, 2012.

[36] T. Hvitved, F. Klaedtke, and E. Zălinescu. A trace-based model for multiparty contracts. *Journal of Logic and Algebraic Programming*, 2011.

[37] S. Khosla and T. S. E. Maibaum. The prescription and description of state based systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, pages 243–294, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[38] S. O. Kimbrough. A note on the good samaritan paradox and the disquotation theory of propositional content. In *Sixth International Workshop on Deontic Logic in Computer Science (DEON 02)*, 2002.

[39] M. Kyas, C. Prisacariu, and G. Schneider. Run-time monitoring of electronic contracts. *Automated Technology for Verification and Analysis*, pages 397–407, 2008.

[40] R. M. Lee. A logic model for electronic contracting. *Decision support systems*, 4(1):27–44, 1988.

[41] P. F. Linington, Z. Milosevic, J. Cole, S. Gibson, S. Kulkarni, and S. Neal. A unified behavioural model and a contract language for extended enterprise. *Data & Knowledge Engineering*, 51(1):5–29, 2004.

[42] P. McNamara. Deontic logic. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Winter 2014 edition, 2014.

[43] J.-J. Meyer, F. Dignum, and R. Wieringa. *The paradoxes of deontic logic revisited: a computer science perspective*. Number UU-CS-1994-38. University of Utrecht, 9 1994.

[44] J.-J. C. Meyer and J. Treur. *Handbook of Defeasible Reasoning and Uncertainty Management Systems: Volume 6: Dynamics and Management of Reasoning Processes*, volume Volume 6 of Handbook of Defeasible Reasoning and Uncertainty Management Systems. Springer, 2011.

[45] A. Paschke, J. Dietrich, K. Kuhla, et al. A logic based sla management framework. In *4th Semantic Web Conference (ISWC 2005)*, 2005.

[46] S. L. Payton Jones. Composing contracts: An adventure in financial engineering. In *Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity*, volume 2021 of *Lecture Notes in Computer Science*, page 435. Springer, 2001.

[47] S. L. Peyton Jones and J. M. Eber. How to write a financial contract. In J. Gibbons and O. de Moor, editors, *The Fun of Programming*, Cornerstones in Computing, pages 105–130. Palgrave, 2003.

[48] J. Poole. *Textbook on Contract Law*. Oxford University Press, 11 edition, 2012.

[49] C. Prisacariu and G. Schneider. A formal language for electronic contracts. In *International Conference on Formal Methods for Open Object-Based Distributed Systems*, pages 174 – 189. Springer, 2007.

[50] C. Prisacariu and G. Schneider. Towards a formal definition of electronic contracts. Technical report, Technical Report 348, Department of Informatics, University of Oslo, Oslo, Norway, 2007.

[51] C. Prisacariu and G. Schneider. A dynamic deontic logic for complex contracts. *The Journal of Logic and Algebraic Programming*, pages 458–490, 2012.

[52] A. Ross. Imperatives and logic. *Philosophy of Science*, 11(1):30–46, 1944.

[53] I. Song and G. Governatori. Nested rules in defeasible logic. *Rules and Rule Markup Languages for the Semantic Web*, pages 204–208, 2005.

[54] The Coq Development Team. The Coq Proof Assistant Reference Manual : Version 8.7.2. Technical report, INRIA, 2017.

[55] V. Tosic and B. Pagurek. On comprehensive contractual descriptions of web services. In *The 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service, 2005. EEE'05.*, pages 444–449. IEEE, 2005.

[56] J. van Benthem and A. ter Meulen. *Handbook of Logic and Language*. Elsevier, 2 edition, 2010.

[57] F. Van Harmelen, V. Lifschitz, and B. Porter, editors. *Handbook of Knowledge Representation*, volume Volume 1 of Foundations of Artificial Intelligence. Elsevier Science, 2008.

[58] A. Z. Wyner. *Violations and Fulfillments in the Formal Representation of Contracts*. PhD thesis, Department of Computer Science, King's College London, 2008.