Violations and Fulfills in the Formal Representation of Contracts

Adam Zachary Wyner

Submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy at King’s College London

December, 2008
Abstract

The thesis reports research on the language, logic, and automated processing of the deontic concepts obligation, prohibition, and permission. We provide a flexible, open framework and implemented tool in which to express and exercise alternative definitions of the deontic concepts as applied to complex actions for the purpose of abstractly modeling legal contracts. Deontically specified actions are reduced to fine-grained markers of violation and fulfillment. The research is set within an interdisciplinary context. We linguistically analyse well-known problems of Deontic Logic, resulting in well-motivated logical forms. We focus on the Contrary-to-Duty Paradox in State-wise and State-changing logical approaches to the concepts. The State-wise theory does not treat context change, gives rise to a host of secondary problems, and does not make use of a fine-grained reductionist approach. The State-changing theory does not directly account for the Contrary-to-Duty Paradox, has unrealistic notions of action negation, and must differentiate between obligations on sequences and sequences of obligations. We outline a tool with which one can exercise the operators as they apply to complex action expressions and given a specific choice of definitions of them. The tool provides simple and complex actions as state-transitions, calculable lexical semantic relations among the actions, and deontic specifications on actions which are reduced to fine-grained markers for violation and fulfillment. A contract is given as a contract state, which is a finite list of deontically specified actions relative to an agent, along with a set of rules, which are rules of modification of a contract state relative to violation or fulfillment markers. The relevant forms of the Contrary-to-Duty Paradox are accounted for in terms of how contract states are modified relative to these rules.
# Contents

1 Societies, Laws, and Language ........................................ 18

1.1 Introduction .................................................. 18

1.2 Contents of the Chapter ....................................... 20

1.3 An Interdisciplinary Study .................................... 21

1.3.1 Legal Contracting ......................................... 21

1.3.1.1 Basic Structure of a Contract ....................... 21

1.3.1.2 Promises, Breaches, and Remedies ............... 22

1.3.1.3 Deontic Concepts and Legal Contracting .......... 24

1.3.2 Electronic Contracting ..................................... 24

1.3.2.1 Previous Proposals .................................. 25

1.3.2.2 Limitations ........................................... 25

1.3.2.3 A Focus on Deontic Concepts .................... 26

1.3.3 The Logical Analysis of the Deontic Concepts .......... 26

1.3.3.1 A Family of Deontic Logics ....................... 26

1.3.3.2 An Open Approach to a Complex Problem ........ 27

1.3.4 Linguistic Considerations ................................ 27

1.3.4.1 Natural Language Syntactic and Semantic Analysis .... 28
1.3.4.2 Previous Linguistic Research ........................................... 29
1.3.4.3 Limitations ................................................................. 29
1.3.4.4 Linguistic Analysis and Requirements Analysis ................. 30
1.3.5 Multi-Agent Systems, Organisations, and the Semantic Web .... 31
  1.3.5.1 Topics in Multi-agent Research ................................. 31
  1.3.5.2 Organisations and Norms ............................................. 32
1.4 The Contrary-to-Duty Paradox as the Organisational Topic ........... 32
1.5 Research Objective and Contributions .................................... 33
1.6 Thesis Structure ............................................................... 34
1.7 A Note on Presentational Style .......................................... 36
1.8 Summary ............................................................................ 36
1.9 Next Chapter: State-wise Deontic Logic and its Paradoxes ......... 36

2 State-wise Deontic Logic and its Paradoxes ........................... 37
  2.1 Introduction ..................................................................... 37
  2.2 Chapter Overview ............................................................ 38
  2.3 State-wise Deontic Logic .................................................. 39
    2.3.1 SDL ........................................................................ 39
      2.3.1.1 Syntax ............................................................... 39
      2.3.1.2 Semantics ........................................................ 40
      2.3.1.3 Derived Theorems .............................................. 41
    2.3.2 The Kanger-Andersonian Reduction ................................ 41
      2.3.2.1 Syntax .............................................................. 42
      2.3.2.2 Semantics ........................................................ 43
2.4 Issues with State-wise Deontic Logic ........................................ 44

2.5 The Lexical Semantics of Obligation ........................................ 46
  2.5.1 Background in Lexical Semantics ....................................... 46
  2.5.2 Epistemic and Non-epistemic Interpretations ......................... 47
  2.5.3 Generics, Habituals, and Progressives ................................. 48
  2.5.4 Morphologically and Lexically Related Words ......................... 49
  2.5.5 Evidence to Distinguish Interpretations ............................... 50
    2.5.5.1 Comparing Epistemic and Non-epistemic Interpretations .... 51
    2.5.5.2 Agentive Subjects .................................................. 54
    2.5.5.3 Predicate Restrictions ............................................ 56
    2.5.5.4 Derivational Relationships and Impersonal Obligations ...... 57

2.6 Conditionals and Obligations .............................................. 59
  2.6.1 $[\text{OB}(P) \rightarrow \text{OB}(Q)]$ ........................................ 61
    2.6.1.1 Linguistic Evidence ............................................... 61
    2.6.1.2 An Analysis ....................................................... 62
    2.6.1.3 Additional Considerations ...................................... 63
    2.6.1.4 Restricted Distributions ....................................... 64
  2.6.2 The Wide-Scope Operator ............................................... 64
    2.6.2.1 $\text{OB}[P \rightarrow Q]$ ......................................... 65
    2.6.2.2 $\text{OB}(P/Q)$ ..................................................... 65

2.7 Deontic Operators and Adverbs ............................................ 66
  2.7.1 The Gentle Murderer Paradox .......................................... 66
  2.7.2 Logical Forms of Adverbial Sentences ................................ 68
  2.7.3 Focus ................................................................. 69
2.10 Deontic Logic and the Formal Representation of Contracts ............... 100
   2.10.1 The Epistemic and Non-epistemic Distinction .......................... 101
   2.10.2 Relationships among Obligations ...................................... 101
2.11 Summary .................................................................................. 102
2.12 Next Chapter: State-wise Deontic Logic and the CTD Paradox ............ 103

3 State-wise Deontic Logic and the CTD Paradox 104
   3.1 Introduction .......................................................................... 104
   3.2 Contrary-to-Duties and SwDL .................................................. 105
      3.2.1 The Original CTD Paradox .............................................. 105
      3.2.2 Our Working Form for the CTD Paradox ............................ 106
      3.2.3 Additional CTD Structures ............................................. 109
         3.2.3.1 The Dog Example .................................................. 109
         3.2.3.2 The Gentle Murderer .............................................. 110
   3.3Towards a Reanalysis of the CTD Structure .................................. 110
      3.3.1 Abstracting from Secondary Obligations ............................ 111
      3.3.2 Abstracting from the Primary Obligation ............................ 111
      3.3.3 Alternative Contexts ..................................................... 112
         3.3.3.1 The Primary Obligation is Fulfilled .......................... 113
         3.3.3.2 A Proposition Unrelated to the Primary Obligation ....... 114
   3.4 An Alternative Formalisation ................................................... 115
      3.4.1 An Informal Set of Expressions ...................................... 116
      3.4.2 A Formal Set of Expressions ......................................... 117
      3.4.3 A Comment on Logical Form ......................................... 118
3.4.4 A Generalisation to Other Operators ........................................ 119
3.5 The Criteria .................................................................................. 120
3.6 Summary ....................................................................................... 122
3.7 Next Chapter: Deontic Specification and the CTD Problem in ScDLs .... 122

4 State-changing Deontic Logic, the CTD Paradox, and Sequences .......... 124

4.1 Introduction ................................................................................. 124
4.2 ScDL ........................................................................................... 125
  4.2.1 Basic Elements of ScDL ............................................................. 126
    4.2.1.1 Atomic Actions, Complex Actions, and Action Negation .... 126
    4.2.1.2 Deontic Operators on Atomic Actions in ScDL ............... 127
    4.2.1.3 Deontic Operators on Sequences of Actions in ScDL and the CTD Structure ......................................................... 128
  4.2.2 Background for Problems with the CTD Structure for ScDL ........... 128
    4.2.2.1 An Intuitive Example of the Basic CTD Problem in ScDL 129
    4.2.2.2 A Formalisation of the Basic CTD Structure in SwDL ......... 130
    4.2.2.3 Structures Not Considered ................................................. 132
  4.2.3 The Problems with the Analysis of the CTD Structure in ScDL .......... 132
    4.2.3.1 Problem One: Changing the Primary Obligation ............... 132
      4.2.3.1.1 Informal Presentation ............................................. 132
      4.2.3.1.2 Formalisation of Problem One .................................. 133
    4.2.3.2 Problem Two: Partitioning the Action Space ..................... 135
    4.2.3.3 Problem Three: The difference between Obligations on Sequences and Sequences of Obligations ............................. 136
      4.2.3.3.1 Introduction or Elimination of Obligations .................. 136
4.4.1.1 Basic Definitions ........................................... 157
4.4.1.2 Basic and Complex Types ................................. 158
4.4.1.3 Semantics .................................................. 160
  4.4.1.3.1 Intensional Semantics ............................... 160
  4.4.1.3.2 Synchronicity Sets .................................. 161
4.4.1.4 Propositional Dynamic Logic ......................... 163
4.4.1.5 Propositional Dynamic Deontic Logic ................. 165
4.4.1.6 Prohibitions on Sequences ............................. 166
4.4.1.7 Obligations on Sequences ............................... 167
4.4.2 Deontic Action Logic – [106] .............................. 168
  4.4.2.1 Levels of Analysis .................................... 170
  4.4.2.2 Deontic Action Logic .................................. 172
  4.4.2.3 Discussion ............................................. 174
  4.4.2.4 Management of Deontic Specifications ............... 177
4.5 Action Negation ............................................... 178
4.6 Summary ...................................................... 181
4.7 Next Chapter: Introduction to the Implementation ........ 182

5 The Abstract Contract Calculator .................................. 183
  5.1 Introduction .................................................. 183
  5.2 Bridge to the Abstract Contract Calculator ............... 185
    5.2.1 A Dynamic Language ................................... 185
    5.2.2 Directed Violation and Fulfillment ................... 186
    5.2.3 Obligations on Basic Actions .......................... 186
5.2.4 Spelling out Violation and Fulfillment .......................... 187
5.2.5 Obligations on Sequences ............................................ 187
5.2.6 Action Negation .......................................................... 188
5.2.7 Relationship between Notions ....................................... 189
5.2.8 Negations on Deontic Specifications ............................... 190
5.2.9 CTDs ........................................................................... 191
5.3 Conceptual Overview of the Tool ....................................... 191
5.3.1 A Basic Example ............................................................ 192
  5.3.1.1 Deontic Specification on an Action and a State-of Affairs . 192
  5.3.1.2 The History ............................................................... 192
  5.3.1.3 The Rule Function ....................................................... 193
  5.3.1.4 Updating the VSL with respect to a Triggered RF ......... 194
  5.3.1.5 The Contract and the Value Specified Context ............ 194
5.3.2 A Graphic Representation .............................................. 195
  5.3.2.1 Violating the Obligation ............................................ 195
  5.3.2.2 Fulfilling the Obligation ............................................ 197
  5.3.2.3 Output VSCs and Complex Actions ............................ 197
5.3.3 The Components .......................................................... 197
  5.3.3.1 Basic Components ...................................................... 198
  5.3.3.2 Complex Components ............................................... 198
  5.3.3.3 Functions on the Components ..................................... 199
  5.3.3.4 Graphic Representation ............................................. 200
5.4 Haskell as the Language of the Implementation ..................... 201
5.5 Overview of the Implementation ......................................... 203
5.5.1 States of Affairs .............................................. 204

5.5.2 A Lexicon of Basic and Agentive Actions ................. 206
5.5.2.1 Types for Basic and Agentive Actions ................. 207
5.5.2.2 Executing an Action Relative to a SOA ............... 209

5.5.3 Lexical Space and Lexical Semantic Functions .......... 212
5.5.3.1 Systematic Lexical Relationships ....................... 212
5.5.3.2 Unsystematic Lexical Relationships ..................... 213
5.5.3.3 A Graphic Representation ................................. 213
5.5.3.4 Lexical Semantic Functions ............................... 215
5.5.3.5 Implementation .......................................... 216

5.5.4 Deontic Specifications ....................................... 217
5.5.4.1 The Creation of Value Specification Lists ............. 217
5.5.4.2 Manipulations on ContractFlagStates .................. 220
5.5.4.3 Additional Functionality ................................ 221

5.5.5 Histories ..................................................... 222

5.5.6 Rule Functions and CTDs ..................................... 228
5.5.6.1 Implementation .......................................... 229
5.5.6.2 Basic Examples .......................................... 232
5.5.6.3 Additional Rules .......................................... 234

5.5.7 Deontic Specifications on Complex Actions ............... 239
5.5.7.1 Complex Action Combinators .............................. 239
5.5.7.2 Obligations on Sequences ................................. 242
5.5.7.2.1 \( \text{OB}_{\text{dist}} \) .................................. 242
5.5.7.2.2 \( \text{OB}_{\text{coll}} \) .................................. 243
5.5.7.2.3 \( \text{OB}_{\text{int}} \) ........................................... 243

5.6 Inputs to and Outputs of the Program ............................................. 245

5.7 Some Comparisons ................................................................. 246

5.8 Summary ............................................................................. 250

5.9 Next Chapter: Conclusion ........................................................... 251

6 Conclusion ................................................................................. 252

6.1 Introduction ............................................................................. 252

6.2 Summary of the Results of the Thesis ......................................... 252

6.2.1 General Objective and Contributions ....................................... 252

6.2.2 Contributions by Chapter ...................................................... 254

6.3 Limitations .............................................................................. 255

6.4 Future Work ........................................................................... 257

References ..................................................................................... 261
List of Figures

5.1 Graphic for Actions Executed Relative to Contract . . . . . . . . . . . . . . 196
5.2 Graphic for Actions Executed Relative to a Value Specified Context . . . . 200
5.3 Structure of the Lexical Space . . . . . . . . . . . . . . . . . . . . . . . . . . 214
Acknowledgements

I thank my supervisors who carried on through several shifts in personnel: Aspassia Daskalopulu, Tom Maibaum, Andrew Jones, and Michael Luck.

I am grateful to Hewlett-Packard Research Labs, Bristol, for its sponsorship.

I also thank all my friends, colleagues, and acquaintances for their companionship the last several years.
Dedication

To honour the memory of my mother

She encouraged me to learn and ask questions. With a Ph.D. in education as well as a Juris Doctor, she would have appreciated this research.

Nancy Joan Braverman Wyner (1931-1999)
Chapter 1

Societies, Laws, and Language

1.1 Introduction

Humans are social animals. We live in complex societies comprised of individuals who interact with one another and who are organised into subgroupings. Clearly, if each of us behaved randomly or arbitrarily, our societies would quickly decay. What, then, guides individual behaviour such that from the behaviour of individuals interacting, the complex social structures arise? While the laws found in the physical sciences may, at some level of analysis, be relevant, these do not explain social structures if only because the terms of the physical sciences do not seem to apply: human societies are not crystalline structures formed by chemical bonds. Other animals such as ants exhibit complex social structures. But so far as we know ([93]), individual behaviour and the social structures which arise from them are largely guided by genetic and environmental factors; individual ants react to concrete aspects of immediate, local circumstances. For primates such as baboons, which live in complex societies, there is evidence that individuals have some conceptual capacities which they use to guide their behaviour ([34]): that is, baboons appear to think about what they are doing in order to achieve some goal. The hallmark of this behaviour is that the individuals do not appear to be making behavioural choices solely on the basis of concrete aspects of immediate, local circumstances, but rather with respect to abstract properties, for example, with respect to the ‘place’ of other baboons in the troop’s social hierarchy. However, we can only infer this on the basis of observations. Moreover, we do not know the terms and rules which they use in their thinking. We do not, then, know how individual baboons guide their behaviour.
In contrast, for human societies, we often have explicit information about the terms and rules by which individuals guide behaviour. For example, people make legally binding contracts with one another. Among other things, contracts contain terms for the deontic concepts of obligation, permission, and prohibition. These concepts are applied to expressions of actions which are executed by individuals who are parties to a contract. Thus, the concepts express what an individual ought to do, may do, or should not do. We can say that a deontic concept ascribes a property to an action that an individual performs.

As an example, suppose some individual Bill who is able to perform some simple action such as moving the left toggle up on some control board. We can say that in the expression Bill’s moving the left toggle up is obligatory, the property of obligation is ascribed to the action Bill’s moving the left toggle up. Whether Bill moves the left toggle up or does something else in part depends on how Bill chooses to guide his behaviour with respect to the obligatory property of the action. However, what is it about obligation or, by the same token, the other deontic concepts such that they can be used to guide behaviour?

To address this question, consider that one of the distinctive characteristics of the deontic concepts is that they are violable. Even though Bill’s moving the left toggle up is obligatory, presumably Bill has other actions which he could, in principle, execute instead. The individual has a certain degree of freedom or latitude in his actions. If Bill does one of these other actions, we intuitively understand that he has violated his obligation. Alternatively, if Bill does move the left toggle up, we intuitively understand that he has fulfilled his obligation.

Violability would appear to go part of the way to addressing the question, for we can say that the individuals use violations and fulfillments to guide behaviour. Note that there are notions which are closely related to violability such as exceptions to rules or constraint; we must distinguish among them, but we leave the distinctions aside for the time being. What is it about violation or fulfillment such that they can be used to guide behaviour?

In and of themselves, one might say not much. After all, Bill, having violated his obligation, could just shrug his shoulders and say “So what?” However, if the violations and fulfillments imply something of greater consequence or greater value to Bill, then he may be more motivated to fulfill his obligation and avoid violation. For instance, suppose that Bill, having violated his obligation, is then obligated to pay a penalty fee. If Bill violates this obligation, he would be thrown in jail. We can suppose that Bill certainly does not want to go to jail and is highly motivated not to spend more money than he has to. In contrast, his objections to moving the left toggle up are low; indeed, he may be rewarded financially to do so. These give Bill adequate motivation to fulfill rather than violate his
initial obligation to move the left toggle up. Thus, violations and fulfillments are used to
guide behaviour because, at some point, they are associated with or imply rewards and
punishments that motivate individuals, like an abstract system of *carrots* and *sticks*.

Still, this raises another question. Why have such *intermediate layers*, where violation of
one obligation implies some other obligation, which itself can be violated? For example,
suppose we are considering a rental contract with several layers. One is obligated to pay
one’s rent in full and on time; if one does not, then one is obligated to pay a penalty;
if one fails this, one is obligated to show up in court; if one fails to show up in court,
one is tracked down and put in jail. Why have this layering instead of simply imposing
the rule that if one does not pay one’s rent on time and in full, then one is put in jail?
That would get people to pay their rent on time and in full! But, surely this is too blunt,
simplistic, impractical, and does not allow for any number of reasons for why any one
obligation was not (or appeared not to be) fulfilled. Thus, we speculate that the reason
for the intermediate layers is to allow individuals to behave in a range of alternative ways
in relation to a complex environment and value system, which consequently gives rise to
our rich social structure.

This thesis reports our research on the deontic concepts. We provide an analysis and
representation of the deontic concepts which is relevant to contractual issues. We outline a
computer program which animates our representation of the deontic concepts in contracts.
For us, the analysis, representation, and implementation are the means to *understand* the
deontic concepts and contracts, which are *intrinsically interesting* as well as useful in
applications in computer science.

### 1.2 Contents of the Chapter

In this chapter, we introduce the interdisciplinary setting of our research, then list the
research achievements, and end with an outline of subsequent chapters. We should note
that the next section has a dual role. On the one hand, we review topics which are
relevant to set the broad context of our research, citing the relevant literature, but which
*are not* further elaborated in the thesis. Thus, we set parameters around our discussion.
On the other hand, we introduce topics and approaches which *are* further developed in
subsequent chapters. In particular, in the body of the thesis, we discuss linguistic and
logical issues in depth, citing and discussing literature as relevant. In this chapter we do
not, therefore, follow common practice of first presenting a review of relevant literature
and then the results of our research. In our view, the variety of topics and manner in


which we approach them do not lend themselves to such a presentation. Moreover, in much of the body of the thesis, we evaluate previous approaches and justify our analysis in comparison and contrast to them.

1.3 An Interdisciplinary Study

The issues and approach we take are at the intersection of a range of disciplines and topics, which we discuss briefly here. We start with legal contracting, which restricts the conceptual space in which we consider the research into the deontic concepts. We then consider electronic contracting, which we consider to be a species of legal contract in which the deontic concepts are automatically processed by a computer program. We present linguistic considerations, which we use to ground the conceptual analysis of the deontic concepts in linguistic phenomena. The relationship of our topic to logic, particularly Deontic Logic, is outlined. Finally, we discuss multi-agent systems and related topics by way of broadening the context of the research and pointing out related topics that we could discuss, but do not. While some topics are largely context-setting and motivational, much of the thesis is devoted to linguistic and logical matters, which are developed more extensively in the body of the thesis.

1.3.1 Legal Contracting

Our research relates to the deontic concepts for the purpose of a formal representation of contracts. Therefore, it is relevant to review selected elements of the legal literature on contracts ([144], [146], [19]). Thus, we ground the concepts to their meaning in the law.

1.3.1.1 Basic Structure of a Contract

Contracts are said to have the following basic attributes ([146] and [19]):
• Are oral or written agreements.
• Involve two or more parties.
• Involve the parties in an exchange relationship.
• Introduce promises.
• Are recognised to be legally enforceable.

In addition to these central attributes, contracts are understood to be set in a context of a body of law and values which govern the construction and execution of the contract. For instance, the parties involved in making the contract have to be free to do so; the parties have to be protected from undue advantage; the promises made have to be made in good faith. Out of these topics, we focus on the issue of promises. Though the other topics are important to the overall analysis of legal contracts, promises make contracts distinct from other legal domains which involve the other features. Moreover, without a good understanding of promises, we would not have a good understanding of contracts.

1.3.1.2 Promises, Breaches, and Remedies

[19] elaborates on promises and relates them to notions of breach and remedy. [19, p.468] defines a promise as: an undertaking to act or refrain from acting in a specified way at some future time.\(^1\) Furthermore, the essence of a contract is an exchange of promises. For example: The buyer promises to pay a price for specified land from the seller and The seller promises to convey specified land to the buyer. While in actual contracts the relationship between the particular buyer, seller, objects, and money is important, we simplify issues to two or more agents who promise to undertake to act or to refrain from acting in a specified way at some future time. Furthermore, while exchange relationships are key (somewhat along the lines of Hohfeldian relationships as in [5], [114], [155], [42]), we do not address this dimension of contracts because we want to first be clear about the nature of the deontic concepts themselves, though we believe our analysis and implementation

\(^1\)We use the term “specified” and “deontic specification” in a specific sense found here. In software engineering, a (logical) specification is a theory presentation, which is comprised of a language (e.g. first-order predicate logic) and its axioms. The specification generates its theory, which consists of the consequences of the axioms ([172]). We do not mean “specified” or “deontic specification” in this way, but rather that something “specific” is indicated. A “deontic specification” means that a deontic operator applies in some expression, making it more specific by ascribing an additional “aspect” to it. For example, in Bill’s moving the left toggle up is obligatory, Bill’s action has, in addition to the usual properties it has, the “aspect” of being obliged. An alternative term – “deontic constraint” – would have to be understood as violable as opposed to inviolable, so is not only itself unclear, but presupposes what we seek to examine.
could be fruitfully extended to relationships. Note the centrality of actions with respect to the future.

Contracts may be conditioned, that is, the promises may hold only where certain contingent circumstances occur and not otherwise. For instance, performance of the promised actions above could be conditioned on a *rezoning of the land within a specified time*. If it does, then the participants in the contract must fulfill their promises; if not, then the promises are void. Clearly, complex structures of temporally sequenced conditions could be provided. By the same token, a contract could contain *exceptions* or *exemptions*.

Having made promises, the next issue between the parties is whether or not the parties have *honoured* them. If not, then it is said a *breach of contract* has occurred. A breach occurs where the performance which was promised did not occur or was not in compliance with the promise. For example, where the *The buyer promises to pay a price for specified land from the seller* and whatever conditions on the execution of this promise have been satisfied, the buyer either pays the price or does not pay the price. Where the buyer does pay the price, then the promise has been honoured; where the buyer does not pay the price, then the promise has been breached. Similar points may be made for the seller’s promise. Breaches can have various degrees: material and total, material but partial, substantial. Following these different sorts of breaches, different consequences follow as specified by the law. For our purposes, we shall take breaches to be *all or nothing*; that is, either the action promised occurred (no breach) or did not (breach) and shall not differentiate among the consequences which follow the different degrees. Clearly, though, the identification and specification of the breach is a central concern in legal reasoning about contracts.

When a breach occurs, a remedy is prescribed relative to the sort and extent of the breach, the role of the parties, and a range of other factors. A remedy could be, for example, a promise to pay a penalty fee. At some point, remedies are backed by the force of law (e.g. jail or some other action by legal authorities) to ensure that they are, unlike the contractual promises, not breached. Note that remedies are tightly associated with the sort of breach; that is, a material and total breach is deserving of one remedy, while a material but partial breach is deserving of another. We assume that a remedy is prescribed, but do not relate these prescriptions to varieties of breach, roles, or other legal aspects.

We note a significant difference in the relation between a condition and a promise and between a breach and a promise; that is, conditions and breaches are not the same. While a condition may hold (or not) irrespective of some other promise, a breach only holds following an unfulfilled promise. We may see a breach as a particular *sort* of condition, namely one which itself is conditioned on an unfulfilled promise. This implies that breaches...
arise systematically, while ordinary conditions need not be systematic. In the body of the thesis, we largely focus on the structure of promise, breach, and remedy, though contrast it at points to condition and promise.

1.3.1.3 Deontic Concepts and Legal Contracting

It is rather notable that the language in which contracts are discussed is not always in terms of obligation, prohibition, and permission. For instance, in [19], we see promises, while in [146, p.368] we find instead contractual obligation and duty. We may similarly understand refraining from acting as prohibition in the contractual context where one has promised to refrain, implying that there can be a breach if the promise is not honoured. We shall assume that we can use the deontic notions. Thus, where we find promise in a legal contract, we interpret it in terms of obligation, prohibition, or permission. Furthermore, we want a more neutral term than breach, so where a promise is honoured, we say the deontic specification is fulfilled; where it is breached, we say it is violated. Remedies are taken to be consequences which follow from a violation. These may be additional deontic specifications. By the same token, we assume it to be possible that there are consequences which follow from fulfillment such as might occur where a party’s fulfillment of an obligation is a condition for another party’s obligation.

Finally, it is not our intention to model actual contracts, but to model central aspects of them as they are taken to be in legal contracting (section 1.3.1) and in AI and Law (chapters 2, 3, and 4), though making these abstract aspects as related to actual notions based in natural language as seems reasonable.

In the following section, we discuss aspects of electronic contracting, then return to comment on some comparisons between legal and electronic contracting.

1.3.2 Electronic Contracting

With the rise of the World Wide Web and commerce conducted over the internet automatically, electronic contracting has become an important research and application area ([51], [52], [48], [49], [50]). Where human contracting covers the negotiation, agreement, execution, and adjudication of legally binding contracts between two or more parties, automated contracting may be taken to mean the automated processing of such behaviors by computer agents, say over the Internet. A basic example might be that a human agent empowers her computer agent to search for an apartment to rent, to negotiate (within parameters) the terms of the contract, and to make a legally binding agreement between
the parties. Another example might be two companies which automate the processes of exchanging goods and services for money. As successful contract formation, execution, and monitoring is key to a successful enterprise, yet expensive in terms of time and money, automation could yield significant savings. As we provide an implementation of our representation of contracts, our research is in the context of electronic contracting.

1.3.2.1 Previous Proposals

In [53] and [47], we find some of the earliest discussions of how one might use logic tools in a variety of ways to construct and execute contracts, for example, to provide a support tool for writing contract documents such that one can identify implicational relations between contract clauses. A somewhat related work ([69]) focuses on Service-level Agreements. Other proposals provide schematic overviews of automated contract frameworks ([14], [138], [137], [25], [161], [160], and [159]), where the phases and elements of contracting are outlined. Another approach is to focus on logical languages in which we can represent contracts along the lines of a Formal Language of Business Communication ([107] and [54]), which is a sublanguage of the event semantics ([55] and [142]). Focussing on XML mark-up languages to represent the rules of a contract, the deontic notions appear in a Defeasible Logic ([82], [84], and [86]).

In addition to these notions, a host of subsidiary issues arise. How does one locate a negotiating partner? How do the agents communicate and negotiate? If there are disputes, how are they resolved. What are the components of a legally binding contract? What is trust between the agents? What is security? What is the ontology for the legal domain? How do we contextually relativize the interpretation of legal terms? What is an agent and what are its ‘rights’? And there are many other issues.

1.3.2.2 Limitations

As a research project with relevance to e-commerce, the implementation is not a commercial product, does not provide support for commercial products, nor can it be processed over the Internet. As a theoretical study, we do not implement an existing contract.

We should note that legal and electronic notions of contracts may differ. In particular the latter can be a simulation or animation of the former. Consequently, a range of differences may arise concerning each and their relationship. However, we simplify and do not address this difference, focussing more on taking legal contract notions as a ground for related notions in the implementation.
1.3.2.3 A Focus on Deontic Concepts

Throughout this literature on e-commerce, a key and constant issue appears to be how to formalize and automate deontic concepts such that one could calculate with respect to good and bad behavior? We discuss this further in section 1.3.3 as well as in subsequent chapters. However, in specific relation to electronic contracting, [52] view a contract as a set of deontically specified actions: actions are state-transitions and the deontic concepts define the acceptable or unacceptable states which result from the execution of particular actions. This is a conception of the formal representation of contracts which we broadly adhere to.

1.3.3 The Logical Analysis of the Deontic Concepts

In the logical literature, we also find discussions of the deontic concepts and contracting. [162], [99], and [156] claim that formal, logical representations of the deontic concepts are central to reasoning with legal notions such as those found in a contract. While the deontic concepts are discussed in the law (as outlined in section 1.3.1) and e-commerce (as outlined in section 1.3.2), they have been analysed in particular depth in the logical literature. In this section, we only tie the logical literature into our interdisciplinary approach; a substantive portion of the thesis is devoted to in depth presentations of logical formalisms and issues.

1.3.3.1 A Family of Deontic Logics

Deontic Logic is that area of logic particularly concerned with the logical analysis of the deontic concepts given as logical operators in a formally precise language with an explicit syntax and semantics. We have several surveys of the area ([74], [92], [133], [185], [128], and [163]). There are two main varieties, which we refer to as state-wise ([128] and [30]) and state-changing ([130], [106]) deontic logics, SwDL and ScDL, respectively. Broadly speaking, SwDLs do not reason with respect to change that arises with the execution of actions over time, while the state-changing deontic logics do. We focus our attention on these main approaches. The deontic concepts are also formalised in other logics or formal languages: Defeasible Logic ([141], [80], [83], and [85]), the Event Calculus ([69], [70]), or the Causal Calculus ([166] and [10]). In chapters 2-4, we discuss key issues, pointing out what we do or do not adopt or adapt. In this chapter, we confine ourselves to general, context-setting remarks.
1.3.3.2 An Open Approach to a Complex Problem

In the body of thesis, we review issues related to previous analyses of the deontic operators. As the literature shows, there are a large and heterogeneous set of problems that arise for any analysis of the deontic operators. There are, as well, a range of alternative proposals. However, there is not widespread consensus on either the problems or proposals. Moreover, we found that several of the problems are based on weak linguistic analyses, which we touch on in the next section, then return to in chapter 2.

In reviewing these issues, our objective is not to argue against any particular analysis of Deontic Logic, which can stand as a formal theory irrespective of the linguistic observations or the lack of consensus. Rather, our evaluations are based on an approach to the deontic concepts which is more grounded in common sense reasoning, natural language, and their use in legal contracting than are other approaches. Furthermore, and in contrast to previous proposals, we do not propose some specific logic which claims to account for all the problems and is useful for the formal representation of contracts. Instead, we provide a flexible and open implemented framework in which to express alternative interpretations of the deontic concepts. Given the implementation, we can experiment with these alternative interpretations to see if they suit our understanding of the deontic concepts relative to a particular domain of application.

1.3.4 Linguistic Considerations

Electronic contracts, as a formalised species of legal contracts, are derived from a conception of contracts expressed in natural language. Furthermore, in our view, the formalisations of deontic logics are derived from semantic intuitions of natural language expressions. Therefore, it makes sense to provide well-founded linguistic analyses of the relevant expressions as a first step in the development of a formal logical theory.

What sets this thesis off from much of the previous work in Deontic Logic is the linguistic turn applied to the problems of Deontic Logic. We apply our knowledge of the analysis of the syntax and semantics of natural language to the analysis of the deontic concepts. Some literature in contemporary linguistic and computational linguistic approaches to the syntax and semantics of natural language can be found in [37], [136], [134], [117], [36], [45], [72], [35], [119], [150]. As with the logical analysis, in this section we tie linguistic issues to our overall research topic. In chapter 2, we provide more detailed background and analyses, particularly as they bear on the analysis of the deontic concepts. In later chapters as well, we ground our analysis in linguistic observations.
1.3.4.1 Natural Language Syntactic and Semantic Analysis

The deontic operators are *sentential operators* in a linguistic analysis; that is, they are operators which apply to expressions to produce propositions. As such, they bear a significant resemblance to other sentential operators in natural language such as *rudely* as in *Rudely, Bill left without saying goodbye* and on which we have done previous research ([189] and [195]). We can, then, draw on aspects of the analysis of other sentential operators to clarify our view of the deontic operators.

Broadly speaking, a substantive body of linguistic research has been devoted to the study of the *syntax-semantics interface*, which is concerned with systematically parsing and then semantically interpreting linguistic expressions in a logical representation that can be used for inference. The research explores a wide spectrum of natural language expressions, issues, and problems which have not yet received attention in the purely mathematical or logical literatures. Furthermore, as an *empirical* research, logic and set-theory are taken as the formal *tools of analysis* rather than the objects of analysis. In addition, like other empirical sciences and following the approach introduced in [136], the emphasis is on a rational analysis of a *particular domain of problems* rather than the language as a whole; the result of the analysis is a *fragment of an overall formal language*, rather than a *complete* formal language as in mathematics or logic. The fragment is subject to revision as empirical data come to be better understood or as the formal language is developed. The fragment also may introduce informal machinery as a stop-gap in addressing some problem as well as make assumptions that a fully developed formal theory might not make. This implies that some central concerns of mathematics, logic, and computer science are not engaged, for the time being, in favour of advancing clarity on the phenomena. Presumably, once the data are fully understood and we have in hand the technical capacity to account for natural language as a whole, then we may provide a formalisation with suitable properties to satisfy these concerns.

The linguistic turn is related to our general objective of grounding our analysis of the deontic concepts and making that analysis relevant to contracting. Thus, our analysis hews closely to empirical data and attempts to *integrate* the analysis of the deontic concepts with related phenomena found in natural language semantics. However, as it is *not* an exercise in the context of linguistic research, we have eschewed a formal, compositional analysis or many other issues related to the syntax-semantics interface.

Our considerations lead us to abstract from narrow linguistic issues in order to focus on what we believe to be a key problem which has not previously been sufficiently examined. In particular, we argue that the complex structure of the expression to which the deontic
operator applies to must be taken into consideration. As a case study, we examine obligations on sequences of actions, which we understand as protocols, such as Bill is obliged to move the left toggle up, and then the right toggle right. We believe, but do not argue for in depth, that the relationship between a deontic operator and a complex expression deserves greater attention, particularly where one wants to apply deontic operators in “real world” domains.

1.3.4.2 Previous Linguistic Research

Though some linguists have analyzed the syntax and semantics of some aspects of deontic expressions ([183], [110], [100], [97], among others), they either focus on syntactic issues rather than semantic issues that are relevant to the logical analysis or they do not relate their research to Deontic Logic. Our work is, therefore, a bridge between these various areas.

1.3.4.3 Limitations

We should point out that we are not doing linguistic data-mining. So far as we know, little work on data-mining deontic concepts has been carried out, and we claim no expertise in the area. There is related work in rule identification ([169], [168], and [135]), but it comes to indeterminate conclusions. By the same token, we are not providing a system to parse sentences into their constituent structure. Nor do we formalise a compositional semantic analysis as in Montague Grammar ([136]). While any of these approaches would be interesting, useful, and eventually ought to be carried out, they would take us too far afield of related approaches in computer science and deontic logic.

Our linguistic approach is not free of its own problems. First, we cannot presume familiarity with linguistics, must present this material as accessibly as possible, and counter presumptions concerning language and linguistics. Yet, we cannot devote substantive sections of the thesis to revision. By the same token, the literature on formalisations of the deontic concepts is large, complex, and technical; there is little consensus among researchers about the intuitions, logical forms, or formal theories. We cannot, then, purport to linguistically address or resolve every issue that has been raised in the literature on deontic logic, but we pick and choose, making comments as to why certain choices are made and what remains to be done.
1.3.4.4 Linguistic Analysis and Requirements Analysis

To bridge between our approach and computer science, we can take linguistic analysis in this context as a means of carrying out requirements engineering on the deontic concepts themselves. Requirements engineering has five main stages ([139]):

- eliciting requirements
- modelling and analysing requirements
- communicating requirements
- agreeing on requirements
- evolving requirements

Requirements engineering highlights “real-world goals” in motivating software development, where the goal may be to formally model some domain or system of interest which is not itself formalised. The goals must be precisely specified so as to support the analysis of the domain, validate that the requirements are as the stakeholders want, define the structure of the software, and verify that the software meets the specification. Furthermore, as the understanding or use of the domain changes, so must the specification and the software.

Logic is the vehicle for the formal analysis of the goals. However, since what is being modeled is not itself formalised, there are substantive issues concerning when and how to formalise. We cannot presume that any particular logic is itself capable of formalising the domain or system, but may have to modify our logical analysis so as to meet the needs of the goals.

Requirements engineering is also concerned with what the stakeholders believe about the domain (epistemology), what is observable in the world (phenomenology), and what the stakeholders agree is objectively true (ontology). Thus, in providing an analysis of the deontic concepts, we want to address these issues as best we can.

We can take a linguistic analysis as a form of requirements elicitation. It provides a means to elicit requirements via comparison and contrast of linguistic forms as well as analysis of logical forms using linguistic theory. We model and analyse our requirements for the deontic concepts in a formal language. By clarifying what we take the requirements to be and their analysis, we can better communicate them to others. As we take an evidential
address (given the best evidence we have), we provide the means to argue over and agree on the requirements, allowing them to change as need be.

1.3.5 Multi-Agent Systems, Organisations, and the Semantic Web

Our research is broadly set within Multi-agent Systems, which concerns formalisations and implementations of systems of acting and interacting agents which form complex social organisations ([182]). Simplifying, we can construe the term agent with an individual object which can act. In this section, we briefly mention some of the ways our research is or is not related to multi-agent systems research. In chapter 5, we have some further discussion of the relationship between our work and related work found in multi-agent systems research.

1.3.5.1 Topics in Multi-agent Research

A fundamental component of a multi-agent system is that the agents interact. To do so, agents usually have a means to communicate. In the subarea of electronic commerce, communication languages and protocols have been proposed ([44], [43], [115], [107]). Furthermore, it has been proposed that the internal mental states of agents are relevant to modeling complex, multi-agent systems as in the Belief-Desire-Intention analysis ([188]). A somewhat related field is Social Simulation, which is a form of computational modeling in which multi-agent systems are defined and run for the purpose of generating data and thereby simulating aspects of social interaction ([13], [79], [59], [61], [71], [68]). Finally, multi-agent systems research is often tied to the semantic web. In the semantic web, web-based information is enriched with semantic markers which support query, reasoning, and reaction ([17]). Agents which engage in electronic commerce must be endowed with the capacities to use such information and to engage in query, reasoning, and reaction.

While such topics in multi-agent system research are broadly relevant, we do not take them into consideration here. Indeed, given our focus on the deontic concepts, our agents do not interact with one another, nor do our agents have complex internal structure. We are abstracting from multi-agent systems research. Therefore, communication between or beliefs of agents are not discussed. As our implementation does generate data with respect to the behaviours of agents relative to a contract, it can be taken as a rudimentary social simulation system. However, the thesis does not develop this line of inquiry, favouring a largely theoretical analysis of the deontic notions and a prototype implementation. And
while our analysis and implementation is broadly compatible with the objectives of the Semantic Web, we do not provide semantic markers or web oriented tools.

1.3.5.2 Organisations and Norms

A large body of research addresses the specification of organisations, taking into consideration roles, a range of conceptions of norms, powers, organisational hierarchies, and trust ([9], [62], [179], [123], [10], [124], and [157]). Some of this research focuses on contracts ([22], [23], [21]). In this literature, norms are taken to be highly relevant in multi-agent systems research. As norms are often related to the deontic concepts, our research has a place in relation to multi-agent systems research.

We do not engage with issues related to organisations and norms because we are primarily focussed on the analysis and implementation of the deontic concepts. The formalisation and implementation of organisations would take us to issues beyond the scope of this research. More importantly, as we discuss further in chapter 2 and again in chapter 5, we must be clear about the very concept of norm, which in our view, is not sufficiently clear in the literature on organisations.

Finally, we may add here, though this is a thesis within Computer Science, a host of computer science issues are not addressed such as complexity, speed, or internet functionality. Rather, its place in Computer Science is established by a body of work in which this thesis follows ([165], [109], [130], [106], and [98], among others).

1.4 The Contrary-to-Duty Paradox as the Organisational Topic

Though the thesis discusses a range of problems and approaches to the deontic concepts, it is organised around a single topic – the so-called Contrary-to-Duty Paradox (CTD Paradox). As we discuss in chapter 3, this is a structure of statements which give rise to counter-intuitive implications. In the introduction to the thesis, we gave an intuitive and informal example which relates to this paradox: Bill’s moving the left toggle up is obligatory; Bill does some action other than moving the left toggle up, thereby violating his obligation; having violated his obligation, Bill is then obligated to pay a penalty fee. The CTD Paradox is widely regarded as distinctive to the deontic concepts ([128] and [30]), and it is widely discussed. It is also central to contracting, as we discussed in section 1.3.1, where it is there understood as the structure of promise-breach-remedy.
Chapter 1

1.5 Research Objective and Contributions

Over the course of the thesis, we discuss various aspects of the paradox, how others have approached it, and related problems. Though at points, the thesis appears to digress to the related problems, we will see how they bear on them.

1.5 Research Objective and Contributions

In carrying out our research, we found that there is no consensus analysis of the deontic concepts which is formalisable in the logical sense, which is relevant to the formal representation of contracts, or well-motivated by considerations of natural language. In light of this, our general objective in the thesis is:

**General Objective:** to provide a flexible, open framework and implemented tool in which we can express and exercise alternative definitions of the deontic concepts as applied to complex actions.

The body of the thesis presents our observations and arguments concerning the lack of consensus analysis about the deontic concepts, develops the framework in a stepwise fashion, and describes the tool.

In meeting our general objective, our research makes the following major and related minor novel contributions to research in the field:

i. *linguistically* well-motivated semantic interpretations of the deontic concepts as well as *linguistically* well-motivated logical forms for expressions which contain the deontic concepts. These interpretations and logical forms are suitable for the formal representation of contracts.

   a. distinction between epistemic and non-epistemic interpretations of the deontic operators;

   b. restriction of the available forms of conditional obligations;

   c. solution to the *Gentle Murder Paradox* using *focus*;

   d. solution to the *Good Samaritan Paradox* using *Discourse Representation Theory* and introduction of the related *restrictive relative clause* problem; and,

   e. introduction of the *Antonym Restriction* on obligations.
ii. a reanalysis of the CTD Structure in State-wise Deontic Logics.
   a. abstraction and generalisation of the CTD structure; and,
   b. introduction of the violation and fulfillment markers as intermediate concepts in State-wise Deontic Logics;

iii. a reductionist, fine-grained analysis of deontic operators in State-changing Deontic Logics.
   a. introduction of complex violation and fulfillment markers for State-changing Deontic Logics;
   b. reanalysis of the CTD structure in State-changing Deontic Logics;
   c. distinction between obligations on sequences and sequences of obligations; and,
   d. interpretation of action negation as the lexical semantic relation of antonym;

iv. a generic prototyping tool for exercising the deontic operators relative to some particular definition of them.
   a. generation of trails of executions of actions by agents, indicating the agents’ violation or fulfillment of the deontic specifications;
   b. introduction of contracts as deontic specifications on actions and a rule function;
   c. violation and fulfillment markers recorded as secondary effects of the execution of an action and relative to a contract;
   d. lexical semantic functions on actions; and,
   e. alternative definitions of deontic specifications on complex actions;

1.6 Thesis Structure

In overview, the chapters to follow are:

- Chapter 2: Standard Deontic Logic and its Paradoxes
  
The chapter presents the research contributions (i). We review State-wise Deontic Logics as represented by Standard Deontic Logic and the so-called Kanger-Andersonian Reduction. Several problems and paradoxes are presented. Evidence is presented for a distinction between norms which express what usually occurs from norms which imply violations. For contracting, we focus on the latter interpretation.
We give extensive linguistic evidence and arguments by which we restrict forms of conditional obligations and resolve two paradoxes of Deontic Logic. We more briefly discuss our positions on several other key issues of Deontic Logic. A main objective of the chapter is to clear the way for a simplified discussion of the so-called Contrary-to-Duties Paradox (CTD Paradox).

- Chapter 3: A State-wise Deontic Logic and the CTD Paradox
The chapter presents the research contributions (ii). We present the CTD Paradox. We show a variety of ways in which to reanalyse the paradox by abstracting over its components and considering alternative contexts. We provide an alternative formalisation using intermediate concepts which represent violation and fulfillment. We review and evaluate the proposal of [30], outlining why we do not adopt it or state-wise deontic logics for our purposes.

- Chapter 4: State-changing Deontic Logic, the CTD Paradox, and Sequences
The chapter presents the research contributions (iii). We initially present basic elements of State-changing Deontic Logic and key problems it faces in the analysis of the CTD Paradox. In particular, we distinguish between obligations on sequences (an obligation on a protocol) and sequences of obligations. We introduce complex violation and fulfillment markers to provide a reductionist, fine-grained analysis for obligations on complex expressions, showing how they address the CTD Paradox. We compare and contrast two alternative proposals of state-changing deontic logics which provide different logics for deontic operators on complex expressions. Finally, we discuss alternative notions of action negation.

- Chapter 5: The Abstract Contract Calculator
This chapter presents the research contributions (iv). The tool provides abstract actions in terms of preconditions and postconditions. Actions are executed with respect to a state-of-affairs where a contract holds. A contract has two components. The first component is a list of deontic operators as applied to actions. Deontic operators on actions are reduced to expressions of fine-grained violation and fulfillment markers on complex expressions plus the action and agent. We use lexical semantic functions to define the actions which fulfill or violate a deontic operator applied to an action. The second component is a list of rules which determine how the first component is modified relative to what action has occurred. We use the interactions between these two components to model the CTD problem in a dynamic language.
Chapter 6: Conclusions

The chapter reviews how the thesis has attained the main objective and provided the research contributions. It closes with some suggestions for future work.

1.7 A Note on Presentational Style

Given the interdisciplinary nature of the research, our evaluations of previous proposals, and our contributions, we have followed a presentational style which suits the material. In chapters 2-4, we first outline logical frameworks of the deontic concepts against which we consider several problems, and then how we address the problems. We cite the contributions of others, which we compare to our contributions. In chapter 2, which contains several subissues and linguistic approaches, we review the necessary background to make sense of our proposals. In chapters 3-5, we make our contribution, then evaluate comparable prior proposals.

1.8 Summary

In this chapter, we have given an overview of the main issues which the thesis addresses, the context of the research, the research contributions, and an overview of subsequent chapters.

1.9 Next Chapter: State-wise Deontic Logic and its Paradoxes

In the next chapter, we introduce State-wise Deontic Logic and its paradoxes. We discuss a range of linguistic evidence and theory which contributes to clarifying the interpretation of norm that is appropriate for our domain as well as the analysis of several complex structures where the deontic operator applies to expressions with conditions, adverbs, and non-restrictive relative clauses. We then present our positions on a range of addition subsidiary (and important) issues. One objective of the chapter is to clear the way for a simplified discussion of the CTD Paradox.
Chapter 2

State-wise Deontic Logic and its Paradoxes

2.1 Introduction

Our goal in this chapter is to provide linguistically well-motivated semantic interpretations of the deontic concepts and the logical forms for expressions which contain them. The primary motivation for this goal is that we want interpretations and logical forms which are suitable for the formal representation of contracts and for our implementation. As we discussed in chapter 1, linguistic analysis is key since legal and electronic contracts are expressed in or derived from natural language; and furthermore, intuitions concerning the deontic concepts are based on natural language. While it was not uncommon to discuss a linguistic approach ([183], [74], [31], [80], [100]), it has fallen out of favour; the chapter suggests by example that this be reconsidered.

As part of the goal of this chapter, we keep in mind our organisational topic – the Contrary-to-Duties Paradox (CTD Paradox). In this chapter, we do not discuss this topic, which is discussed extensively in chapters 3 and 4. Nonetheless, the topic is in the background, for we have abstracted several issues away from the CTD Paradox so when we return to it, we can provide a more straightforward discussion. In our view, problems concerning the deontic concepts are easily conflated; thus, our approach is a strategic, analytic modularisation.
Chapter 2

2.2 Chapter Overview

We review one main approach to Deontic Logic, *State-wise Deontic Logic* (SwDL)\(^1\), introduce and analyse a range of problems and issues that arise with it, and take positions on these problems and issues. While we do not adopt an SwDL analysis, much of what is known about the deontic concepts is with respect to this analysis ([128], [163], [30]). Thus, in order to review what we know and understand, to clearly compare and contrast the alternative approaches, and to identify components which ought to be incorporated into our implementation, it is necessary to discuss SwDL and its problems in some depth.

### 2.2 Chapter Overview

In the following, we provide first two versions of SwDL – *Standard Deontic Logic* (SDL) and the so-called *Kanger-Anderson Reduction* (KAR). The latter reduces the deontic operators to Alethic Modal Logic and a *violation* marker, which the former does not. We point out that though SDL and KAR are two *alternative* formalisations, their relationship in the literature is ambiguous. Our discussion is one of the steps in our analysis of the role of violation and fulfillment markers in deontic logic. In the presentation of the deontic operators, we focus largely on *obligation*, which we represent as \(\text{OB}\); where relevant, we also discuss *permission* (\(\text{PE}\)) and *prohibition* (\(\text{PR}\)). The main reason is that \(\text{OB}\) is the most interesting, complex, and relevant to the discussion of the CTD Paradox.

Then several issues of SwDL are discussed in depth: the lexical semantics of *ought* and *obliged*, where we distinguish between the epistemic and non-epistemic interpretations; a restriction on the logical forms of conditional obligations based on linguistic evidence; a solution to the *Gentle Murderer Paradox*, using *focus*; a solution to the *Good Samaritan Paradox*, using *Discourse Representation Theory*. These topics not only demonstrate the value of linguistic analysis in clarifying issues relevant to Deontic Logic, but also help to simplify our discussion of the CTD Problem in chapter 3. We then more briefly discuss a range of additional issues and take positions on them by way of identifying concepts that need to be addressed in the implementation. We end the chapter with some discussion of the relevance of Deontic Logic to the formal representation of contracts, a chapter summary, and an overview of chapter 3.

---

\(^1\)SwDL is our cover term for *Standard Deontic Logic* and closely related logics.
2.3 State-wise Deontic Logic

In this section, we review two forms of SwDL – Standard Deontic Logic (SDL) and the so-called Kanger-Andersonian Reduction (KAR). Introducing KAR highlights two issues that are relevant as our discussion develops: the use of explicit violation and fulfillment expressions and the need for the expressions to be articulated.

In the following, we present a sketch of SDL and KAR, focusing on relevant definitions and issues.

2.3.1 SDL

Standard Deontic Logic is a Normal Modal System ([33] and [95]), where the syntax and semantics of the deontic operators is largely based on a presumed analogy with the alethic operators necessity $\Box$ and possibility $\Diamond$. As we discussed previously, having such an analogy and formal theory is clearly an advantage in many respects since it gives structure to considerations in the domain of inquiry. However, it can also be a drawback in that it gives greater and lesser prominence to certain issues than they might otherwise have without the analogy. In our view, the better strategy is to treat the logical analogy and the formal theory it provides as one approach, not the approach, to identifying the semantic properties of the operators for our domain.

We follow the presentation in [128], where we assume classical Propositional Logic, an infinite set of propositional variables, and the operators $\neg$, $\rightarrow$, and $\text{OB}$, which stands for obligation.

2.3.1.1 Syntax

Definition 1 (System KD – Syntax)

- **PC WWFs**: All valid, well-formed formulae of Propositional Calculus are theorems of KD.
- **OB-K**: $\vdash \text{OB}[P \rightarrow Q] \rightarrow (\text{OB}(P) \rightarrow \text{OB}(Q))$
- **OB-D**: $\vdash \text{OB}(P) \rightarrow \neg \text{OB}(\neg P)$
Chapter 2 2.3 State-wise Deontic Logic

- **Modus Ponens**: if ⊢ P and ⊢ P → Q, then ⊢ Q

- **OB-NEC**: if ⊢ P, then ⊢ OB(P).

Modal systems which include PC and **Modus Ponens** along with respective versions of OB-K and OB-NEC are normal modal systems.

OB-K says that if some conditional expression is obligatory, then if the antecedent is obligatory, so is the consequent. OB-D says that if a proposition is obligatory, then it is false that the negation of the proposition is obligatory; in other words, there can be no conflicts about what is obligatory; it cannot hold, for example, that OBp and OB¬p. OB-NEC says that all theorems are obligatory.

Just as necessity is interdefinable with possibility, obligation is interdefinable with permission PE, where PE is a modal operator: to be obligated with respect to A means to not have permission that not A holds, and vice versa.

**Definition 2 (INTER)**

\[ \vdash PE(Q) \equiv \neg OB \neg(Q) \]

Furthermore what is prohibited PR, where PR is a modal operator, can be defined in terms of permission.

**Definition 3 (Prohibited)**

\[ \vdash PR(Q) \equiv \neg PE(Q) \]

2.3.1.2 Semantics

The models for the semantics of SwDL are standard modal logic models. A model is \( M = (W, R, \mathcal{V}) \), where W is a non-empty set of worlds, R is a binary relation on worlds (the accessibility relation), and \( \mathcal{V} \) is the valuation function which assigns sets of worlds to atomic sentences. \( \mathcal{V}(p) \) denotes the set of worlds where p is true. The D schema is valid where the accessibility relation R is serial, where wRv is read as w is in the accessibility relation to v: \( \forall w \exists v wRv \).

For Deontic Logic, we assume that the accessibility relation wRv means that v is a deontic alternative to w. Another way to say it is that wRv means that v is an ideal version of w.
We have the definition of a formula $A$ true in a world $w$ of a model $M$, written $M \models_w A$. For the deontic expressions, this means the following:

**Definition 4 (System KD – Semantics)**

Models for $OB(P)$

$M \models_w OB(P)$ if and only if $\forall v (if wRv, then M \models_v P)$

Models for $PE(Q)$

$M \models_w PE(Q)$ if and only if $\exists v (if wRv, then M \models_v Q)$

This is to be understood informally as the formula $OB(P)$ is true in world $w$ if and only if $P$ is true in all of the *ideal* versions $v$, which are just those worlds accessible from $w$. Those worlds accessible from $w$ in which $\neg P$ is true are understood to be *subideal* versions.

### 2.3.1.3 Derived Theorems

In System KD, that *obligation* implies *permission* follows trivially from $OB$-$D$ and Definition 2.

The theorem $OB$-$RM$ can easily be proven.

- Show: If $\vdash P \rightarrow Q$, then $\vdash OB(P) \rightarrow OB(Q)$.

- Proof: Assume $\vdash P \rightarrow Q$. By $OB$-$NEC$, $\vdash OB(P \rightarrow Q)$, and then by $OB$-$K$, $\vdash OB(P) \rightarrow OB(Q)$.

Several other theorems and derived rules can be provided, but these serve our purposes.

### 2.3.2 The Kanger-Andersonian Reduction

In addition to the SDL version of SwDL, we have what is called the *Kanger-Andersonian Reduction* (KAR) of the SDL deontic operators to *Alethic* Modal Logic plus a designated proposition that indicates violation (or, alternatively, fulfillment). The underlying idea is that explicit specification of violation (or fulfillment) is *intrinsic* to the deontic operators. In this section, we introduce the basic elements of the language.
This alternative is one of the primary alternative logical representations of the deontic concepts. The reason we introduce it is because the main topic of the thesis is the explicit introduction of articulated violation and fulfillment markers for deontic reasoning. Thus, the introduction of KAR ties our proposal to a significant historical strand of research. Our proposal is a refinement of KAR, though in a Dynamic Logic.

As our purpose is to introduce KAR, we do not discuss it further in much of the body of the thesis, but focus on SDL, which appears to be a far more commonly researched formalism. Therefore, most of the issues we discuss in the following are cast in terms of SDL. Yet, as SDL is contained in the reduction ([128]), what we say about the deontic concepts in terms of SDL apply as well to the reduction.

### 2.3.2.1 Syntax

Assume that we have a language of modal Propositional Logic, with a distinguished propositional constant. There are two versions here: [102] introduces the constant $D$, meaning all (relevant) normative demands are met, while [7] and [6] introduce the constant $V$ meaning some normative demand has not been met. While the propositions can be viewed as interdefinable ($V =_{df} \neg D$), we prefer the latter, for violation is discussed in ScDL and it does not require universal quantification over norms.

**Definition 5 (System KV – Syntax)**

- **PC WWFs:** All valid, well-formed formulae of the Propositional Calculus are theorems of KV.
- **K:** $\vdash \square (P \rightarrow Q) \rightarrow (\square P \rightarrow \square Q)$
- **V:** $\vdash \neg \square V$
- **Modus Ponens:** If $\vdash P$ and $\vdash P \rightarrow Q$ then $\vdash Q$
- **NEC:** If $\vdash P$ then $\vdash \square P$

KV is just the normal modal logic K with $V$, which tells us that $V$ does not hold in every world.

The deontic concept for obligation is then expressed in terms of this logic.
2.3 State-wise Deontic Logic

Definition 6 (OB in KV)

\[ OB(P) =_{df} \Box (\neg P \rightarrow V) \]

KV has all the theorems of SDL as well as some specific to it because it has the additional syntactic ingredients of the designated violation proposition \( V \), \( \Box \), and \( \Diamond \). However, these issues are not relevant to our central concerns.

2.3.2.2 Semantics

The Kripke-style semantics for KV are similar to those above except that we have a designated set of worlds \( \text{VIO} \). A model \( M = (W, R, V, \text{VIO}) \), \( W \) is a non-empty set of worlds, \( R \) is a binary relation on worlds, \( V \) is the valuation function, and \( \text{VIO} \) is the set of worlds in which some normative requirement is violated. The accessibility relation \( R \) has no interpretation as deontic alternative, but is the accessibility relation of Alethic Modal Logic where \( \forall i \exists j (iRj \land j \in \text{VIO}) \).

We have the usual clauses for Alethic Modal Logic to which we add:

Definition 7 (System KV – Semantics for OB)

\[ M \models_i V \iff i \in \text{VIO}. \]

Models for \( OB(P) \)

\[ M \models_i OB(P) : \forall j[if iRj \land M \models_j \neg P, then j \in \text{VIO}] \]

Note that worlds which are \( \text{VIO} \) worlds are also worlds in which \( V \) holds. As we want to focus on the notion of obligation, we omit further discussion of truth-conditions for permission and prohibition.

As we discuss in chapter 4, violation and/or fulfillment markers were initially introduced with [102], [7] and [6], then subsequently used, with variants, in [106], [130], [105], [186], [187], [162], [99], [30], [46], [132], [174], [62], and [29]. Our proposal is a justification, refinement, and implementation of this approach.

We have provided a brief sketch of key elements of two variants of the syntax and semantics of SwDL.
In this section, we discuss some of the problems of SwDL, focussing on the SDL formalism, which is largely the context in which the problems are discussed. [128] presents a catalogue of problems of SwDL, some bearing on the axioms of SwDL, some with theorems, and some bearing on additional issues. To a great extent, the issues arise for two reasons. First, it is presumed that deontic operators are analogous to the alethic operators, which obscure those aspects which differentiate them. Second, SwDL is usually couched in terms of Propositional Logic such that the deontic operators apply to any well-formed expression of Propositional Logic. Along these lines, no additional mechanisms are allowed which might otherwise avoid the problems. Where neither of these points hold, problems arise. As we have said earlier, these criticisms can be taken to have little consequence on SwDL as a formal theory in and of itself; in particular, one may have no objections where the propositions that the deontic operators apply to are from a formal domain such as mathematics ([67] and [1]).

We do not rehearse all problems, issues, and alternatives from the literature. Rather, we focus on only some of them in order to show that that SwDL is not a suitable formal theory of common sense deontic reasoning as indicated and supported by natural language examples. Furthermore, we make decisions with respect to these issues, supporting them as we can. We do not claim to have provided a definitive, logical solution to every problem, but rather discuss several issues and how we address them relative to linguistic analysis or reasoning based on natural language. In particular, we show that there are alternative solutions to the problems based on independently argued for linguistic observations and theory. Such observations provide a rationale for why SwDL finds little application in domains where natural language and common sense reasoning is at the fore, as in legal contracting; the less the logical formalism represents natural language forms and common sense reasoning, the less suitable it is. Nonetheless, consideration of SwDL provides an opportunity to identify elements and issues that we incorporate, avoid, or have as options in the development of our tool.

We discuss the following topics in some depth. First, we discuss the lexical semantics of obligation in order to clarify the particular interpretation of the term that is of interest to us in legal reasoning. Then we discuss three particular structures: conditionals and the obligation operator, the Gentle Murderer Paradox, and then the Good Samaritan Paradox. We show that that there are substantive syntactic and semantic issues that must be addressed before one makes claims about the logical analysis of the deontic operators. Moreover, that in order to better formally represent contracts, we ought to be guided by
well-formedness conditions as found in the language in which legal contracts are cast. This is discussed in the following sections.

Following these, we more briefly touch on a spectrum of topics that are relevant to general considerations of the deontic concepts as well as our implementation. These are brief topics, not because they are not intrinsically of interest and worthy of a whole thesis, but because we must focus our discussion on our goal. Here we discuss whether conflicting obligations can hold, obligations and deadlines, obligations and antonyms, the relation between obligations and permissions, and some comments on the relevance of Deontic Logic to the formal representation of contracts.

Running through several of these discussions is the issue of whether OB-RM holds or not of OB; we provide a discussion specifically on this matter. We are neutral on whether OB-RM holds of OB or not since none of the following discussion is a matter either of denying OB-RM, per se, or affirming defeasible approaches ([80] and [141]). Rather, in several cases, we provide alternative logical forms with respect to which OB-RM is not at issue. As we see, these alternative logical forms arise because we take into consideration grammatical and semantic interactions between the operator and the proposition which have not previously been taken into account.

As some of the following discussion is based on linguistic analysis, we recall section 1.3.4, where these were first introduced. As our approach to linguistic analysis is based on intuitions of well-formedness, a remark on the practice of linguists is in order ([37], [38], [39], [87], [152], [150], [45], [35], [36], [91], [118], [56], and [117]). Linguists are trained in the analysis of and debate about natural language. The literature previously cited contains a wealth of data and analysis as well as references for further research. We have drawn on our knowledge of this literature and experience in research in providing our analysis and judgements. However, as previously mentioned, we cannot provide a tutorial or a revision of contemporary linguistics. A standard practice in this approach to the analysis of language is to take well-known structural “frames”, to substitute in items of interest, to make judgements of well-formedness, and then to check these judgements with colleagues equally qualified to evaluate them. There are other approaches to linguistic analysis, and no approach is “perfect”; each has strengths and weaknesses. However, in our view, such judgements are better than those founded on less knowledge of and experience in the analysis of language. Therefore, in the following, our evaluations of the well-formedness of expressions is based on the standard practice just mentioned. Where relevant, we cite references to background literature.
One other preliminary remark is worth making as well. In linguistic analysis, the well-formedness and interpretation of natural language expressions are used to guide logical forms and not vice versa ([35], [36], [91], [56], [117]). If natural language expressions are found to be ill-formed, semantically anomalous, or to have one interpretation but not another, then the logical form, which represents the form of the meaning of the expressions, should not be well-formed either. A significant amount of research in linguistic theory is devoted to such a tight relationship between the natural language forms and their logical forms. Our approach here follows in this line of research.

2.5 The Lexical Semantics of Obligation

In this section, we point out that there are different interpretations of lexical items such as ought and obliged, which express the concept of a norm. As the lexical items and the notion of norm are ambiguous, it is necessary not only to identify which sense we work with and to provide concrete evidence for the distinction, but also to use that sense consistently in any subsequent expression. Not to do so consistently risks conflating issues.

For our purposes, in the formal representation of contracts, we focus on that sense which implies violation or fulfillment. The other generic interpretation, which admits of exceptions, is relevant, but not distinctive of legal reasoning. To avoid the ambiguity, we focus on terms which are less ambiguous. Thus, this section is an exercise in filtering out alternative senses to focus on that which we find most relevant for our purposes. We discuss central observations, making no claim that we have exhausted the issues.

We have several subsections to make our points clear. We first discuss a variety of norms that do not relate to violation. To facilitate discussion, we briefly outline some morphosyntactic issues, which allow us to put aside conflating issues. Having established that there are two different interpretations and their forms, we present a range of evidence and reasoning to show just the extent of the differences.

2.5.1 Background in Lexical Semantics

In Linguistics, lexical semantics is that area that focuses on the syntactic contexts and meanings of particular words as well as the words in an ontology [45], [150], [119]). Among the issues, we find that one word form may have different and unrelated meanings (e.g. bank as in river bank and financial institution) as well as different and related meanings (e.g. bank as in place in I went to the bank or as financial institution as in The bank
wound bankrupt). We also find that one sentence meaning can be represented with two alternative sentence forms and correlated lexical items (e.g. the active Bill kissed Jill and passive Jill was kissed by Bill). Furthermore, one sentence form can have two alternative meanings with the same lexical items (e.g. the alternative interpretations of Every man loves a woman). We can find all these issues bearing on our understanding of the meaning of a deontic concept. Therefore, it is rather important to be clear about which meaning we are using and the alternative forms that can take, for otherwise we are liable to use one meaning as an argument concerning another (intended) meaning. While the lexical semantics of the deontic concepts is a worthwhile research topic in and of itself, we shall only briefly outline some of the relevant differences and highlight that which we take to be central to our concerns.

2.5.2 Epistemic and Non-epistemic Interpretations

The logical and linguistic literature discusses examples, interpretations, and classifications of the modal auxiliaries such as ought ([103, p.121], [147, p.21,36], [183], [181], [98, p.13], [97], [180], [163], among others). A range of distinct interpretations have been expressed: directive, evaluative, prescriptive, non-epistemic, epistemic, and others. Linguists primarily distinguish between epistemic modals and non-epistemic modals ([96], [110], [111], [26]); these are auxiliary verbs such as must, ought and may which have an interpretation concerning degrees of knowledge and belief (so the epistemic interpretation) and another where violation is relevant (so the deontic or non-epistemic interpretation). We use this terminology in the thesis to be consistent with this literature; however, other terminology appears in computer science. The epistemic relates to a notion of regularities or presumptive expectations, while the second relates to a system of sanctions and rewards which can be specified even for novel, rare, or unanticipated states or actions. One way to distinguish the senses is relative to background context, where epistemic modals relate to contexts of what is known or believed, while the non-epistemic modals relate to what is ideal, much along the lines of SDL, though in a different formal semantics ([110]). As we are primarily interested in the non-epistemic interpretation, we discuss in this section ways to differentiate it from the epistemic. To make these matters clear, we break down the issues.
2.5.3 Generics, Habituals, and Progressives

To clarify the epistemic interpretation, consider cases without a deontic operator but with a closely related semantic interpretation. This highlights one interpretation of norm as regularity.

**Example 1**

\(a.\) Birds fly.

\(b.\) Bill smokes.

\(c.\) Bill is crossing the road.

The first is a generic statement, the second an habitual, and the third a progressive ([143] and [116]). In one sense or another, they all relate to what usually holds or a usual course of events, which allows for exceptions. Thus, they can be understood as norms: Example (1a) requires that most birds do fly, though there are some that do not; Example (1b) requires that Bill smokes on enough occasions (which is vague) to assert that he is a habitual smoker, though there are many moments of time in which he does not smoke; Example (1c) is true where Bill is following a path such that, were he to complete it, he would have crossed the road, though should he be struck dead in the middle of the road, he would not have crossed it. Note that the simple present tense in these examples (and sometimes other tenses) can give rise to a generic or habitual interpretation ([41]).

We should be careful not attribute such an interpretation to some other sentence element which is properly attributable to the tense. We could say of these examples that the exceptional individuals, times, or circumstances are somehow violations of or deviations from the norms. These expressions are much like a norm Bill ought to leave in that there is an interpretation under which one has an expectation that, in usual circumstances, Bill leaves, but there may be an exceptional circumstance in which our expectation is not satisfied.

However, we point out that we cannot say that these are legislated or contractual norms which hold in virtue of some party stipulating that the norm holds where it previously did not. For example, we cannot point to some class of natural objects such as worms which have never cried, but by saying worms cry make it so. Nor can we point to some individual Phil who has never smoked and, by ascribing to him Phil smokes, make it so. By the same token, it is not reasonable to suggest that the individuals, times, or circumstances which do signal something outside the purview of the norm have thereby themselves violated it.
and must bear the consequences. Nor for that matter must there be any consequences to make sense of the norm and its exception: it makes little sense to punish a penguin for being a non-flying bird. Thus, the norms in Example (1) are, in our view, more along the lines of epistemic norms, which are not the sort of norms that we are concerned with in this thesis. We do not want to conflate the epistemic with the non-epistemic. Rather, we want norms which can be contractually established, can be novel (so not relating to presumptive expectations), can be violated (where there can reasonably be consequences which follow from a violation), and where there is a party who somehow suffers the consequence of the violation of the norm.

2.5.4 Morphologically and Lexically Related Words

Another issue to clarify in drawing out the relevant interpretation is to distinguish among several morphologically or semantically related forms, for we want to use those forms which best serve to make our point, rather than to obscure matters or reintroduce ambiguities. These observations are familiar from morpho-syntactic and lexical semantic discussions in linguistics.

Example 2

a. Obligatorily, Bill visits Jill.

b. It is obligatory that Bill visits Jill.

c. Bill is obligated to visit Jill.

d. Bill is obliged to visit Jill.

e. It ought to be that Bill visits Jill.

f. Bill ought to visit Jill.

g. Bill must visit Jill.

Let us generalize and say there is the obliged group, indicated with OB, in Examples (2a)-(2d) and the ought group, indicated with OU, in Examples (2e)-(2f). We will leave aside Example (2g) with must, which has syntactic and semantic properites similar to ought. It is not clear that the differences between the obliged group are significant for our purposes.
There is much one can say about the syntactic differences between the forms and correlated semantic interpretations, which are very similar to other sentential adverbs with alternative morphological forms of *reluctantly* and *stupidly* ([189]).

**Example 3**

\[ a. \text{Reluctantly, Bill visited Jill.} \]
\[ b. \text{Bill was reluctant about visiting Jill.} \]
\[ c. \text{Stupidly, Bill visited Jill.} \]
\[ d. \text{It was stupid of Bill to have visited Jill.} \]

However, we will overlook this element of the discussion for the time being and assume that despite the different surface syntactic forms, they have the same underlying logical form. The alternative forms may be derivationally related.

Leaving aside syntactic and semantic tests used to distinguish sentential from predicational operators (e.g. sentence adverbs versus manner adverbs discussed in [170] and [189]), we take the underlying logical forms to be comprised of a propositional operator applied to a proposition. As we want to distinguish epistemic from non-epistemic interpretations of the operators, we use \( \text{OU} \) to represent the epistemic interpretation and \( \text{OB} \) to represent the non-epistemic interpretation of *ought* or of *obliged*:

**Example 4**

\[ a. \text{OU}(P) \]
\[ b. \text{OB}(P) \]

The main reason to make this distinction is that while *ought* seems to be ambiguous, *obligatory* is not; we use the contrast between them to highlight the sense we are primarily interested in.

**2.5.5 Evidence to Distinguish Interpretations**

In the following, we provide a series of examples which speakers report systematically distinguish between the *ought* and *obliged* interpretations ([96], [110], [111], [26]). While the
examples and discussion are novel, they broadly follow the linguistic methodology of *sorting forms and interpretations according to collocational restrictions, which are systematic patterns of cooccurrence.* Note that while we provide particular examples, these are representative of classes of examples, so supporting our claim to have provided a systematic characterisation. In the following, # indicates intuitive semantic or syntactic unacceptability. It is common practice in linguistic research to compare and contrast well-formed and ill-formed expressions, using the observations to infer systematic syntactic and semantic properties. Judgements can be sharpened with contexts. The data can be *objectified* with *psycholinguistic* data-gathering tasks and analysis ([77]). As and where needed, we articulate the contexts which support alternative interpretations. In the following sections, we start with general considerations, then focus on agents, predicates, and derivationally related forms.

### 2.5.5.1 Comparing Epistemic and Non-epistemic Interpretations

In the following, we can identify distinct primary meanings between the groups. We see there is something distinctly odd or ungrammatical about the (a) examples in contrast to the (b) examples. We have chosen the predicates *rain, weigh, and add up* because they have interpretations which need not be either agentive or causative; that is, while someone (i.e. the agent) can make (i.e. cause) a stone weigh 10 kilos (say by carving it), we consider the non-agentive, non-causative contexts ([45], [150], [119]).

**Example 5**

a. # *It is obligatory that it rains.*

b. *It ought to be that it rains.*

**Example 6**

a. # *It is obligatory that the stone weighs 10 kilos.*

b. *It ought to be that the stone weighs 10 kilos.*
Example 7

a. # It is obligatory that the numbers add up to 9.

b. It ought to be that the numbers add up to 9.

In these examples, the *ought* expressions primarily imply a high likelihood or expectation that something will hold, but if it does not, it need not be that consequences such as sanction or violation follow. They do not imply that an external authority has imposed something on an overt or implied agent; in fact, there need not be (or cannot be) any such authority or agent. Where there can be no agent or authority, as in the case with rain, it seems distinctly odd to impose an obligation. The only way to do it is to introduce some abstract agent or authority, such as a deity, though there need not be one. Finally, as the (b) sentences are well-formed and easy to interpret, there is no need to provide some explanation to accommodate them.

On the other hand, in the (b) examples with *obliged*, there need not be an assertion about the likelihood of something occurring or holding. Indeed, an obligation can be placed on something which is of a very rare occurrence as in *Bill is the first and only person to be obliged to walk on a bed of coals in Westminster Abbey*. Yet, if the occurrence is not realized, a violation follows. It suggests an external authority has imposed the obligation on an overt or implied agent. And finally, the oddness of the expressions requires some explanation to accommodate them, which is the provision of some context in which the statement makes sense. For instance, a Mafia don might threateningly order someone to sum 4 plus 4 to 9, even though fulfilling this order is impossible.

Matters are clearer when we use our operator as a modal verb, predicating directly of a subject. This appears to remove any possibility of introducing an agent which makes the property hold. In the following cases, the likelihood interpretation of *ought* predominates. The cases with *obliged* are clearly very awkward, perhaps ungrammatical for semantic reasons.

Example 8

a. # It is obliged to rain.

b. It ought to rain.
Example 9

a. # The stone is obliged to weigh 10 kilos.

b. The stone ought to weigh 10 kilos.

Example 10

a. # The numbers are obliged to add up to 9.

b. The numbers ought to add up to 9.

If one is unclear about the interpretation of a sentence where the operator is a sentential operator as in Examples (5)-(7), one can try similar cases with the modal verb along the lines of Examples (8)-(10).

Clearly, ought can, in the right circumstances, have the interpretation we associate with obliged. In (11b) we may have a clearer interpretation of obligation rather than likelihood as there is an overt agent who can bear the obligation.

Example 11

a. It ought to be that Bill leaves.

b. Bill ought to leave.

This illustrates that there are subtle interplays between the meaning of the operator, the subject, and the particular predicate. We can make this clearer by focussing on properties of the subject and the predicate.

In the following, the only sensible interpretation of (12a) is where Bill is not yet dead, but is about to be (say in the context of a mafia hit). (12b) need not have this interpretation, but can report an expectation that he is already dead, which is in contrast to the fact that he is not dead.
Example 12

a. It is obligatory that Bill be dead, though he is not yet dead.

b. # It ought to be that Bill is dead, though he is not yet dead.

The difference here indicates that obligated imposes some notion of state change which is not needed for ought. The most natural interpretation of (12a) is where at the moment of utterance, Bill is not yet dead, and that at some future time there is a state at which he is dead so as to fulfill the obligation. This interpretation appears to be a presupposition which can be defeated, but it would seem rather strong. (12b) seems to introduce the opposite presupposition, namely, that Bill is dead at the moment of utterance.

We reinforce these observations with the following. Even though in principle Bill could be an agent who bears an obligation, he cannot be when he is dead. It would be rather odd, though not unthinkable, to impose on Bill the obligation that he brings it about that he is dead – say for some reason Bill has to kill himself. Perhaps part of the oddity here is that Bill would not be the beneficiary of having fulfilled the obligation as is often the case. No such strenuous interpretations are needed for the case with ought.

Example 13

a. # Bill is obligated to be dead.

b. Bill ought to be dead.

2.5.5.2 Agentive Subjects

To sharpen the distinction between epistemic and non-epistemic interpretations, we can point to a variety of selection restrictions on the subject. For instance, the non-epistemic interpretation requires subjects which are highly agentive. [65] provides a semantic analysis of agentivity in natural language, arguing that agentivity should be decomposed into a set of more basic properties and that agentivity is a prototypical concept, whereby those arguments with more of the basic properties are understood to be more agentive than those with fewer. We provide the typical examples of these agentive properties, where we say that the subject bears the property. We contrast nouns which can be or are intrinsically thought to be highly agentive with nouns that are not thought to be agentive.
Example 14

a. Bill is being polite to Jill.
   # The cloud is being polite to Jill.
   **Volition**

b. Bill knows/believes/is disappointed at the statement.
   Bill sees/fears Jill.
   # The cloud fears Jill.
   **Sentience/Perception**

c. Bill’s loneliness causes his unhappiness.
   Bill made the glass break.
   # The cloud made the glass break.
   **Causation**

d. The rolling tumbleweed passed the rock.
   Bill left the room.
   # The ground left the room.
   **Independent Movement**

In our Examples (8)-(10), (11), (13), we can differentiate the well-formed from ill-formed cases with respect to prototypical agentivity, where *obliged* selects for highly agentive subjects while *ought* is relatively less selective.

In Example (15a), which is marginally acceptable, the subject *the roast* is low in agentive properties; the only way to accommodate the statement is to infer that there is some individual who is an agent somehow bears the obligation that the roast cannot. In other words, we would paraphrase it as *Someone is obliged to have had the roast ready by now.* In contrast, Example (15b) is well-formed and needs no such inference or paraphrase to accommodate it ([98, p.13], [103, p.121], and [147, pp.21, 36]).

Example 15

a. *The roast is obliged to be ready by now.*

b. *The roast ought to be ready by now.*
2.5.5.3 Predicate Restrictions

We have seen selection restrictions on the sorts of subjects which can bear an obligation in the non-epistemic sense. We can see that given the same agentive subject, there are also selection restrictions on the sorts of predicates. In Examples (16), we see that the non-epistemic interpretation is incompatible with predicates of psychological attitude or perception, while the epistemic interpretation in Examples (17) is not as the sentences in Examples (16) are semantically anomalous, while those in Examples (17) are not.

Example 16

a. # Bill is obliged to believe in God.

b. # Bill is obliged to realize the answer.

c. # Bill is obliged to see red

Example 17

a. Bill ought to believe in God.

b. Bill ought to realize the answer.

c. Bill ought to see red.

Example (16a) is only acceptable as a statement of religious doctrine, and even in this case it is problematic. One can be obliged to behave or speak in a way which others take as consistent with a belief in God, but it seems unreasonable to investigate whether Bill truly fulfills or violates the obligation, for Bill has privileged access to his attitudes. Along these lines, so far as we know, there are no legal requirements with respect to a psychological attitude or a perception of any sort, though there may well be expectations on such an attitude or perception. In other words, while there are laws which prohibit murder, and an intent to murder may play a role in a conviction, there are no (current, rational) laws which prohibit the intent to murder itself; this would be tantamount to thought-policing.

Another selection restriction is that the non-epistemic operator cannot appear with individual-level predicates as in Example (18a), but can appear with state-level predicates as in Example (19a). The epistemic operator has no such restriction. Individual-level predicates
are those which hold constantly of an individual over an extended period of time, while state-level predicates are those which hold intermittently and over narrower extents of time ([112]).

Example 18

a. # Bill is obliged to be tall.
b. Bill ought to be tall.

Example 19

a. Bill is obliged to be available.
b. Bill ought to be available.

Notice that our observations on selection restrictions for *obliged* indicates that OB-NEC cannot hold for *obliged* since the axiom applies indifferently to any propositional theorem. However, since *ought* appears not to have the same selection restrictions, OB-NEC may apply to it.

2.5.5.4 Derivational Relationships and Impersonal Obligations

Having identified some of the selection restrictions on our target sense of *obligation* and norm, we comment on the relationships between forms of the expression. We have used expressions of the form:

Example 20

a. It is obligatory that Bill leave.
b. Bill is obliged to leave.

For our purposes, we regard them as *synonymous* in the sense that both are equivalent to the same logical form, where the sentential operator OB applies to the meaning Bill leave. In the literature ([113], [129], and [127]), the relationship between these forms is discussed in terms of a distinction between *ought-to-do* and *ought-to-be*, personal and impersonal
obligation, and the so-called brings-it-about operator. This is referred to as the Meinong-Chisholm Reduction where Jane Doe is obligated to bring it about that p if and only if it is obligatory that Jane Doe brings it about that p. It is sometimes understood as a reduction of personal obligation in Example (20b) to impersonal obligation and agency in Example (20a). In [191], we claim that ought-to-be is implicitly an action, one where the agent is obligated to act so as to maintain some property. As we are primarily interested in a dynamic theory of action, we do not use the brings-it-about operator. Given our discussion above, the obliged concept appears to require an agent, so is always personal.

This leaves somewhat open the issue of what is a so-called impersonal obligation ([127]). The discussion in the literature is not set against general linguistic considerations. Here we just touch on some relevant issues, but leave them for future study, as they are not central to our investigation.

There are various ways to construe the notion impersonal, and it is not clear what is being referred to or how it is being used in the deontic logic literature. In Example (21a), the subject it is called ([37]) a pleonastic subject which serves only a grammatical role, but is contentless. In Example (21b), we have an indefinite (i.e. existentially quantified) subject; the sentence is true where someone left, but we do not know who. In Example (21c), the agent of the kissing has been argued to be semantically implicit ([66]). Finally in (21d), we have an unaccusative verb break in which the subject semantically plays the role of the object, so leaving open what is the subject ([120]). It is, then, rather unclear exactly what is intended by the term impersonal and how it is used in the analysis.

**Example 21**

a. It rained.

b. Someone left.

c. Jill was kissed.

d. The vase broke.

Furthermore, the syntactic and semantic relationship of the statements in Example (20) appear to be closely related to so-called raising verbs in Examples (22a) and (22b). More broadly, the issue bears on the syntactic movement of arguments to yield synonymous interpretations.
Example 22

a. *Jane Doe seems to have left.*

b. *It seems that Jane Doe has left.*

While there are a range of issues which bear on the motivation of the distinction as well as theoretical issues, at least in the tradition of grammatical analysis following on from Transformational Syntax ([37] and [87]), there is but one lexical item in such structures. The alternative forms are given by other mechanisms of the grammar. This is in contrast to the proposals where one interpretation is “reduced” to another.

For our purposes, we can remain neutral about these issues for we do not, in the course of the thesis, subscribe to some of the other components described above, but provide an analysis into which either form is expressed, the differences arising perhaps by grammatical mechanisms.

In this section, we have examined the lexical semantics of obligation in order to focus our discussion on an interpretation that is relevant to legal contracting. We have shown that “normative” interpretations are not all alike and have fixed on that interpretation which requires an agent, selects agentive predicates, and implies violation should that which is obligated not hold. It is in contrast to other work in Deontic Logic ([128] and [30]) or on norms ([123] and [10]) which does not take into consideration the linguistic properties of the operators which underlie their formal analyses.

2.6 Conditionals and Obligations

In this section, we discuss conditionals and obligations, which are critical to understand the *Contrary-to-Duties Paradox*. In the formal syntactic and semantic literature, ([67], [1], [57], [88], [89], among others), a range of empirical issues and theoretical approaches concerning the conditional are discussed. The empirical issues are richer and more complex than represented in classical Propositional Logic. There appears to be little theoretical consensus other than to claim that the classical syntax and semantics of the material conditional does not correctly account for the range of natural language expressions. Each of the theories is problematic in terms of how it is used in natural reasoning. It is not our objective here to review all these approaches or data in depth, which is beyond the scope of this work. Rather, we focus narrowly on topics relevant to our current investigation.
We first outline the issues, then discuss two particular issues. We make two contributions. We draw attention to the relevance of the linguistic analysis of the interactions between modals and conditions to discussion of conditional obligations in the deontic logic literature. Second, we focus attention on the non-epistemic interpretation, which is not discussed in depth in the linguistic literature ([57]).

One of the key features of SwDL is that the deontic concepts are normal modal operators, which is underwritten by Axiom OB-K. In this section, we focus on Axiom OB-K and leave related issues OB-RM to section 2.9.1. Axiom OB-K is comprised of the expressions of the form OB(P → Q) and [OB(P) → OB(Q)], where OB(P → Q) is the antecedent and [OB(P) → OB(Q)] is the consequent. In the following subsections, we discuss each of these expressions separately, the latter first. Whether the axiom holds or not is relevant to the analysis of CTD problem. We summarise our key points.

While Axiom OB-K may be applicable in mathematical or other formal domains, there is linguistic evidence that both the antecedent and consequent of the axiom are grammatically ill-formed. [OB(P) → OB(Q)] is syntactically ill-formed as it is claimed in the literature which we review that natural language conditionals do not allow certain sorts of modal operators in the antecedent of a conditional. We extend this point to the deontic operators, claiming that [OB(P) → OB(Q)] is ill-formed as the obligation operator cannot appear in the antecedent as OB(P). In addition, OB(P → Q) is ill-formed given our discussion in section 2.5 since (P → Q) is the wrong sort of expression for OB to apply to and there is no agent. However, [P → OB(Q)] is a well-formed expression in natural language. In general, unlike the syntactic well-formedness rules of Propositional Logic, natural language imposes additional constraints on the distribution of deontic operators. Thus, the natural language correlate to Axiom OB-K is ill-formed. This is not a logical problem per se, but rather that the logic does not accurately model the syntax and semantics of the natural language expressions.

However, there are important subtleties to present. In particular, there appear to be differences between the epistemic and non-epistemic interpretations which bear on the well-formedness and interpretations of OB(P → Q) and [OB(P) → OB(Q)]. We discuss these in the following sections.
2.6.1  $[\text{OB}(P) \rightarrow \text{OB}(Q)]$

Consider Examples (23a) and (23b), assuming they both have the logical forms in Examples (23c) and (23d). Here we use the modal auxiliary verb *ought* for our deontic operator; we shall discuss the relationship between *ought* and *obligatory* in a subsequent section.

**Example 23**

a. *If you ought to leave tomorrow, then we ought have a drink tonight.*

b. *If obligatorily you leave tomorrow, then obligatorily we drink tonight.*

c. $[\text{OU}(P) \rightarrow \text{OU}(Q)]$

d. $[\text{OB}(P) \rightarrow \text{OB}(Q)]$

There is a substantive body of linguistic research which claims that expressions of the logical form in Examples (23c) and (23d) are not well-formed expressions of natural language ([57], [88], [89]).

The main observation relevant to our discussion is that there appear to be restrictions on the occurrence of epistemic and non-epistemic modals in the antecedent of a conditional, though not in the consequent. Given that modal logic does not, in general, distinguish among *sorts* of propositions, much less the application of operators relative to syntactic position, this distinction cannot be made. Indeed, the phenomenon is more general than the antecedent of conditionals, for there are a range of restrictions on the occurrence of modal operators in the subordinate adverbial clauses, of which the antecedent is but one subclass ([88], [89], [57]). Finally, these observations are attested *cross-linguistically*, so are not incidental to English grammar.

2.6.1.1  **Linguistic Evidence**

We briefly provide some of the evidence from the literature which supports the claim that epistemic or non-epistemic modals do not appear grammatically in subordinate adverbial clauses. Note that we claim that these statements are ungrammatical where there is no prior utterance; we discuss this issue further below.
Example 24

a. # If John must have time on his own, he will do it.
b. # Before John must have tampered with the tapes, we met him.

Similarly, speech-act adverbs frankly and evaluative adverbs unfortunately do not appear in subordinate adverbial clauses. This indicates that we are seeing a general linguistic phenomenon, not one tied specifically to the deontic operators.

Example 25

a. # If frankly John cannot cope, we will have to fire him.
b. # If unfortunately we don’t find our dog, we’ll call the RSPC.

By the same token, the following also appears to be ill-formed:

Example 26

# If obligatorily you leave tomorrow, then let’s drink tonight.

For us, the intuitions for the judgements are clear for Examples (24)-(26).

2.6.1.2 An Analysis

Exactly how to account for this phenomenon is open; a formal analysis has yet to be provided. [57] claims that issues of speaker subjectivity and echoic speech bear on a solution, where speaker subjectivity relates to the degree of belief the speaker has in the utterance and echoic speech refers to an echoing of a previous utterance in a current utterance. For instance, take an example, where one speaker says Who likes Bill? and this is followed by another speaker who says Who likes Bill, indeed? The second occurrence of Who likes Bill is echoic and does not mean exactly the same as the first occurrence. While the first occurrence can be followed by a list of people who like Bill, the second implies that no one likes Bill.

The observation is that expressions with epistemic or non-epistemic modals, as in the examples in (23), cannot be asserted out of context, being true or false relative to one speaker. However, in a dialogue context, they can be used echoically:
Example 27

Speaker A: John must have time on his own.

Speaker B: If John must have time on his own, then he will do it.

Example 28

Speaker A: Obligatorily, I leave tomorrow.

Speaker B: If obligatorily you leave tomorrow, then let’s drink tonight.

In both examples, the antecedent of Speaker B’s sentence appears to be granting what Speaker A says holds and just repeats it; it does not appear to be expressing an obligation. The content of the antecedent could just as well be expressed by expressions such as I agree! or Whatever you say!, followed by the content of the consequent as an independent clause. For Example (28), we could have:

Example 29

Speaker A: Obligatorily, I leave tomorrow.

Speaker B: Whatever you say! Then let’s drink tonight!

2.6.1.3 Additional Considerations

While the intuitions for the cases so far are clear, they may conflate issues. Observe that a paraphrase of the epistemic modal in Example (23a) as in Example (30a) is still unacceptable, while the paraphrase of the non-epistemic deontic operator in Example (23b) as in Example (30b) is more acceptable.

Example 30

a. # If is is usually the case that you leave tomorrow, then it is usually the case that we have a drink tonight.

b. If it is obligatory you leave tomorrow, then it is obligatory that we have a drink tonight.
The acceptability of Example (30b) can be accounted for following the analysis of section 2.6.1.2. It is a relationship between two obligations, stating that where one holds, the other must also hold. However, this is not a purely logical relationship since context is crucial. For contracting, this is relevant in terms of a well-defined contract, where the contract provides the context; that is, the contract itself is under discussion.

2.6.1.4 Restricted Distributions

While there are reported restrictions on the distribution of a range of modal operators in the antecedent of conditionals, there are no known restrictions on the distribution of the modal operators in the consequent of conditionals. In light of this, for OU, expressions of the syntactic and semantic form of Examples (31a) and (31b) are ill-formed, but Example (31c) is well-formed.

Example 31

\[ OU(P) \rightarrow OU(Q) \]
\[ OU(P) \rightarrow Q \]
\[ P \rightarrow OU(Q) \]

For OB, (32a) and (32b) are also ill-formed, and Example (32c) is well-formed in general. However, we keep in mind that in particular circumstances discussed in section 2.6.1.3. This does not detract from our general claim concerning ill-formed expressions with respect to SwDL, which has no representation of the relevant contexts.

Example 32

\[ OB(P) \rightarrow OB(Q) \]
\[ OB(P) \rightarrow Q \]
\[ P \rightarrow OB(Q) \]

2.6.2 The Wide-Scope Operator

Finally, let us consider Example (33).
Example 33

*It is obligatory that if you leave tomorrow, then we have drink tonight.*

We consider a formalisation first with the deontic operator and material conditional, then the operator with the *Dyadic Conditional*.

### 2.6.2.1 \( OB[P \rightarrow Q] \)

We first consider Example (33) translated as Example (34).

Example 34

\[ OB[P \rightarrow Q] \]

Following our discussion in section 2.5, Example (34) cannot be the logical form for Example (33) since it is ill-formed -- the conditional expression is not of the right sort for *obligation* to apply to it and there is no agent. Moreover, speakers do *not* report interpreting \( OB[P, then Q] \) as an *obligation* on a conditional, but rather interpret it as an alternative form of \( [if P, then OB(Q)] \).

However, along the lines as discussed in section 2.6.1.3, there may be an interpretation of Example (33) with respect to well-defined contracts: for example, it may be understood as a stipulation that a contract include the rule \( P \rightarrow Q \) or otherwise the contract is ill-formed.

Even under the interpretation as contractual restrictions, Example (34) does not imply Example (32a) since what is obligatory varies in each case. Indeed, this case is one example where the non-epistemic interpretation of *obligation* is sensitive to the *logical structure* of the expression it applies to and not simply to truth values of propositions.

### 2.6.2.2 \( OB(P/Q) \)

An alternative logical form is the so-called *Dyadic Conditional* for the deontic operators which have the following form ([30]).
Example 35

\[ \text{OB}(P/Q), \text{ read as “It is obligatory that Q, given P.”} \]

This is primarily deployed to avoid some of the paradoxes of deontic logic that other forms are susceptible to; for example, given \( \neg P \), we can infer \( P \rightarrow \text{OB}(Q) \). We discuss some of these paradoxes in subsequent sections. In our view, the logical form \( \text{OB}(P/Q) \) is linguistically unmotivated. [24] rejects it, claiming it simply avoids commitments concerning the relation of the deontic operator to the conditional expression.

As we have said at the outset, it is not our objective to provide a definitive discussion of conditionals and modal operators, but to argue for more realistic semantic representations that can be useful for the representation of contracts.

2.7 Deontic Operators and Adverbs

In this section, we consider how the deontic operators behave with respect to adverbial modification ([189], [190], [195] for a discussion of the syntax and semantics of adverbial modification). This is a relevant example to discuss as it bears on the so-called simultaneous CTD structures, which are also known as the Gentle Murderer Paradox.

We outline the issue and two potential solutions – deny \( \text{OB-RM} \) or deny \( \text{OB-D} \), which we discuss later in this chapter. In addition, we propose an alternative solution based on focus which does not depend on a position with respect to \( \text{OB-RM} \) or \( \text{OB-D} \).

2.7.1 The Gentle Murderer Paradox

The Gentle Murderer Paradox has the following structure. We assume Examples (36a)-(36d) from which we can infer Example (36e).\(^2\)

\[^2\text{We have added in the implicit object of killing, though this is implicit in some examples in the literature. The examples throughout would support a better discussion were they along the lines of “Bill kissed Jill on the cheek”. We have kept to the “kill” examples to be more consistent with discussions in the literature.}\]
Example 36

a. Bill is obligated not to kill Phil.

b. If Bill kills Phil, then Bill is obligated to kill Phil quickly.

c. If Bill kills Phil quickly, Bill kills Phil.

d. Bill kills Phil.

e. Therefore, Bill is obligated to kill Phil.

Example (36c) is an inference for a certain class of adverbs ([189]). Example (36d) is asserted in the context. Given Example (36d), we infer from Example (36b) that Bill is obligated to kill Phil quickly. Given Example (36c) and OB-RM, we can then infer Example (36e). However, given OB-D, there can be no conflict of obligations; that is, given Example (36a), Example (36e) must be false. This is called a paradox since in SwDL, a contradiction is derived.

There are several possible ways to address this paradox; for example, one could deny OB-RM or deny OB-D. In section 2.9.1, we argue against OB-RM holding in natural language, in which case, the illicit entailment from Examples (36a)-(36d) to Example (36e) does not arise. In section 2.9.3, we similarly argue against OB-D, thus even if OB-RM were to hold, there would be no contradiction between Examples (36a) and (36e), which would just be an instance of conflicting obligations.

Though there are these two possible ways to address the paradox, we will consider a third, novel approach based on the semantic phenomena of focus. We provide this as an alternative to the others for it does not depend on the outcome of deliberations about OB-RM or OB-D.

To address the Gentle Murderer Paradox using focus, we give brief overviews from the literature of the logical forms of adverbs and the semantics of focus. We then provide our contribution: we show that the deontic operator obligation deploys focus within the sentence; the logical forms of such expressions block entailments otherwise found between expressions only containing adverbs. The result is that the entailment pattern which gives rise to Gentle Murderer Paradox is blocked. In particular, though Examples (36a)-(36d) can be true, Example (36e) does not follow because Bill is obligated to Kill Phil quickly does not itself imply Bill is obligated to Kill Phil. The reason is that the obligation operator introduces a focal interpretation such that Bill is obligated to Kill Phil quickly and Bill is obligated to Kill Phil have additional logical structure which blocks the inference. Such
an additional focal interpretation is not necessary in the statements of Example (36c). Finally, we briefly review previous discussions on this topic ([167] and [80]).

2.7.2 Logical Forms of Adverbial Sentences

First, it is clear that Example (37a) entails Example (37b) and not vice versa. According to one influential logical representation ([55] and [142]), we can account for this inference as a matter of First-Order Logic, given the logical form Example (37c), which entails the logical form Example (37d). We introduce these logical forms as they make clear the implicational relationship between Example (37a) and Example (37b). Issues relating to tense and the thematic roles of Agent and Theme are not relevant for the purposes of this example.

Example 37

a. Bill killed Phil quickly.

b. Bill killed Phil.

c. $\exists e [\text{killing}(e) \land \text{Agent}(e) = \text{bill}' \land \text{Theme}(e) = \text{phil}' \land \text{quick}'(e)]$,
   There is a killing event which has Bill as the Agent, Phil as the Theme, and is quick.

d. $\exists e [\text{killing}(e) \land \text{Agent}(e) = \text{bill}' \land \text{Theme}(e) = \text{phil'}]$
   There is a killing event which has Bill as the Agent and Phil as the Theme.

e. $P \rightarrow Q$,
   where $P$ is “Bill killed Phil quickly” and $Q$ is “Bill killed Phil”.

Consider these expressions in combination with OB. Suppose that Example (38a) is OB(P), Example (38b) is OB(Q), and Example (38c) represents an implicational relation between them.

Example 38

a. Bill is obligated to kill Phil quickly.

b. Bill is obligated to kill Phil.

c. $\text{OB}(P) \rightarrow \text{OB}(Q)$
In SwDL, Example (38a) \textit{logically} implies Example (38b) (though not vice versa). This follows since given the inference in Example (37e), the assumption of $\text{OB}(P)$, and $\text{OB}$-RM, we must infer $\text{OB}(Q)$.

This would seem to parallel the inference in the alethic operators, where we understand that if it is true that in every world, Bill killed Phil quickly, then it is true that in every world, Bill killed Phil.

\textbf{Example 39}

\begin{itemize}
  \item[a.] \textit{Necessarily, Bill killed Phil quickly.}
  \item[b.] \textit{Necessarily, Bill killed Phil.}
  \item[c.] $\Box(P) \rightarrow \Box(Q)$
\end{itemize}

However, we show that there is no such inference as claimed in Example (38). Rather, \textit{obligation} introduces a \textit{focus} interpretation ([153]) such that Example (38a) and Example (38b) have \textit{different logical forms} which blocks the relevant entailment. This is in contrast to alethic modalities, which do not introduce focal interpretations. To make our case, we very briefly outline the underlying intuitions behind the semantics of focus in section 2.7.3 and then return to the problem in section 2.7.4.

\textbf{2.7.3 Focus}

At this point, let us turn to consider the phenomenon of focus ([153]) and see how it is relevant to our discussion. We first illustrate how focus works with adjectival examples that are formally similar to the adverbial case ([142]). We then provide an adverbial example. Our point is that focus interpretation is a general semantic mechanism independent of adverbial modification. In the following section, we want to deploy a focus in our analysis of obligations and adverbs.

In Examples (40c) and (40d), we have the \textit{focus} element \textit{only} which focuses on a particular element in the sentence, indicated with capital letters to suggest \textit{intonational emphasis}. We see that Example (40a) entails Example (40b), However, Example (40c) does \textit{not} entail Example (40d) since Example (40c) could be true if Bill killed a tall man and also killed a woman, while Example (40d) is false if Bill killed a woman. Note that Example (40c) does entail Example (40b), so is it crucial to be specific about what is entailed.
Chapter 2 2.7 Deontic Operators and Adverbs

Example 40

a. Bill killed a tall man.

b. Bill killed a man.

c. Bill only killed a TALL man.

d. Bill only killed a MAN.

In terms of a focus interpretation ([153]), Example (40c) is true where Bill killed a tall man relative to the comparison class of men of various heights. The relevant parameter is height of men. We can paraphrase this as: relative to the class of men of varying heights, Bill killed a man who was tall and none of the short or medium height men. In other words, not just any man will do – it must be a tall man. However, the comparison class says nothing about other individuals such as women, so Bill could have killed a woman as well. In contrast, in Example (40d), the implication is that Bill killed a man relative to the class of people: relative to the class of people, Bill killed a person who was a man and none of the other subclasses of people, say women. Example (40d) would be false if Bill happened to have also killed a woman. As we have said, Example (40c) could be true if Bill killed a tall man and also killed a woman. In such a circumstance Example (40d) is clearly false. Furthermore, Example (40d) could be true where Bill killed a short man; so Example (40d) does not imply Example (40c). We cannot, then, say that either of Examples (40c) or (40d) imply the other since their truth-conditions are relativised to different comparison classes.

Thus, we have an explanation for the failure of entailment between Examples (40c) and (40d): the truth-conditions are defined with respect to different comparison classes. Note that, as the focal element shifts position or we shift the items bearing focal intonation, so shifts the semantic interpretation. For instance, if Bill kissed only a TALL GIRL, then the comparison class would be relative to height and sex. However, these shifts of interpretation are systematically dependent on the position and intonation.

By the same token, we can provide an adverbial example which is relevant to our discussion of obligation. Example (41a) can be true where Bill not only killed Phil quickly, but he also robbed Phil; Example (41b) is false in this situation. So, just as with Examples (40c) and (40d), Example (41a) does not entail (41b) where they both have focal interpretations. However, we do have entailments from focal interpretations to non-focal interpretations as well as from non-focal interpretations to other non-focal interpretations: Example (41a) entails Example (41c); Example (41b) entails Example (41d); and finally, Example (41c)
entails Example (41d). It is, then, crucial to be specific about which interpretation we are considering.

**Example 41**

a. *Bill only killed Phil QUICKLY.*

b. *Bill only KILLED Phil.*

c. *Bill killed Phil quickly.*

d. *Bill killed Phil.*

At this point, we return to our examples with obligation.

### 2.7.4 Obligation, Adverbs, and Focus

Our claim is that the *obligation* operator requires a focal interpretation in the sentence it applies to. Exactly what is focussed may vary (giving different interpretations), so we consider examples along the lines as discussed above. Thus, Examples (38a) and (38b) *implicitly* contain focal interpretations; we represent these *explicitly* in Example (42). Because each statement with an *obligation* operator has a focal interpretation, we can therefore *block the inference that gives rise to the* Gentle Murderer Paradox. Simply put, since *obligation* introduces a focal interpretation such as in Example (42) and we know that Example (41a) does not entail Example (41b), then from Example (42a) we cannot infer Example (42b).

**Example 42**

a. *Bill is only obligated to kill Phil QUICKLY.*

b. *Bill is only obligated to KILL Phil.*

In Example (42a), Bill has an obligation with respect to killing Phil quickly: killing Phil quickly fulfills the obligation, while killing Phil slowly violates it. This statement allows there to be other actions which Bill could do, but Bill does not have an *obligation* with respect to them. In particular, Bill could also rob Phil, but that is not obligatory. In Example (42b), Bill has an obligation with respect to killing Phil: killing Phil fulfills the
Chapter 2 2.7 Deontic Operators and Adverbs

obligation, while also robbing Phil would violate it. Thus, Example (42a) and Example (42b) represent different obligations, and Example (42a) does not imply Example (42b).

Let us return to Example (36) and our account of the Gentle Murderer Paradox. Following our claim that obligation requires focal interpretation, we suppose focal interpretations in all sentences where the obligation operator appears. Therefore, rather than Example (36), we represent the sentences which comprise the Gentle Murderer Paradox as in Example (43). Note that where no obligation operator appears, focal interpretation is optional (e.g. Examples (43c) and (43d)).

Example 43

a. Bill is only obligated NOT TO KILL Phil.
b. If Bill kills Phil, then Bill is only obligated to kill Phil QUICKLY.
c. If Bill kills Phil quickly, Bill kills Phil.
d. Bill kills Phil.
e. Therefore, Bill is only obligated to KILL Phil.

We assume that Examples (43a)-(43d) hold. Example (43d) applied to Example (43b) implies that Bill is only obligated to kill Phil QUICKLY. In this case, even though Example (43c) holds, it does not apply to an expression of the form Bill is only obligated to kill Phil QUICKLY, so we cannot infer Bill is only obligated to KILL Phil, for the logical forms are distinct. As we have shown, Bill is only obligated to kill Phil QUICKLY does not imply Bill is only obligated to KILL Phil. Therefore, we cannot infer Bill is only obligated to KILL Phil from the statements in Example (43). This solves the Gentle Murderer Paradox.

Our explanation resembles aspects of the analysis of ([189], [190], and [195]) for so-called factive adverbs such as wisely, stupidly, and rudely where they function as sentence modifiers: Rudely, Jill spoke to the Queen loudly does not entail Rudely, Jill spoke to the Queen. Thus, rather than providing an analysis particular to the obligation operator, we have integrated the analysis of the operator to related sentence modifiers.

It is not our intention here to provide a fully spelled out discussion of the issues. However, we have given good reason not to support the inference found in the Gentle Murderer Paradox where we take into consideration the interpretation of focus.
Notice that our analysis accounts for the failure of the inference *no matter what one’s position is vis a vis* OB-RM or OB-D since they play no role in the analysis.

### 2.7.5 Previous Proposals

Our proposal bears a resemblance to [167], but criticised in [80]. [167] claims that the obligation operator may apply to different propositions in the logical representations of Examples (37); however, this is not based on any linguistic theory, much less an analysis using focus. In particular, Example (44a) is translated as in Example (44b), where what is obligatory is the location of the kiss; Example (45a) is translated as in Example (45b), where what is obligatory is the kissing *per se*. In this way, the application of the OB operator is specific to different propositions. However, there is no notion in this approach of different comparison classes or the source of these different interpretations.

**Example 44**

\[
\text{a. Bill is obligated to kiss Jill on the lips.}
\]

\[
\text{b. } \exists e \left[ \text{kissing}(e) \land \text{Agent}(e) = \text{bill’} \land \text{Theme}(e) = \text{jill’} \land \text{OB(on-the-lips’}(e)) \right],
\]

There is a kissing event which has Bill as the Agent, Jill as the Theme, is on the lips, and it is obligatory that it be on the lips.

**Example 45**

\[
\text{a. Bill is obligated to kiss Jill.}
\]

\[
\text{b. } \exists e \left[ \text{OB(kissing}(e)) \land \text{Agent}(e) = \text{bill’} \land \text{Theme}(e) = \text{jill’} \right],
\]

There is a kissing event which has Bill as the Agent, Jill as the Theme, and such a kissing event is obligatory.

[80] criticises this analysis because the event-theoretic analysis of [55] does not clearly apply to adverbials such as *slowly* or *swiftly*. [142, p. 44-45] provides an analysis of these adverbials within the framework of [55] using an additional parameter, so the criticism of [80] does not stand. In our discussion above, we have presumed the solution of [142]. Instead, [80] proposes that defeasibility plays a role ([140], [145], [141], [175]). Defeasibility refers to a relationship between the premises and conclusion of an argument that *presumably holds*, but which may be defeated by additional information or by exceptions.
Chapter 2 2.8 Deontic Operators and Non-restrictive Relative Clauses

As we have justified earlier, we do not engage with issues of defeasibility or exceptions in this thesis as they do not appear central to our concerns. Our analysis does not make use of defeasibility since we are ascribing distinct semantic interpretations to expressions such that one does not imply the other. The implication between the expressions is not defeated for we do not claim it holds in the first instance.

2.8 Deontic Operators and Non-restrictive Relative Clauses

In this section (based on [194]), we consider the logical form of expressions with obligation and relative clauses. The logical form as assumed in SwDL gives rise to counter-intuitive logical inference, which is the Good Samaritan Paradox. However, this can be straightforwardly resolved by considering a linguistically well-motivated alternative logical form.

In the following, we review the Good Samaritan Paradox and provide a solution. Our solution relies on a syntactic and semantic analysis of non-restrictive relative clauses (NRCs) in contrast to restrictive relative clauses (RRCs). In particular, the NRC is not under the scope of the operator, while RRCs are. The Good Samaritan Paradox rests on the mistaken grammatical analysis of NRCs as RRCs. To develop our analysis, we review linguistic evidence concerning NRCs, then provide a review of Discourse Representation Theory (DRT), which is applied to NRCs and RRCs. Our contribution is to solve the Good Samaritan Paradox using DRT. In addition, we introduce a new related problem concerning obligations and RRCs. However, as this bears on the logic of quantification, we leave this for future work. This section again illustrates the importance of applying linguistic analysis both to solve a problem as well as to uncover a novel problem.

2.8.1 The Good Samaritan Paradox

The Good Samaritan Paradox is the following problem. Consider the following example:

Example 46

a. It is obligatory that Bill help Phil, who has had an accident.

b. Therefore, it is obligatory that Phil has had an accident.
Intuitively, it can be the case that Example (46a) is true, yet Example (46b) is false; that is, it may be the case that Bill is obligated to help Phil when Phil has had an accident, but it is by no means obligatory that Phil have the accident.

The paradox arises as a consequence of the linguistic analysis of Example (46a) and OB-RM. Where P is *Bill help Phil* and Q is *Phil has had an accident*, the presumed analysis of Example (47a) is Example (47b), for the moment filtering out issues associated with tense:

**Example 47**

*a. Bill helped Phil, who has had an accident.*

*b. $P \land Q$*

We would then analyse Example (46a) as Example (48a); since $[P \land Q]$ implies Q, then by OB-RM we infer Example (48b):

**Example 48**

*a. $OB[P \land Q]$*

*b. $OB[Q]$*

Our central claim in this section is that there is ample linguistic evidence that the clause *who has had an accident* in Example (46a) is a NRC. Furthermore, that in general, NRCs are presupposed information which appear outside the scope of sentential operators. Thus, a linguistically motivated analysis would provide Example (46a) with a logical form along the lines of:

**Example 49**

$$OB[P] \land Q$$

Clearly, this does not entail $OB[Q]$.

To provide a solution, we want a linguistically well-founded analysis which allows the non-restrictive relative clause to appear outside the scope of the sentential operator. As a step towards this analysis, we discuss the analysis of NRCs and RRCs in [8].
2.8.2 Restrictive and Non-restrictive Relative Clauses

[8] discusses the syntax and semantics of NRCs in contrast to RRCs, showing that they have different logical forms. In particular, RRCs are analyzed as composing with the nominal, while NRCs are treated to an analysis in Discourse Representation Theory (DRT). In this section, we focus on the syntactic and intuitive semantic distinctions between NRCs and RRCs. In the next section, we present DRT.

Consider the contrast between the following two sentences, where an NRC appears in (50a) and the RRC in (50b).

Example 50

a. *I bought the cheapest book, which was not a paperback.*

b. *I bought the cheapest book which was not a paperback.*

There is a phonological difference, which we overlook. The semantic difference is significant, where we see that the sentences do not have the same truth conditions. In (50a), the NRC does not restrict the meaning of *the cheapest book*, while the RRC of (50b) does. In (50a), the cheapest book out of all the books was bought; it just happened not to be a paperback. In (50b), only the cheapest among the non-paperbacks was bought, and it could well be that the cheapest book out of all the books was a paperback. RRCs are interpreted *intersectively*, taking the intersection of the set of things which are books and the set of things which are not paperback books. (50a) entails (51), but (50b) does not.

Example 51 *I bought the cheapest book.*

Furthermore, (50a) has almost the same meaning as a discourse, where we use subscripting to indicate co-reference.

Example 52 *I bought [the cheapest book]i. Iti was not a paperback.*

NRCs appear to be interpreted as if they are independent sentences, though the relative pronoun *which* has properties similar to an anaphoric pronoun such as *itself*, thus linking the NRC anaphorically to an antecedent in the sentence of which it is a part. [8] shows that the relationship of NRCs and discourse anaphora is significant.
Chapter 2  2.8 Deontic Operators and Non-restrictive Relative Clauses

[8] provides a very extensive and thorough catalogue of syntactic and semantic differences between NRCs and RRCs. Most significantly for our purposes, [8] shows that NRCs are outside the scope of sentential negation and more so escape the scope of propositional verbs. Example (53a) is a well-formed sentence with an RRC: the relative clause restricts the cars being considered to those with a broken window; Sam may own a car, but not one with a broken window. In contrast, Example (53b) is an ill-formed sentence with an NRC: it attempts to state that Bill does not own any car, and that the car which Sam does not own is a car with a broken window. Example (53c) is the discourse analogue of Example (53b).

Example 53

a. Sam doesn’t own a car that has a broken window.

b. # Sam doesn’t own a car, that has a broken window.

c. # Sam doesn’t own a car. It has a broken window.

In the following, we see the difference between RRCs and NRCs under the scope of a propositional operator.

Example 54

a. Kim believes that linguists who use the IPA are clever.

b. Kim believes that linguists, who use the IPA, are clever.

In (54a), Kim has a belief about linguists who use the IPA (International Phonetic Alphabet), implicitly contrasting them to linguists who do not use the IPA and so are not clever. In (54b), Kim has a belief about linguists in general, and it just so happens that linguists use the IPA. (54b) entails (55), while (54a) does not.

Example 55  Kim believes that linguists are clever.

2.8.3 Discourse Representation Theory

[8] provides an analysis of NRCs and RRCs in Discourse Representation Theory (DRT) ([101] and [11]). DRT is a formal semantic framework in which to represent natural
language discourse anaphora. In this subsection, we present just that background of DRT needed to make sense of the analysis of [8], which is presented in the next section.

DRT represents a discourse in Discourse Representation Structure (DRS), which are formal structures that represent the available discourse entities, the predicates and relations that hold of them, and any interpretive relationships between the entities. In addition, there is a means to restrict relationships which hold between the entities. DRT provides a formal, set-theoretic theory, but these are more easily presented in terms of a box notation. For example, we represent (56a) as in (56b).

Example 56

a. Sam owns a car. It is dented.

\[
\begin{array}{c}
s, c, w \\
\text{Sam}(s) \\
\text{owns}(s, c) \\
\text{car}(c) \\
\text{dented}(w) \\
c \approx w
\end{array}
\]

Boxes are divided in two. The top box of a box represents available discourse referents, which are just the individuals that can be referred to and predicated of. The bottom box contains predicates and relations among the discourse referents. Boxes can contain embedded boxes. There are three discourse referents – s, c, and w. The box in Example (56b) is read as: there is an individual s who bears the name Sam, there is an object c which is a car and which is owned by Sam, there is an object w which is dented, and w and c are the same thing. We have five statements. The first three represent the first sentence. The fourth represents the discourse continuation which introduces a new discourse referent w into the top box. The fifth, c \approx w, is a construal rule of referential dependency to link c and w. DRT provides the means to represent quantifiers found in natural language semantics as well as a range of other expressions.

DRT accounts for the contrast between a well-formed discourse as in Example (56a) and an ill-formed discourse as in Example (57a), by introducing additional structure into the representation along with restrictions on anaphoric construal.
Example 57

a. # Sam, doesn’t own a car, It, is dented.

b. $\langle s, w \rangle$

$Sam(b)$

$dented(w)$

$w \approx c$

$c$

$\approx car(c)$

$owns(s, c)$

The difference between Example (56b) and the representation of Example (57a) as in Example (57b) is the addition of a subordinate box which contains its own discourse referents and predicates. We see that owning a car appears in a box within the box that represents denting the box. There are systematic rules for introducing boxes and the material they contain. Furthermore, general rules apply which specify the accessibility of discourse referents at different levels of the structure. In this case, negation makes discourse referents under their scope unavailable for subsequent discourse anaphora. (57b) is ill-formed due to a violation of a constraint on DRSs: the discourse variable $c$ in the expression $w \approx c$ must be accessible to the discourse referent contained in the top box of the subordinate box that represents Sam doesn’t own a car, but is not. In general, discourse antecedents in subordinate boxes (boxes contained in boxes) are not accessible for anaphora outside of the box, but anaphors within a box can take discourse antecedents in superordinate boxes.

2.8.4 The DRT Analysis of NRCs and RRCs

We focus on the semantic analysis of NRCs and RRCs in [8]. The analysis of RRCs follows the well-accepted syntactic and semantic analysis of [158] where the RRC is a complex predicate formed by the meaning of the relative clause and the nominal. In addition, the RRC, as a grammatical unit, has a quantifier; we follow the Generalized Quantifier analysis of [15] and [104], where the quantifiers are relations between the noun and predicate. While
the existential quantifier, which introduces a flat representation, is not the best illustration of quantification, it is the simplest for our purposes. In the next subsection, we introduce our analysis of obligation and NRCs, applying the analysis presented here.

The DRT representation for the RRC in (58a) is (58b). Here, the relative pronoun is not treated as discourse anaphora.

**Example 58**

\begin{enumerate}
\item Someone\textsubscript{p} who\textsubscript{p} is tall left.
\item person(s) \nlft(s) \ntll(s)
\end{enumerate}

The RRC condition who is tall appears in the same DRS box as the conditions for the head noun a person.

In contrast, while NRCs are syntactically adjoined to the nominal just like RRCs, they are interpreted as independent sentences, and the relative pronoun functions as an anaphoric pronoun. Indeed, the discourse example and the NRC example have the same semantic representation.

**Example 59**

\begin{enumerate}
\item Bill left. He is tall.
\item Bill, who is tall, left.
\end{enumerate}
As in the discourse continuation, the NRC appears in the same box as the main clause, and a new discourse referent is introduced which is linked anaphorically to an extant discourse referent with $b \approx v$. The key difference between the representation in (58b) and (59b) is that, in the latter, an independent clause is introduced along with a construal of the pronoun with the antecedent.

The more important case for our purposes is the difference between embedded RRCs and NRCs.

**Example 60**

```
a. Sam$_a$ believes someone$_p$ who$_p$ is tall will win.
b. Sam$_a$ believes Kim$_k$, who$_w$ is tall, will win.
c. Sam$_a$ believes Kim$_k$ will win. She$_w$ is tall.
```

The RRC is represented as:

**Example 61**

```
\[
\begin{array}{c}
s \\
Sam(s) \\
believes(s, \{ \text{person}(p), \text{tall}(p), \text{will} - \text{win}(p) \})
\end{array}
\]
```

In this case, the discourse referent introduced for `person` as well as the content of the relative clause are all introduced in the box which represents the subordinate clause of `believe`.

In contrast, an NRC in (60b), like the discourse continuation in (60c), is *not* semantically in a subordinate clause. All the discourse antecedents are introduced in the top box.
Given a discourse continuation, we specify a referential dependency given by $w \approx k$; no construal rules are violated since the discourse antecedents are all accessible.

**Example 62**

\[
\begin{array}{|c|}
\hline
s, k, w \\
\hline
Sam(s) \\
Kim(x) \\
w \approx k \\
tall(w) \\
\hline
\end{array}
\]

As [8] notes, treating NRCs as discourse continuations addresses a range of semantic and syntactic observations. Key among them is that, by treating the semantic content of the NRC this way, the NRC is *outside* the scope of the propositional attitude verb and the scope of sentence negation.

While there are a range of issues related to discourse anaphora, such as accommodation phenomena and others, these are not issues particularly associated with the *Good Samaritan Paradox*. We can take this as a general solution. It should be said that while [8] provides this analysis, we are not provided with the compositional rules, which would be along the lines of λ-DRT ([11]).

### 2.8.5 The DRT Analysis of the Good Samaritan Paradox

At this point, we can apply the analysis of [8] for propositional attitude verbs to our modal operator *obligation*, for the analysis is essentially the same. As with other NRCs, (63b) means the same as the discourse (63c).³

³Notice that while the RRC can have present tense, the NRC is ungrammatical with it. This grammatical difference between the forms may be tied to the semantics of the NRC, which is *backgrounded* information. We can use the present tense form where we put it in the antecedent of a conditional, which
Chapter 2 2.8 Deontic Operators and Non-restrictive Relative Clauses

Example 63

a. It is obligatory that someone, who takes the money leaves the house.

b. It is obligatory that Bill, who took the money, leaves the house.

c. It is obligatory that Bill leaves the house. He took the money.

For the RRC with obligation, we have the following DRS.

Example 64

For the NRC or discourse continuation, we have the following.

---

also appears to have scope outside the obligation operator, though this touches on issues related to so-called conditional obligations.

a. # It is obligatory that Bill, who takes the money, leaves the house.

b. It is obligatory that Bill, if he takes the money, leaves the house.
Example 65

The key point to observe is that in the NRC, in contrast to previous presentations of the Good Samaritan Paradox, the relative clause does not fall under the semantic scope of the obligation operator, and so one cannot draw any inferences from it.

This then refocusses the issue to RRCs, which have a structure which would appear to lend themselves to the same sort of inferences as we find in the Good Samaritan Paradox.

2.8.6 Obligations and RRCs

Having discussed and solved the Good Samaritan Paradox as it is usually characterized, we have uncovered a distinct problem – the relationship between deontic operators and RRCs. One of the key differences is that while NRCs do not oblige one to analyse quantifiers, RRCs do. As such, we do not have recourse to the implication that arose in the Good Samaritan Paradox, but must consider those implications which follow from the quantifiers themselves in the structure they apply to. We consider the issues briefly, by way of introducing the problem, which we leave for future work. So far as we are aware, the problem we pose, the Restrictive Relative Clause Paradox is a novel problem for Deontic Logic, even if it is related to the Good Samaritan Paradox.

Consider the following quantified expressions.
Chapter 2 2.8 Deontic Operators and Non-restrictive Relative Clauses

Example 66

a. Some woman left the house.

b. Two women left the house.

c. Every woman left the house.

d. Most women left the house.

To account for the syntax and semantics of natural language quantifiers, linguists usually follow Generalised Quantifier theory (GQ) ([15] and [104]), where the quantifiers some, two, every, and most are relations between sets of expressions, the nominal woman and predicate left the house, which denote sets of individuals.4 In the following, we have the denotations as relations between sets, X (e.g. the set of women in the model) and Y (e.g. the set of individuals who left the house in the model), where X and Y are subsets of a universe of discourse U.

Example 67

a. some′(X)(Y) = true iff (X ∩ Y) ≠ ∅.

b. two′(X)(Y) = true iff (X ∩ Y) contains two or more members.

c. every′(X)(Y) = true iff (X ⊆ Y).

d. most′(X)(Y) = true iff (X ∩ Y) is bigger than (X − ∩ Y).

One of the crucial observations in GQ theory is that there is a spectrum of systematic implicational relations keyed to the lexical semantics of the quantifiers and that are not apparent from the surface syntax. For example, Example (66a) implies there was a woman, and Example (66b) implies there were two women. But Example (66c) does not imply there were every women (which is ungrammatical) or that everything (in the domain) was a woman, and Example (66d) does not imply that most things (in the domain) were women.

4GQ theory is a second-order theory as it requires relations and quantification over sets. In this thesis, we are not concerned with complexity and decidability issues, but a formal account of the empirical issues. Presumably, the facts and properties observed in GQ theory will to have find some computationally tractable solution, as after all, people do use the expressions.
Chapter 2  
2.8 Deontic Operators and Non-restrictive Relative Clauses

In order to sensibly combine quantificational issues with deontic operators, we use RRCs.\(^5\) First, we observe implications from RRCs. We have the quantifiers some, two\(^6\), every, and most. The nominals are woman who took the money; the predicates are went to jail.\(^7\) The facts are: Example (68a) implies Example (68a'); Example (68b) implies Example (68b'); Example (68c) does not imply Example (68c'); and Example (68d) does not imply Example (68d').

Example 68

\[\begin{align*}
    a. & \text{ Some woman who took the money went to jail.} \\
    a'. & \text{ Some woman took the money.} \\
    b. & \text{ The two women who took the money went to jail.} \\
    b'. & \text{ Two women took the money.} \\
    c. & \text{ Every woman who took the money went to jail.} \\
    c'. & \text{ Every women took the money.} \\
    d. & \text{ Most women who took the money went to jail.} \\
    d'. & \text{ Most women took the money.}
\end{align*}\]

Now we can consider the interactions between the operator obligatory and the quantified sentence. What is very interesting here is that given the entailments in Example (68) and OB-RM, we would not expect entailments from Example (69c) to (69c') or from Example (69d) to (69d'). However, we would expect entailments from Example (69a) to (69a') or from Example (69b) to (69b'). Yet, we do not find these entailments.

Example 69

\[\begin{align*}
    a. & \text{ It is obligatory that some woman who took the money goes to jail.} \\
    a'. & \text{ It is obligatory that some woman took the money.} \\
    b. & \text{ It is obligatory that the two women who took the money go to jail.}
\end{align*}\]

---

\(^5\)We provide the leading observations, not the formalisation. DRT provides for GQs and RRCs, so the DRT representations of the examples would be somewhat different from RRCs in Example (58).

\(^6\)Here and in the following, we overlook the syntactic difference between the two and two.

\(^7\)We overlook the number difference between woman and women.
b'. It is obligatory that two women took the money.

c. It is obligatory that every woman who took the money goes to jail.

c'. It is obligatory that every women took the money.

d. It is obligatory that most women who took the money go to jail.

d'. It is obligatory that most women took the money.

We refer to this as the *Restrictive Relative Clause Paradox*. Unlike the *Good Samaritan Paradox*, it is not apparent how a structural analysis could account for the missing inferences. In this case, we could apply our focus analysis as we argued for the *Gentle Murderer Paradox*; essentially, the predicate *go to jail* falls under the focus of the deontic operator, so the *obligation* is borne by those women with respect to going to jail, not on women taking the money. However, as the issues are not central to our primary topic, we leave these observations for future consideration.

### 2.9 Other Issues

To this point, we have discussed several issues in some depth. Broadly speaking, we have introduced problems of interpretation or logical form found in logical discussions of deontic concepts, then shown how linguistic considerations can shed light on the problems. In the following, we discuss several additional topics more briefly, due to space considerations, in the same manner. We do not attempt an exhaustive discussion of each topic or every topic found in the literature ([128]). Our objective is to take a position on these topics in order to provide a ground for our analysis of the CTD problem, which appear in chapters 3 and 4, and the implementation, which appears in chapter 5.

The issues we discuss, in order, are:

- Linguistic issues related to OB-RM;
- Obligations and temporal notions;
- Conflicting obligations;
- Antonyms and obligations;
- The relationship between obligations and permissions;
• Deontic Logic, *Sees-to-it-that*, and refraining;

• The relevance of Deontic Logic to the formal representation of contracts.

The subsections are separate units, highlighting discrete issues, rather than a thread of argument.

### 2.9.1 OB-RM

In section 2.6, we argued that the natural language correlate to expressions of Axiom OB-K are ill-formed, so we do not have evidence that the axiom is operative in natural language. In this section, we argue that the natural language expressions found in OB-RM are ill-formed, so OB-RM is not operative in natural language either. We briefly consider an argument which would seem to indicate that OB-RM does apply in natural language, but then provide an alternative analysis. Our contribution is to highlight relevant linguistic reasoning that can bear on the issue.

#### 2.9.1.1 Where OB-RM Appears to Apply

The logical form of OB-RM would appear to reasonably hold in the following. Clearly, Example (70a) implies Example (70b); it is also unobjectionable that Example (70c) implies Example (70d).

**Example 70**

a. Bill leaves the house and Jill leaves the house.

b. Bill leaves the house.

c. It is obligatory that Bill leave the house and Jill leave the house.

d. It is obligatory that Bill leave the house.

Symbolically, we have the following:
Example 71

a. \( [P \land Q] \rightarrow P \)

b. \( OB[P \land Q] \rightarrow OB(P) \)

However, in section 2.5 we argued that the obligation operator can only apply to expressions of the right sort, namely, in agentive, non-stative expressions. Arguably, a conjunctive expression per se does not meet these criteria, so \( OB[P \land Q] \) is unacceptable.

In addition, we can argue that Example (70c) is underlyingly to be understood as Example (72), which entails Example (70d) as well.

Example 72

*It is obligatory that Bill leave the house, and it is obligatory that Jill leave the house.*

In this case, Example (70c) and Example (72) have the logical form in in Example 73; \( OB[P \land Q] \) does not, then arise.

Example 73

\( OB(P) \land OB(Q) \)

A substantial part of work in *Transformational Grammar* ([37] and [39]) is devoted to accounting for how different sentence forms have the same semantic interpretation, postulating a basic form and a derived form. If this is the case, then we can account for the entailment without recourse to OB-RM.

Alternatively, were one to deny the implication in Example (71b), one could claim that Bill and Jill left together, and *that* is what is obligatory; that is, if Bill leaves first, and then Jill leaves afterwards, we would not consent that the obligation in this case has been met.

The point is that it is rather crucial to initially fix or define what structure and semantic interpretation one is working with and with respect to that, determine the implications; one cannot presume to do it for expressions which appear to be synonymous, but in fact are distinct.

89
At this point, we have shown that OB-RM is not operative in natural language. While there are other so-called paradoxes to discuss, such as the Ross Paradox and the Free Choice Permission Paradox, we leave them to future research. As we have previously pointed out, issues related to SwDL and particularly to analysis of the deontic concepts as propositional operators are tangential to our main topic.

2.9.2 Obligations, Time, and Deadlines

Temporal reasoning has long been a central topic in Deontic Logic ([125], [105], [58], [83], [27], [32]). In this section, we informally present some of the main issues.

In the literature on deontic operators, deadlines appear central to the analysis of the deontic concepts; they appear in the formal analyses of contracts as well ([83]). We contribute to the discussion elements of linguistic theory which show that the deontic operators also apply to processes over an interval of time; while deadlines are important to consider, they are but one sort of interaction between the deontic operators and temporal expressions. Deadlines cannot, in and of themselves, be used to address the CTD problem. Another relevant temporal issue, particularly for the analysis of the CTD problem, is sequence of tense. Turning to linguistic analysis, we contribute to the discussion by showing that sequence of tense is a distinct and highly complex issue from the deontic operators. In general, our claim here is that interactions between the deontic operators and temporal expressions ought not to be conflated. In our analysis, we simplify the issues to get at core aspects of the deontic operators before combining them with other operators.

2.9.2.1 Deadlines

Logics that fall under the rubric of Deontic Temporal Logic are combinations of Deontic Logic (of some form) and Temporal Logic ([173]), where Temporal Logic concerns modal operators which quantify over time points (or intervals) or introduce propositions relative to time points in order to logically represent natural language expressions such as always will be, at some time has been, or before time x, and so on.

As an instance of the issues, [83] provides a range of temporal cases which are relevant to protocols, rental agreements, and service level agreements. They discuss a logical representation of a range of types of deadlines, specify sanctions which can be introduced, and distinguish between persistent and non-persistent obligations. The logical language is Temporal Defeasible Deontic Logic, which we do not discuss. The following are two
sorts of deadlines, one a specific action in time and another which holds over an extended period of time.

Example 74

a. **Deadlines**: Bills must be paid within 30 days after receiving the invoice. Otherwise, a fine must be paid.

b. **Maintenance Deadlines**: Customers must keep a positive balance for 30 days after opening a bank account. Otherwise, a fine must be paid.

Where the obligations are violated, a violation predicate is introduced. For instance, if a customer does not keep a positive balance for 30 days after opening a bank account, then $\text{viol(pos)}$ is introduced, meaning the obligation to keep a positive balance has been violated.

[83] distinguish between persistent and non-persistent obligations, where a persistent deadline continues to hold over periods of time. Example (74a) is persistent since it holds every time an invoice is issued. Example (75) is non-persistent since after the wedding party the obligation to deliver the wedding cake no longer applies. Others discuss this in terms of the introduction or elimination of deontic expressions ([106]).

Example 75

*A wedding cake must be delivered before the wedding party.*

[83] is rather concrete. Other more theoretically-oriented approaches to similar issues are *Alternating Temporal Logic* ([27]), *Computation Tree Logic* with the action operator *Sees-to-it-that* ([58]), and *Temporal Deontic Logic* ([32]) which is a development of *Modal Action Logic* ([125], [105], [58]).

### 2.9.2.2 Aspectual Distinctions in Natural Language

Though the proposals referred to above often illustrate their discussions with natural language examples (i.e. [83]), there is little if any relation to a compositional semantics of temporal expressions of natural language ([41], [40], [64] and [178]). In particular, while there is some discussion in [27] and [29] of the linguistically relevant aspectual notions
achievements and processes/activities, they do not discuss the meaning or formalisation of these terms which then relate to notions of deadlines and persistence.

By and large, so far as we are aware, the focus of research in the literature on time and the deontic concepts is on achievements, which aspectually are understood to be telic, that is, have an end point. This is in contrast to processes which are atelic, where the end of the activity may not relevant. Nor is there any consideration of the components of sentence meaning which contribute to aspectual meaning such as the lexical semantics of the verb, the semantics of the direct object as count or mass, or interactions with temporal adverbial phrases.

Just to illustrate, a well-known test ([64]) for telicity (being telic or atelic) is whether a given verb phrase (the verb and direct object, for example) is compatible with a time-frame adverbial such as in an hour, understood in the sense of within an hour, or a time-span adverbial such as for an hour. We see grammaticality judgements shift in the following, showing that to build a house is telic, while to build houses is atelic. Such variations appear systematically across a large set of examples.

Example 76

a. John built a house in a month.

b. # John built a house for a month.

c. # John built houses in a month.

d. John built houses for a month.

Of particular relevance to our current topic is that we can apply a deontic operator in either sort of aspectual class:

Example 77

a. John is obligated to build a house in a month.

b. John is obligated to build houses for a month.

In Example (77a), it is clear that John violates his obligation where he fails to have a complete house by the deadline. However, in Example (77b), it is another matter to
determine where John violates his obligation to build several houses over a period of time, which is not specified with respect to a deadline. Presumably, John has the ability and other means to build several houses, but does not do so over the course of the month, so violating his obligation. Such an expression is a reasonable expression in a contract, say for workers in a construction crew.

Our point is that deadlines per se are not a hallmark of deontic reasoning, but another factor that can be considered in the overall analysis. In general, temporal reasoning is arguably not unique to deontic reasoning, nor does it particularly distinguish deontic reasoning from other forms of reasoning. Indeed, because temporal reasoning is so widespread in natural language ([178]), it is less critical to us to analyse in particular. If one does incorporate it into the analysis, then it ought to be based on the full range of issues.

2.9.2.3 Sequence of Tense

One other topic is relevant to clarify for later discussion of the CTD problem. Conditional expressions of the form discussed in section 2.6 are comprised of two clauses, one subordinate to the other, both of which contain tense. Given two clauses, each having a past, present, and future tense which may be simple, continuous, or perfect aspects, there are many interpretive combinations of the two clauses. These interactions between the temporal semantics of the clauses fall under the general rubric of sequence of tense ([94] and [171]).

For example, in Examples (78a) and (78b), we clearly understand Bill’s leaving to have occurred subsequent to Jill’s being happy. In contrast, in Examples (78c) and (78d), it is natural to understand that Bill’s leaving occurs prior to Jill’s being happy.

Example 78

\begin{itemize}
  \item[a.] Bill left. Jill had been happy.
  \item[b.] If Bill left, then Jill had been happy.
  \item[c.] Bill will leave. Jill will be happy.
  \item[d.] If Bill leaves, then Jill will be happy.
\end{itemize}

The mechanisms to formally account for this are intricate and a substantive research area in their own right. In our discussions in later chapters, we choose to filter out this
independent issue of sequence of tense in order to highlight issues specifically related to the deontic concepts. Not to do so requires one to then provide an analysis of sequence of tense.

### 2.9.3 OB-D and Conflicting Obligations

One of the axioms of SDL, Axiom \textbf{OB-D}, implies that there are never conflicting obligations; that is, there is no context in which \textbf{OB}(P) and \textbf{OB}¬P both hold. The rationale is clearer in the semantics, for were \textbf{OB}(P) and \textbf{OB}¬P both to hold in a world \(w\), then in every ideal world \(v\) accessible from \(w\), \([P \land \neg P]\) holds. Clearly, such obligations give rise to a logical inconsistency.

However, outside of purely logical or mathematical domains, and especially in the context of moral reasoning, one often reasons approximately or in terms of counterbalancing some sort of account, rather the absolutes of logic. Indeed, there are different positions on Axiom \textbf{OB-D} ([128]). We state our position on this matter and why.

Though the logic is clear, the axiom does not represent intuitive uses of obligations, where we can have such conflicts. In our view, the problem arises because in SwDL, we only consider the propositions \(P\) and \(\neg P\) which both hold in ideal worlds. Consider the following:

#### Example 79

\begin{itemize}
  \item \textit{a. It is obligatory that Bill take care of his children at home.}
  \item \textit{b. It is obligatory that Bill take care of his mother in the hospital.}
\end{itemize}

Clearly, Bill cannot accomplish both of these actions at once, so one or the other \textit{obligation} will be violated. These two holding in one and the same world are not inconsistent, ill-logical; they do not hold in an uncommon circumstance. Rather, Bill would have to determine which to fulfill or violate relative to other goals or priorities. We want a representation of the deontic concepts, as well as of fulfillment and violation, which allow us to represent such conflicts of obligation without giving rise to logical inconsistency and which would allow the agent, in principle, to reason with the consequences.
2.9.4 Antonyms and Obligations

In this section, we point out a relationship between the actions which fulfill and violate an obligation. In particular, those actions which fulfill and violate an obligation are in a relationship of opposition. Lexical semantic analysis ([45, pages 197-247] and [72]) provides a range of conceptions of opposition, among which we find the logical notion of propositional complementaries of Propositional Logic. This is relevant to the SwDL discussion of the CTD problem, which relies on Propositional Logic, as well as the ScDL (State-changing Deontic Logic), which relies on set-theoretic complements. Our contribution is to point out another notion of opposition, the lexical semantic relation of antonymy. The advantage of such a relation is that it implies specific alternatives which induce violation leaving other actions underspecified with respect to the deontic operator. This is in contrast to propositional negation and set-theoretic complementation.

Antonyms are those expressions which have an opposition in meaning, usually discussed with respect to lexical items in which there is a component of meaning that expresses negation. For example, one can say that alive is the antonym of dead.

2.9.4.1 The Antonym Restriction

We observe that there is a symmetry of obligations in terms of fulfillment and violation conditions, which we refer to as the Antonym Restriction. With respect to the first issue, consider the following:

Example 80

\begin{enumerate}
\item a. Bill is obligated to leave the room.
\item b. Bill is obligated to remain in the room.
\end{enumerate}

Let us assume Example (80a) holds and Example (80b) does not. We intuitively know what it takes to fulfill or violate Example (80a): any action which makes it so that Bill is no longer in the room may be taken to be sufficient to fulfill the obligation; that is, any way that Bill has to move from being in the room to being out of the room is sufficient; on the other hand, any action which does not entail that Bill is any longer in the room might be taken to violate the obligation. For example, if Bill remains in the room, sits in the room, or walks around in the room. We see in this example that it is entirely reasonable to specify for a given obligation the circumstances which give rise to violation and those
which give rise to fulfillment. Indeed, were it to be otherwise, then for every obligation that is defined, a specification of the violation and fulfillment conditions would have to be given in advance before the obligation were understood. But it is rather easy to provide cases, such as that above, where one can determine what they are without having them given in advance.

When we refer to a symmetry of obligations, what we mean is that were the obligation to be shifted from one statement to its antonym, the fulfillment and violation conditions shift correlatively. For instance, if Example (80b) holds and Example (80a) does not, then those actions which fulfilled Example (80a) are those actions which violated Example (80b), and those actions which violated Example (80a) fulfill Example (80b).

The Antonym Restriction appears to hold of SDL and the reduction: if \( \text{OB}(P) \), \( P \) fulfills the obligation and \( \neg P \) violates it; if \( \text{OB}(\neg P) \), \( \neg P \) fulfills the obligation and \( P \) violates it. To match this to our discussion of Example (80), we have to ensure that Bill’s leaving the room and Bill’s remaining in the room correlate to \( P \) and \( \neg P \), respectively. This requires lexical semantic analysis.

2.9.4.2 Underspecified Actions

Lexical semantic analysis is required since there may be actions which are other than those which clearly indicate violation or fulfillment of the obligation. The actions we considered above, doing something to remain in the room versus doing something to get out of the room, do not cover all the possible actions, namely, those which do not entail that Bill is no longer in the room or entail that Bill remains in the room. For instance, lighting a cigarette, speaking to Jill, jumping up and down, or breathing are all actions that Bill (we suppose) can perform, but which do not entail that he is in the room, out of the room, or staying in or leaving the room. Executing any one of these actions ought not, then, itself imply a violation or fulfillment of the obligation to leave the room. Thus in determining which properties or actions contribute to the specification of violation and fulfillment, we want to restrict ourselves to the right set.

We must be careful about the generality of what we are claiming, for in contrast to the example above, we can specify some domain of actions such that every action either implies that a property holds or does not hold. For instance, suppose the only actions one has available or are under consideration are moving a left toggle or a right toggle in one of four directions (up, down, left, right). Furthermore, suppose Bill is obligated to move the left toggle up. In this constrained domain of actions, we may understand that any action
other than moving the left toggle up will violate the obligation because every one of them implies that as a result of the execution of the action, the left toggle is not up. Thus, in this domain, every action is deontically specified. However, this example is a subcase of the previous example, where we had deontically underspecified actions.

### 2.9.5 Obligations and Permissions

In Deontic Logic, the deontic operators are interdefined. In SDL, obligations imply permissions. While this issue is of general concern in the analysis of the deontic concepts, it does not directly bear on the CTD problem. Nonetheless, some comments are relevant here, as they reappear in chapter 4 once we define the deontic operators in terms of complex violation and fulfillment markers; the issues also appear in the implementation in chapter 5. We do not make a contribution to the discussion in the literature other than to place our view.

As we allow conflicting obligations and do not accept Axiom OB-D, then obligation does not imply permission. However, if OB(R) does not imply PE(R), then it would be possible that R is obligatory, OB(R), and R is also prohibited, PR(R). Were OB(R) to imply PE(R), then this could not be so. While we have not ruled out conflicts of obligations, this seems to be a conflict of another sort.

Another intuition is that we may want to distinguish permissions from rights, where rights are relational notions in the sense of [42]. However, we do not address Hohfeldian notions of rights or powers in this thesis. We must also distinguish notions of permission. One notion of permission in SDL is that permission is the absence of prohibition as in Definition 3. Yet another notion is the facultative, where a permission implies what is permitted can also not hold: PE(R) → PE(¬R) [2]. This is a stronger notion of permission than found in SDL. For our purposes, we find it more natural to consider this facultative interpretation in the sense that an agent which is given a permission on an action would want to know the implications of executing or not executing the action, which is clearer in the facultative interpretation.

For our purposes, let us consider the relationship between obligation and permission in terms of violation (and/or fulfillment), for the reductionist analysis requires that every deontic operator be specified in these terms. Furthermore, we consider just the facultative interpretation. We show that obligation does not imply permission under this interpretation.
We have claimed that obligation implies violation and fulfillment conditions. In the same vein, permission too ought to be defined in terms of violation and fulfillment. Suppose Bill is permitted to park his car in the College quad. If he parks his car in the college quad, we do not say that he has violated any deontic specification. If he does not park his car in the college quad, we also do not say that he has violated any deontic specification. He is neither obligated to park his car in the college quad, nor obligated to park his car elsewhere. Indeed, Bill may arguably bear this permission whether or not he is able to execute it for whatever reason. The key point is just that whether he parks his car in the quad or not, he does not violate anything.

Keeping these observations in mind, obligation could not imply permission since obligation implies that where the action is not executed, there is a violation of deontic specification on that action, but in the same circumstances, permission implies there is no violation of a deontic specification on that action. That is, suppose Bill is obligated to park the his car in the College quad, where he then commits a violation by not parking his car in the College quad. But in just this same circumstance, were obligation to imply permission, then Bill has a permission not to park his car in the quad and that not parking his car in the quad does not give rise to a violation. This is contradictory. Note the difference between conflicting obligations in SDL and outright contradictions such as $P$ and $\neg P$ in Propositional Logic, where $P$ is any proposition.

For our purposes, we do not assume that obligation implies permission. The relation between obligation and permission is not, in our view, a matter of one being defined in terms of another, but a matter of how the operators are defined in an overall framework. To make an analogy based on lexical semantics, there are lexical terms that are defined in terms of opposition such as up and down, where defining one allows the definition of another. There are other lexical terms which are defined relationally in the framework of an overall system, such as master and slave ([45]); these terms are not in logical opposition, but in opposition with respect to a conceptual system. We think the relation between obligation and permission falls into this latter class. To systematically associate lexical items in formal semantics, we can introduce meaning postulates ([136], [66], [64]) which are restrictions on the denotations of terms in a model. We return to such issues in chapter 4.

2.9.6 Deontic Logic, Sees-to-it-that, and Refraining

To this point, we have very informally mention actions in the course of discussing examples. In this section, we briefly outline some points concerning actions and refraining from actions in SwDL. We do not adopt SwDL and the notion of action allied with it, for
we are primarily interested in state change. However, it is relevant to make these points by way of comparing and contrasting state-wise and state-changing theories, which is an underlying contribution of the thesis. We outline the issues and relate them to our notion of antonyms of actions.

2.9.6.1 Actions in SwDL

In a state-wise language, we have an operator to represent the result of an action relative to an agent, e.g. $E_a$ and $E_b$, where $a$ and $b$ are agents. The operators apply to propositions. The expression $E_aA$, where $A$ is a proposition, is often read as agent $a$ sees to it that $A$ or agent $a$ brings it about that $A$, we can refer to as Sees-to-it-that (STIT) ([16], and [98], among others). As said in [98, p.11]: "...this logic focuses exclusively on who the agent is, and on the state of affairs which the exercise of his agency produces, ignoring both temporal aspects and consideration of the means by which a particular state is brought about." In this, the operator is intended to be an abstraction of actions. The operator $E_a$ abides by the following axioms, among others ([98, p.11]):

**Definition 8**

- $E.RE$ If $A \leftrightarrow B$, then $E_xA \leftrightarrow E_xB$
- $E.T$ $E_xA \rightarrow A$

The deontic operators of SDL can apply to expressions of the form $E_aA$, which are of type proposition. For instance, in SDL, $OB(E_x F) \equiv OB(F \land E_x F)$.

2.9.6.2 Action Negation and Refraining

Now let us consider negation. As expressions of the form $E_xA$ are propositions, we can apply negation: $\neg E_xA$, which has the interpretation that it is false that $x$ brings it about that $A$ holds. This is the notion of inaction, where an agent has not done something to bring about a particular result. Note that this is not the same as a notion of action negation; it is a proposition which states that no action has been executed by the agent to bring about $A$.

A notion of refraining (RF) can be defined with the action operator and the negative expression:
Chapter 2  2.10 Deontic Logic and the Formal Representation of Contracts

Example 81

\[ RF_x(A) = \text{def } E_x(\neg E_x A) \]

We read this as: Agent x refrains from seeing to it that A if and only if Agent x sees to it that x does not see to it that A. Thus, refraining from doing something to see to it that A is distinct from an inaction to see to it that A; indeed, it is another action. Considering our example with the toggles, suppose Bill has two actions he can execute, moving the left toggle up or down; he must execute one or the other. Bill’s execution of the action to move the left toggle up can be understood as an action which refrains from moving the left toggle down, and vice versa. By moving the left toggle up, Bill has seen to it that he has not seen to it that the left toggle is down.

However, refraining in this sense is intuitively too general, for any A which is different from B, where an agent sees to it that A then the agent refrains from seeing to it that B. However, we do not examine these issues more closely here since the significance of the refraining operator is not as a definition of a notion of refraining, but rather in the relationship of actions in systematic opposition with respect to some property – doing one action implies an opposition to some other specific action which yields a different result. In our discussion of action negation in chapter 4, we return to this topic.

2.10  Deontic Logic and the Formal Representation of Contracts

We have in this chapter considered a range of issues bearing on SwDL, some empirical and others more theoretical. We have claimed that some purported problems are avoided by better linguistic analysis of the underlying logical form. In other instances, our claims have been based on intuitive arguments.

As we made clear in chapter 1, the focus of our research is the formal representation of contracts. Our approach so far has been to evaluate which logical forms of SwDL suit the language in which the concepts of contracts are expressed. At this point, let us step back from our particular investigations to ask of the range of problems and structures we have so far examined, which of them appear in actual sample contracts ([99], [70], [85], and [83]), are discussed in the legal literature specifically devoted to legal contracts ([19] and [146]), or are discussed in the more general legal literature ([144]).
2.10 Deontic Logic and the Formal Representation of Contracts

2.10.1 The Epistemic and Non-epistemic Distinction

First, there is evidence of the non-epistemic interpretation of legal obligations. It is far less clear that the epistemic interpretation is relevant to discussions of the law per se. Though many matters may be contractually specified such as Do not water the plants outside if it rains for more than two days straight, this does not imply that the weather is a relevant legal concept. In other words, simply because a term or expression can or does appear in a legal contract does not necessarily imply it is a legal concept and so subject to our examination.

Moreover, notions tied to the non-epistemic interpretation explicitly do appear in discussions of the law and of contracts; breach of contract and sanction are central to the law ([19], [146]), and [144]). Thus, the Kanger-Andersonian Reduction would appear to be a very appropriate logical formalism for the representation of legal contracts.

2.10.2 Relationships among Obligations

In section 1.1, we introduced an example which reappears later in the thesis where one obligation is violated gives rise to another obligation. We can refer to the first as a primary obligation and the second as the secondary obligation. We discuss the issues below further in chapter 3, but briefly mention them in the context of this section.

There is support in the legal literature for the distinction and the importance in representing a temporal consequence of breach of contract or sanction; this is particularly relevant to the CTD problem, discussed in chapters 3 and 4. Concerning the effects of a breach of contractual obligation in the United Kingdom and with reference to Photo Production Ltd v Securicor Transport Ltd [1980] AC 827 (HL):

Lord Diplock: . . . [B]reaches of primary obligations give rise to substituted or secondary obligations on the part of the party in default, and, in some cases, may entitle the other party to be relieved from further performance of his own primary obligations. . . .

Every failure to perform a primary obligation is a breach of contract. The secondary obligation on the part of the contract breaker to which it gives rise by implication of the common law is to pay monetary compensation to the other party for the loss sustained by him in consequence of the breach . . .

[146, p.370]
Where conditional obligations appear, they appear in the form $P \rightarrow OB(Q)$; as pointed out in chapter 1, these are obligations which hold relative to some condition (i.e. antecedent). However, in the contract samples available to us, the forms appear to be so-called *forward* conditional, where the antecedent temporally precedes the consequent.

The other forms of conditional obligations discussed in section 2.6 (i.e. *obligation* operators on both antecedent and conclusions; an *obligation* operator over the whole conditional) are not, so far as we see, explicitly discussed in the legal literature, nor are they clearly in evidence in sample contracts.

In a similar vein, it is notable that the many other problems discussed with respect to Deontic Logic simply do not appear or are not topics of analysis in the legal literature. For example, neither the *Gentle Murderer Paradox* nor the *Good Samaritan Paradox* are discussed. Similarly, neither the *Ross Paradox* nor the *Free Choice Permission Paradox* appear. Deadlines are clearly relevant, but these are not a characteristic “problem” of contracts. We find no discussion that *obligations* imply *permissions*. In sum, there appears to be little *empirical support within* the legal literature or from sample contracts to the many concerns raised by Deontic Logic. It would appear that, by and large, Deontic Logic is not *relevant* since its concerns are not concerns shared with *legal professionals* since the problems do not, by and large, arise. Given the intensive scrutiny that the language of the law is under by legal professionals over time and given the highly detailed spectrum of problems they discuss, we may fairly conclude that the problems of Deontic Logic do not appear in legal language. The best explanation of this empirical gap is that the logical forms and semantic interpretations of Deontic Logic are not well-founded in natural language observations, as we have claimed in this chapter.

2.11 Summary

In this chapter, we have introduced two main variants of SwDL – SDL and KAR. We have discussed the lexical semantics of the deontic concepts which we take to be relevant for our domain of inquiry, distinguishing between the *epistemic* and *non-epistemic* interpretations. On empirical grounds, we argued that *obligation* operators appear in the *consequent* of a conditional only and that *Axiom K* does not apply in natural language. The *Gentle Murderer Paradox* and the *Good Samaritan Paradox* were resolved using linguistic analyses based on focus and the logical form of non-restrictive relative clauses, respectively. We then briefly considered a range of additional issues, clarifying our position on them.
In general, we have shown that linguistic analysis can fruitfully be applied to problems of Deontic Logic, whether by resolving them, showing what properties the logic ought to have according to the facts of the domain of inquiry, or introducing novel problems. As we noted at the outset of the chapter, one of the main points was to clarify a range of problems that are relevant to the CTD paradox.

2.12 Next Chapter: State-wise Deontic Logic and the CTD Paradox

In the next chapter, we discuss State-wise deontic operators with particular reference to the CTD problem. This is the problem where fulfillment or violation of one obligation implies another obligation. We outline the problem, then present our reanalysis of it, abstracting the issues from the deontic operators and showing significant issues which have not previously been emphasised. In our reanalysis, we introduce violations and fulfillments as intermediate elements of the argument. We then review and evaluate the main, current proposal concerning the CTD Paradox, showing how it differs from our proposal and does not clearly apply to contracting.
Chapter 3

State-wise Deontic Logic and the CTD Paradox

3.1 Introduction

In this chapter, we discuss the *Contrary-to-Duties Paradox* (CTD paradox) with respect to SwDL, for it is claimed to be the central problem concerning the deontic concepts and the one which distinguishes logics for the deontic concepts from logics for other modal concepts ([30] and [129]). Thus any analysis of the deontic concepts must provide a thorough analysis of the CTD paradox. In addition, as we have shown in sections 1.3.1, the structure of *Promise-Breach-Remedy* is central to legal contracting. We claim that it corresponds to the structures of expressions in the CTD paradox. Finally, as we have said, the CTD paradox provides a thread which relates issues and approaches.

Our contributions to the literature appear in four sections. In section 3.2, we first review the CTD paradox (following [30] and [129]), which is the formal version of what we intuitively introduced in chapter 1: an agent is obligated to do something, violates the obligation, and consequently bears some additional obligation. Given our discussion in chapter 2, we streamline the logical form of the problem by removing conflating issues (in comparison see [30] and [129]). In section 3.3, we then discuss and reanalyse the problem in a novel way. We show that a more general form of the problem is relevant to other modal operators; thus, the underlying logical form of the CTD problem is *not* distinctive of the deontic operators, contra [30] and [129]. Moreover, we show that a range of alternative contexts beyond those usually considered ([30] and [129]) help us to better understand the

---

1We use the terms CTD paradox, CTD problem, and CTD structure interchangeably.
roles of the components of the logical form of the CTD problem. Our reanalysis motivates our alternative, novel formalisation in section 3.4, in which we introduce explicit propositions that represent the intermediate concepts that express the violation or fulfillment of an obligation. However, it is not a reductionist analysis of the deontic operators as in section 2.3.2. More importantly, we use these propositions to reason to secondary obligations. What distinguishes the deontic operators from other modal operators is that they imply a particular interpretation of the intermediate concepts. In section 3.5, we consider our approach against the criteria for an analysis of the CTD paradox proposed in [30] and show what our approach addresses.

The two main points to take away from this chapter and carry into chapter 4 are:

- The CTD problem can and should be represented in a streamlined structure which explicitly represents and reasons with propositions that represent violation and fulfillment; and
- Alternative contexts are relevant

In addition, the material in this chapter presents issues in a form which are again relevant in chapter 4, where we discuss ScDLs. Thus, the discussion provides clear and novel points of comparison and contrast between these approaches.

### 3.2 Contrary-to-Duties and SwDL

In this section, we review the CTD paradox, following the discussion presented in [30] and [129, Section 4.5]. We start with the original version of the CTD paradox, then discuss how and why we streamline the discussion.

#### 3.2.1 The Original CTD Paradox

We start with some discussion of the original form of the CTD paradox because it is relevant to the criteria of [30, p.210]:

Example 82

a. *It ought to be that a certain man go to help his neighbours.*

b. *It ought to be that if he goes he tells them he is coming.*

c. *If he does not go to help his neighbours, he ought not to tell them he is coming.*

d. *He does not go to help his neighbours.*

[30] claim that there is consensus among researchers that these expressions are consistent (can all be true in one context), logically independent (none are logically derived from another), and imply that the obligation expressed in Example (82a) is violated given Example (82d), which itself implies that the man ought not to tell his neighbours he is coming. However, there is disagreement about how to formulate SwDL such that the logic represents the expressions, allows them to be consistent and logically independent, and does not yield intuitively implausible results. The statements in Example (82) represent a paradox in the sense that there is disagreement about the formulation and implications of the expressions together.

[30] provide a background discussion of previous proposed alternative logical forms that could be provided for this set of expressions and the inferential problems that arise (see [129, Section 4.5] for a succinct and clear presentation).

However, we streamline the discussion to the form found in Example (82), which is called the forward CTD paradox. The reason we focus on this example is other CTD structures conflate issues which we have discussed in chapter 2: *ought* is ambiguous; we find the deontic operator with scope over the conditional in Example (82b); Example (82b) and Example (82c), though they are used in similar ways in the overall structure of the discussion, have distinct surface forms, which is an unnecessary conflation of issues; we have backwards sequence of tense; and the appearance of a negated expression in the consequence of the conditional in Example (82c) (i.e. the negation of the consequence of the conditional in Example (82b)) is unnecessary for the overall considerations. To get to the basic issue that CTDs raise, it makes strategic sense to strip away all these confounding issues.

### 3.2.2 Our Working Form for the CTD Paradox

With these points in mind, rather than Example (82), we consider the set of expressions in Example (83). We have chosen to work with this set of expressions, called the forward
Chisholm set, since they avoid the confounding issues just mentioned. We point out other forms of the Chisholm set which we do not address in section 3.2.3. ²

Example 83

a. It is obligatory that Bill visits Jill.

b. If Bill visits Jill, it is obligatory that he leaves his visiting card with her.

c. If Bill does not visit Jill, it is obligatory that he sends a letter of regret to her.

d. Bill does not visit Jill.

These expressions are consistent. Example (83d) implies that the obligation expressed in Example (83a) is violated. Furthermore, Example (83d) together with Example (83c) imply that it is obligatory that Bill sends Jill a letter of regret. There is, however, a general logical problem, namely, that the expressions are not logically independent. We return to this in a moment after we provide the logical forms of the set of expressions and introduce some of the terms of discussion.

We assume the set of expressions above is represented as follows, where OB is obligation, P is Bill visits Jill, Q is Bill leaves his visiting card with Jill, and R is Bill sends a letter of regret to Jill. Otherwise we have the material conditional and propositional negation.

²Furthermore are surface syntactic forms that are closer to the semantic interpretation of the logical forms proposed in [30]; indeed, while [30] provide examples that follow this surface syntactic form, they provide a different logical form, yet have a semantics which has a narrow scope deontic operator. Briefly, [30] use examples with a surface syntactic form along the lines of If A, then it is obligatory that B, where the deontic operator only appears in the consequent of the conditional. However, for a logical form, they use a Dyadic obligation operator, OB(B/A), where B/A is a form of the conditional meaning wherever A, then B. [30] appear to adopt the dyadic operator to address independence. The Dyadic operator appears to correspond to the wide-scope syntactic form in It is obligatory that if A, then B. Yet, from the semantics, which we present more fully later, OB(B/A) is true in a model if and only if, in any context X, where A is true and B is possible, B is obligatory. Clearly, deontic specification semantically applies most significantly to the consequent; this is consistent with our claims in chapter 2. We agree with [24], who claims that OB(B/A) makes no commitments about the logical relationship of the deontic operator and the conditional. One would have to inquire as to the precise semantic difference, if any, between OB(B/A) and (OB(B))/A, whether we adopt this as a relevant difference, and what bearing the difference has on our core topic of violation and fulfillment conditions. Moreover, there may be alternative formalisations, unexplored in [30], that circumvent these criticisms.
Example 84

a. $\text{OB}(P)$

b. $P \rightarrow \text{OB}(Q)$

c. $(\neg P) \rightarrow \text{OB}(R)$

d. $\neg P$

For terminology, we follow [129, Section 4.5]. Example (84a) is referred to as the primary obligation, which says that it is obligatory that $P$ holds. It is a primary obligation in the sense that it does not follow conditionally; that is, it is not the consequent of a conditional, but holds as a statement. Example (84d) is the factual claim. Example (84a) conjoined with Example (84d) implies that the primary obligation has been violated. Were $P$ to hold instead, then $P$ conjoined with Example (84a) implies that the primary obligation has been fulfilled. $\text{OB}(R)$ in Example (84c) is the contrary-to-duty obligation (CTD obligation), for it is the obligation which is implied in the context where the primary obligation has been violated; in this instance, the primary obligation has been violated where $\neg P$ holds. $\text{OB}(Q)$ in Example (84b) is the compatible-with-duty obligation (CWD obligation), for it is the obligation which is implied in the context where the primary obligation has been fulfilled; in this instance, the primary obligation has been fulfilled where $P$ holds. CTD and CWD obligations are referred to as secondary obligations in the sense that they hold conditionally; they are the consequents of conditionals.

One problem discussed in the literature (cf. [30]) is the logical independence of the statements in Example (84). This is claimed to be a problem for the analysis of the CTD paradox. In particular, for a proposition $\phi$, it logically implies a conditional with the negation of $\phi$ as antecedent: that is, $[\phi \rightarrow (\neg \phi \rightarrow \psi)]$. This holds in general for Propositional Logic, and it is not particular to modal operators. As we mentioned in chapter 2, the problem of logical independence is a complex, widespread, and very significant problem for modal operators; so while it is a problem for the logical analysis of the deontic operators, it is not a problem particular to them. Presumably, a general solution ought to be provided for the interactions of implication and modal operators which would apply as well to the deontic operators. We have not here attempted to address much less resolve these issues. Moreover, in chapter 5, where we present our implementation, we have production rules rather than a logical system. In the following, to simplify our discussion, wherever we have $\rightarrow$, we assume an interpretation of the conditional which enforces independence, though we do not consider the matter further.
Before discussing this structure in greater detail, we refer to some additional CTD structures and why we do not take them into consideration in this thesis.

### 3.2.3 Additional CTD Structures

Before we move on to our reanalysis, we should briefly discuss what appear to be alternative forms of the CTD problem. We claim that there is essentially one CTD structure, but the interactions between the deontic operator and other linguistic phenomena give rise to the impression of a multiplicity of structures. As we discussed in chapter 2 and suggested above, it is best to separate out independent and confounding issues which must be addressed separately in any case. Once we have additional clarity about these independent issues and the deontic concepts, then one would be in a position to systematically examine their interactions.

We discuss two examples for illustration. We provide only the primary obligation along with conditional expressions.

#### 3.2.3.1 The Dog Example

The first example concerns obligations on *stative* expressions.

**Example 85**

- *There ought to be no dog.*
- *If there is no dog, there ought not to be a warning sign.*
- *If there is a dog, there ought to be a warning sign.*

There would be (at least) two alternative contexts – one in which there is a dog and one in which there is not. Clearly, the deontic operator applies to stative expressions. Because of this, [30] claim that this constitutes a separate CTD problem from that in which a deontic operator applies to an action expression. However, [191] argues against the distinction and for a notion of *maintenance*, where an action is obligated which maintains or returns the state as required. We do not, therefore, believe that this constitutes a substantively different *abstract* structure for the CTD paradox.
3.2.3.2 The Gentle Murderer

In this example, we have adverbial modification, where *do it* anaphorically refers to the antecedent verb phrase *kill Mr. X*; thus, we understand Example (86) as *If you kill Mr. X, you should kill Mr. X gently.*

Example 86

a. *You should not kill Mr. X.*

b. *If you kill Mr. X, you should do it gently.*

Consider a case where contrary to the primary obligation, you kill Mr. X. In this context, it is implied that you should kill him gently. If killing gently implies killing *under the scope of the deontic operator*, then the implication that you should kill him gently would itself imply that you should kill him, which introduces a conflict with the primary obligation. However, as we have discussed in chapter 2, the implication from *It is obligatory that you should kill him gently* to *It is obligatory that you should kill him* does not hold. So, the so-called paradox does not arise in virtue of logical form.

It is not our goal in this thesis to provide full analyses of every paradox or problem that arises with respect to the deontic operators and the expressions they apply to. Rather, we can keep a focus on one case which most clearly represents that abstract logical form of the CTD problem and which is the most relevant to the development of our analysis, avoiding conflating issues.

3.3 Towards a Reanalysis of the CTD Structure

At this point, let us return to considerations about Example (84). We introduce two further abstractions to the discussion which show that, contra [30], the CTD problem is a particular case of modal reasoning under alternative circumstances. The first abstraction is that violating or fulfilling the primary obligation need not imply secondary obligations, but could instead just imply some other proposition. The second abstraction is that we could have a modal operator other than *obligation*, yet find similar reasoning patterns. Having introduced these abstractions, we then consider the role of alternative factual claims and what they imply. This shows that the CTD structure does not, in and of itself, make clear important semantic intuitions about the *obligation* operator.
3.3 Towards a Reanalysis of the CTD Structure

3.3.1 Abstracting from Secondary Obligations

In Example (84), expressions of the form \( \text{OB}(Q) \) and \( \text{OB}(R) \) are simply propositions; there is no clear reason why it is essential to discuss secondary obligations per se, though this is highly relevant particularly for legal reasoning in contracting. For our purposes here, which is to isolate the core logical structure of the CTD problem, we can abstract from the secondary obligations and use propositions Q and R. Thus we have the following logical form. For convenience, we continue to refer to this as a CTD structure and the propositions as secondary obligations.

Example 87

\begin{enumerate}
\item \( \text{OB}(P) \)
\item \( P \rightarrow Q \)
\item \( (\neg P) \rightarrow R \)
\item \( \neg P \)
\end{enumerate}

As before, R holds in the context where the primary obligation is violated with respect to the factual claim \( \neg P \). Q holds in the context where the primary obligation is fulfilled, e.g. where P holds.

3.3.2 Abstracting from the Primary Obligation

We can also abstract from the deontic operator, using \( \text{Op} \) to represent some modal operator rather than \( \text{OB} \). For instance, we could interpret \( \text{Op} \) as possibility or belief.

Example 88

\begin{enumerate}
\item \( \text{Op}(P) \)
\item \( P \rightarrow Q \)
\item \( (\neg P) \rightarrow R \)
\item \( \neg P \)
\end{enumerate}
We can reason in ways similar to Example (84). For example, if Op is necessity, then Op(P) means that P necessarily holds; in all worlds accessible from the world of evaluation, P holds. It happens to be that ¬P holds, which implies R; in some other circumstance, where P holds, Q is implied. Thus, Q holds in all the worlds accessible from the world of evaluation holds; R holds in the worlds which are inaccessible. Alternatively, if Op is John believes that, then Op(P) means that John believes that P. It happens to be that ¬P holds, then the context is one which we can understand to be contrary to John’s belief; in this context, R holds. We could also consider alternative circumstances, where Q holds in a context which is in keeping with John’s belief. Clearly, these patterns of reasoning are similar in form, which indicates that this underlying, abstract structure of reasoning does not distinctively characterise deontic reasoning.

### 3.3.3 Alternative Contexts

We have one other element of the structure to consider – the role of the factual claim in the argument. To this point, we have considered contexts where ¬P holds. Instead, let us consider two other alternatives – where P holds and where some other proposition, say S, holds. For the sake of discussion, we assume that Op is interpreted as obligation.

We use our observations in this section to draw attention to the following questions:

**Question 1**: Why are two contrasting conditional statements used?

**Question 2**: What is the role of the factual claim?

By contrasting conditionals, we mean two conditional statements along the forms of Examples (88b) and (88c), where the antecedents are negations of one another. By comparison, such contrasting conditionals do not obviously or routinely appear relative to other modal operators, for example, replacing the OB of obligation with possibility or belief. Thus, we can expect answers to these questions to help us to understand the distinct and fundamental properties of the obligation operator.

To address these questions, we consider alternative contexts such as where the primary obligation is fulfilled (in section 3.3.3.1) and where the primary obligation is neither clearly fulfilled nor violated (in section 3.3.3.2).
3.3.3.1 The Primary Obligation is Fulfilled

First, suppose we overtly represent that context in which the primary obligation is fulfilled; that is, in contrast to Example 88, we provide the context where P holds. This makes explicit the SwDL claim that the accessible contexts are partitioned between those where \(\neg P\) holds and which violate the obligation and those where P holds and which fulfill the obligation.

Example 89

a. \(Op(P)\)

b. \(P \rightarrow Q\)

c. \((\neg P) \rightarrow R\)

d. \(P\)

Here Q is implied since it follows from what holds in the context. This would correlate to the CWD obligation.

With respect to Question 1, there are two aspects. For our purposes, the obligation operator implies that there is some context where the content of the obligation is fulfilled and there is some other context where the content of the obligation is violated. While there is no empirical evidence to support it, for legal contracting it is intuitively unreasonable to oblige an agent with respect to some proposition and leave him either with no way to fulfill it or no way to violate it. Indeed, for an agent to be an agent at all implies some choice over what holds, so we claim that in principle, it must be possible for either to hold. This implies that wherever an obligation is expressed, there must be at least two contexts, one in which the obligation is fulfilled and another where the obligation is violated. In addition, there appears to be a strong and fundamental intuition that consequences must follow in either case; indeed, were there to be no consequences from fulfilling or violating an obligation, it is hard to imagine how agents could use it as a guide for behaviour.

We claim that an obligation implies that there is a way to fulfill it or violate it, and furthermore, that there are consequences which follow from fulfilling or violating it. Thus, we are explicitly requiring that there is a possible context in which the statements in Example (87) hold and another possible context in which the statements in Example (89) hold. In effect, we have at least these two templates for reasoning with respect to obligation, and both templates must be instantiated. An alternative way of expressing this is that we fix

113
the deontic expressions and the conditions, while *varying* the factual claim related to the primary obligation. Thus, for a given set of deontic expressions and conditionals, we at least have contexts in which factual claims hold that fulfill or violate each of the deontic expressions. This amounts to an *admissibility* requirement on models in which obligations are expressed. Even if *overtly* we do not express both conditionals in our logical forms, we stipulate that they do hold semantically; in other words, even if one were simply to state only $\text{OB}(P)$, we would suppose conditionals along the lines of Examples (89b) and (89c). The factual claim specifies the context and implies some other proposition. We must emphasise that this requirement may only be relevant for the domain of legal reasoning about contracts and not to *every* way that the deontic (or other normative) notions are used in other domains, about which we make no claims.

### 3.3.3.2 A Proposition Unrelated to the Primary Obligation

With respect to *Question 2*, the factual claim appears to be conflate two roles: it indicates whether the primary obligation has been violated or fulfilled; as the antecedent of a conditional, it also implies a proposition. We believe these different roles of the factual claim ought to be explicitly distinguished in a formal analysis.

To see why they ought to be distinguished, consider a case $S$ is some proposition which is not logically equivalent to either $P$ or to $\neg P$. Furthermore $S$ or $\neg S$ are the antecedents of conditionals.

**Example 90**

- $\text{Op}(P)$
- $S \rightarrow Q$
- $\neg S \rightarrow R$
- $\neg S$
- $\neg P$

In this case, $S$ is not related to the primary obligation. We see that the primary obligation has been violated since $\neg P$ holds. In this example, $\neg S$ implies $R$ and since $\neg S$ holds, $R$ holds.
The question is whether R is a CTD obligation? In one sense, it is not. The factual claim \(\neg S\) which implies R is not related to the primary obligation; there is no “direct” relationship between \(\neg S\), the primary obligation, and R. R is not implied by that which violates the obligation. In another sense, the pattern of reasoning is very similar to Example 89 since after all, R is implied in the context where the obligation has been violated. It just happens to be that what violates the obligation \(\neg P\) is not what implies R. Yet, in our view, what is missing is some explicit expression of the relationship between the primary obligation, the factual claim which indicates that the primary obligation has been violated, and that proposition which implies the CTD. The example ought to clearly differentiate between Example 89 and Example 90.

Finally, in general, the relationship between the factual claim and the primary obligation is not overt in any of the previous informal expressions or their logical representations. Indeed, were the operator \(\text{Op}\) to have a different interpretation, none of the properties discussed with respect to the deontic operators would hold, though it is unclear how the logical forms brings this out.

In this section, we have shown that structures as in Examples (87)-(90) do not make explicit what is necessary for as well as distinctive about deontic reasoning in the CTD structure. We provide an alternative formalisation to make these elements explicit.

### 3.4 An Alternative Formalisation

In this section, we provide a novel formalisation of the CTD problem, drawing some of the observations above together. In particular, we explicitly express violation and fulfillment relative to the primary obligation and a proposition. However, we are not here reducing deontic operators to Alethic Logic and a designated violation proposition as in the *Kanger-Andersonian Reduction*. Rather violation and fulfillment are introduced as intermediate concepts, for they are defined in terms of more fundamental ones – the statement of obligation and the facts that hold in the context. The essential role of such intermediate concepts is argued for in [154], [121], and [12], where the case is made that while they may appear redundant from a purely logical point of view, they are conceptually essential to reasoning, particularly in the law. As we point out later, it is not novel to propose explicit expressions of violation, though explicit expression of fulfillment is less widely recognized. However, it is novel to propose that they are implied concepts rather than defined or postulated. Furthermore, it is novel to explicitly use violation and fulfillment expressions in the antecedents of the conditionals which imply the secondary obligations.
Our proposal is in contrast to Examples (87)-(90), where the antecedents of the conditionals serve several roles. In our proposal, the proposition which appears as the factual is distinct from the proposition which we use to infer the secondary obligation. These latter propositions appear as violation or fulfillment expressions. Their justification is founded in the intuitive function of the factual claim relative to the primary obligation and as antecedent of the conditionals which imply a secondary obligation. Moreover, though such structures may not feature in the surface form of contracts, we claim that this is the underlying logical form by which one reasons to secondary obligations. Having made it explicit, we have also made it easier to implement.

However, we must emphasise that the underlying concept of the proposal is further refined when we turn to consider ScDL in chapter 4. Thus, it is a proposal relative to SwDL.

### 3.4.1 An Informal Set of Expressions

To set the proposal, consider the following informal set of expressions, which are logically consistent.

**Example 91**

a. It is obligatory that Bill visits Jill.

b. If Bill has fulfilled his obligation to visit Jill, it is obligatory that he leaves his visiting card with her.

c. If Bill has violated his obligation to visit Jill, it is obligatory that he sends a letter of regret to her.

d. If it is obligatory that Bill visits Jill and Bill visits Jill, then Bill has fulfilled his obligation to visit Jill.

e. If it is obligatory that Bill visits Jill and Bill does not visit Jill, then Bill has violated his obligation to visit Jill.

f. Bill does not visit Jill.

Asserting the statements in Examples (91d) and (91e) satisfy our intuitive requirement that there is a way to fulfill or violate the primary obligation in Example (91a). Given Example (91f), it is implied that the primary obligation has been violated; if in some
alternative circumstance, *Bill does visit Jill*, then it would be implied that we have a way to fulfill the primary obligation. Examples (91b) and (91c) satisfy our intuitive requirement that something *follows* from fulfilling or violating the primary obligation.

Notice here that we have *shifted* the formal role of the factual claim. As before, it is used in Examples (91d) and (91e) to imply fulfillment or violation of the primary obligation. However, it *no longer serves as the antecedent of the conditions* in Examples (91b) and (91c); as such, the factual claim no longer *directly* implies the CTD or CWD obligations. Rather since both the primary obligation and the factual claim both hold, we have the implication that the primary obligation has been violated. From the violation of the primary obligation, we infer the CTD obligation holds. Thus, we *preserve* the inference to the CTD obligation, which is fundamentally required by the CTD structure. Similarly, if *Bill visits Jill* held, then the primary obligation would have been fulfilled and the CWD obligation would be that Bill leaves his visiting card with Jill. Alternatively, were some factual claim other than either of these to hold and the models to be partial, we *could not make any inference to the fulfillment or violation of the primary obligation, nor what follows from it*.

Our proposal has a clear advantage over previous representations, for it indicates that Example (91) expresses a *tight*, *explicit*, and *coherent* relationship between the primary obligation and the CTD and CWD obligations which is otherwise obscured.

We still have the problem of the *logical independence* of the expressions: Example (91f) implies that Example (91d) is true. As we have pointed out earlier, this is a general problem of Propositional Logic and the properties of the material implication, not a specific problem to our proposal or Deontic Logic; some general approach is therefore warranted.

### 3.4.2 A Formal Set of Expressions

We formally represent our set of expressions in Example (91) as follows, where T and U are propositions that represent the statements of violation or fulfillment (i.e. the consequents in Examples (91b) and (91c)), and where Q and R are any propositions. The other template is expressed where the factual claim is P. Recall that we must assume a general solution to logical independence with respect to \( \rightarrow \), which is a significant problem on which much depends.
Example 92

a. \( OB(P) \)

b. \( T \rightarrow Q \)

c. \( U \rightarrow R \)

d. \( \{OB(P) \land P\} \rightarrow T, \)

\( \) where \( T \) represents the proposition that expresses the fulfillment of \( OB(P) \).

e. \( \{OB(P) \land \neg P\} \rightarrow U, \)

\( \) where \( U \) represents the proposition that expresses the violation of \( OB(P) \).

f. \( \neg P \)

As pointed out earlier, the propositions \( T \) and \( U \) function as \textit{intermediate concepts} that mediate between the primary obligation and the factual claim with what follows should the primary obligation be violated or fulfilled. The advantage of such a representation is that it addresses central underlying conceptual and formal issues in reasoning in the CTD structure, while making these explicitly part of the reasoning process.

Our goal in this chapter has been to provide an analysis of the problem rather than a fully spelled out a solution. In particular, we leave it as an informal matter how the proposition which expresses violation or fulfillment is constructed; for an abstract level of analysis, this is not crucial. As we do not adopt a SwDL analysis, it is not necessary to provide an analysis in this context. However, one advantage, from our point of view, of making explicit what is otherwise often implicit is that it provides a useful conceptual framework which can be \textit{implemented}, particularly for the representation of contracts; it is not clear how the \[30\] system would be implemented.

Nonetheless, in an explicit logic and implemented system, it is crucial to systematically provide some representation of the meanings of the intermediate concepts represented by the propositions \( T \) and \( U \). This is carried out in chapters 4 and 5.

3.4.3 A Comment on Logical Form

It is worth commenting that there is clearly a trade-off between staying close to the surface syntactic form of the natural language expressions and providing this more elaborate logical representation of the implicit semantic structure. In formal analysis of natural
Chapter 3

3.4 An Alternative Formalisation

language semantics, such issues about the relationship between the syntax and semantics often arise. For instance, in *diathesis alternations*, two different syntactic forms have the same semantic interpretation as in the active *Bill kissed Jill* and the passive *Jill was kissed by Bill*; alternatively, one syntactic form such as *Every woman was kissed by some man* may have two non-synonymous logical forms. It is then acceptable to provide an analysis of the logical form of expressions which does not exactly match what appears to be the surface syntax. What is key is to provide well-justified, intuitively plausible, and well-formed representations, which we have done. In comparison, our earlier claim against *Axiom K* rested on the observation that natural language semantics broadly rules out the logical form in which a modal operator appears in the antecedent of a conditional.

3.4.4 A Generalisation to Other Operators

As [30] claim that CTD structures are particular to deontic operators, we should generalise our structure so as to provide a test of the claim. In the following we provide the *Contrary-to-Operator* schematic as announced at the beginning of the chapter: \( \text{Op} \) is a modal operator; \( P, Q, R, S, \) and \( T \) are propositions; we make explicit an implicit assumption of (Example 92), which is that the intermediate concepts are semantically distinct. For instance, they could be contradictory propositions.

**Example 93**

- \( \text{Op}(P) \)
- \( Q \rightarrow S \)
- \( R \rightarrow T \)
- \( [\text{Op}(P) \land P] \rightarrow Q \)
- \( [\text{Op}(P) \land \neg P] \rightarrow R \)
- \( Q \not\leftrightarrow R \)

Where we want to reason to \( Q, R, S, \) or \( T \), we make an assertion of a fact: Where \( \neg P \) holds, then \( R \) holds, from which it follows that \( T \).

The deontic operator \( \text{OB} \) satisfies this schematic. The question is, then, what other modal operators satisfy the schematic, though provide different values for the intermediate concept? While it is worth recording this observation and question, further investigation is beyond the scope of this thesis.
3.5 The Criteria

In this section, we discuss the criteria which [30] claim any theory of the CTD problem ought to address. We discuss how our reanalysis of the issues and alternative formalisation address the criteria.³

Example 94

a. capacity to represent the fact that a violation of an obligation has occurred.

b. consistency, meaning that there must be a context in which all the expressions of the CTD are true.

c. capacity to derive actual obligations

d. capacity to derive ideal obligations

e. logical independence of the members, meaning that we cannot infer any expression from the others.

f. applicability to timeless and actionless CTD-examples as in There ought to be no dog.

g. analogous logical structures for the two conditional sentences.

h. the assignment of logical form to each of the norms in the set should be independent of the other norms in it.

i. capacity to avoid the pragmatic oddity.

We point out how our analysis in Example (93) relates to the criteria. Some of our observations relate to our analysis in chapter 4.

Represent Violations: Our analysis clearly and explicitly represents violations and fulfillments. We motivate them and use them in reasoning.

Consistency: Our analysis is consistent.

Actual and Ideal Obligations: [30] distinguish between actual and ideal obligations. The actual obligation is that obligation which follows from the conditional expression and the factual claim: suppose $\text{OB}(P), P \rightarrow \text{OB}(R), \neg P \rightarrow \text{OB}(S)$, and $\neg P$. We infer $\text{OB}(S)$,

³We have rearranged some of the items from the original.
which [30] call the actual obligation. We call $P \rightarrow \text{OB}(R)$ the ideal obligation for were $P$ to hold instead of $\neg P$, then we would infer $\text{OB}(R)$, which [30] call the ideal obligation. For our purposes, we are not convinced of the need to derive these alternatives; in the implementation in chapter 5, we derive what [30] call the actual obligations, and what they call ideal obligations are actual obligations where some other condition is met.

**Logical Independence:** We have pointed out that this is a general, very significant problem of Propositional Logic and Modal Logic which needs to be addressed independently. It is not problematic in a dynamic theory (cf. [130]) which we discuss in chapter 4, nor is independence an issue in the implementation in chapter 5, where we have production rules.

**Timeless and Actionless Expressions:** In chapter 2 and in section 3.2.1, we claimed that these issues are not directly, in and of themselves, relevant to the analysis of the CTD structure.

**Analogous Logical Structures:** [148] argue that in a context where *it is obligatory that there is no dog*, If there is a dog, there ought to be a warning sign is a CTD conditional, but not *If there is no dog, there ought to be no warning sign*. Only *If there is a dog, there ought to be a warning sign* implies a CTD obligation when the primary obligation is violated. [30] claim that both *If there is a dog, there ought to be a warning sign* and *If there is no dog, there ought to be no warning sign* should be treated alike in terms of logical structures and inferences that can be drawn. We agree with [30]. However, in our view, the issues here fall under the purview of general methodological points concerning the translation of natural language expressions into logical forms, which is, by and large, not discussed in the deontic logic literature ([149], [30], and [128]). Indeed, they are additional matters to be discussed along with the linguistic topics in chapter 2; the position of [30] would be presumed in natural language semantics unless there is reason otherwise. In chapter 5, we provide analogous logical structures.

**Independent Norms:** [148] argue for obligations that are relativised to contexts. In a context where *it is obligatory that there is no dog, if there is a dog, then it is obligatory that there is a sign*, and there is a dog, the CTD obligation can be relativised to this context: $\text{OB}_{\text{no\_dog}}(\text{there is a sign})$. In this way, the obligation on there being a sign is relativised to the obligation that there be no dog. This is one way to address the context dependence of CTD obligations. [30] claim that norms should not be relativised in this way. We agree, and in chapter 5, provide a means to introduce CTD obligations without subsorts of obligations.
Pragmatic Oddity: The remaining problem to discuss is the so-called pragmatic oddity of Example (94i). The pragmatic oddity appears in the CTD examples. Suppose a primary obligation $\text{OB}(P)$ is violated with respect to a factual claim $\neg P$, and $\neg P \rightarrow \text{OB}(R)$. Then, $\text{OB}(P)$ and $\text{OB}(R)$ both hold. This means that in all ideal versions of this world, where $P$ holds and $R$ holds. But, this seems odd and not found in correlated natural language expressions. [30, 215-220] discuss attempts to resolve this issue and provide their own solution. We do not provide an analysis of the pragmatic oddity since it is particular to SwDL, where the deontic operators are defined with respect to deontic alternatives and are applied to propositions. The problem does not arise in ScDL, nor in legal contracts, where there is a clear notion of time and change of state such that the deontic specifications and propositions change from state to state. Finally, the problem does not appear to be germane to our central concerns, which are the uses of articulated violation and fulfillment markers.

Having discussed the CTD structure, a reanalysis, an alternative formalisation, and the criteria for a theory of CTDs, we can turn to consider the approach in [30]. In the following section, we discuss and comment on the fundamental semantic features.

3.6 Summary

In this chapter, we have outlined the fundamental semantics features of [30], identifying aspects we adopt or adapt in the development of our approach. We have discussed some of the basic CTD structures, focusing on one particular case in order to identify for ourselves their underlying common structure and to filter out auxiliary issues that must be dealt with in other ways. We have provided a semi-formal reanalysis of this central CTD structure, introducing explicit violation and fulfillment markers. We have also used intermediate concepts to reason to secondary obligations. The point of the reanalysis is to make explicit components of reasoning which are implicit in other approaches. We use such explicit components in our analysis of the CTD problem in subsequent discussions.

3.7 Next Chapter: Deontic Specification and the CTD Problem in ScDLs

In the next chapter, we turn to consider deontic specification and the CTD structure in State-changing Deontic Logics (ScDLs). Interestingly, though the languages are substantively different, we find similar issues arising with respect to reasoning to secondary
obligations and the conditions under which violation or fulfillment arise. To these con-
siderations we add a further issue, which is the relationship between simple and complex
expressions. The central novel idea we introduce is that the violation and fulfillment ex-
pressions represent not just the deontic operator, that which the operator applies to, and
(in principle) the agent responsible for the deontic specification, but as well, information
related to the *construction* of the expression the operator applies to. This leads us to fur-
ther enrich the representation of violation and fulfillment so as to account for the notion
of violating or fulfilling *procedures* or *protocols*. 
Chapter 4

State-changing Deontic Logic, the CTD Paradox, and Sequences

4.1 Introduction

In this chapter, we discuss obligations, sequences, and the CTD problem with respect to State-changing Deontic Logics (ScDLs), which are logics with which we reason with respect to state change and actions which have deontic operators applied to them. We turn to state-changing logics for the following reasons: we want to model executions of contracts; many of the paradoxes of SwDL do not arise for ScDL, as we point out in this chapter; and ScDL, which is a reductionist theory, does not make use of notions such as ideality as in SwDL. Nonetheless, though SwDL and ScDL are different, some key considerations raised in chapter 2 and 3 are still relevant. In particular, we must examine the ScDL treatment of the the CTD structure.

We consider the CTD structure as it appears in ScDL. One of our contributions is to show that ScDL has several problems with respect to the CTD structure, which are similar to the problems of the SwDL analysis. One central difference between ScDL and SwDL is that ScDL also expresses complex actions such as sequences of actions to which deontic operators can apply; in ScDL, the CTD problem is analysed in terms of obligations on sequences of actions. We show this is not a satisfying solution, and we distinguish between sequences of obligations (SOO) from obligations on sequences (i.e. protocols) (OOS), where one cannot be reduced to the other.
To resolve these problems, we make several novel proposals. First, and the main contribution of the thesis, is the reductionist analysis of deontic operators on actions to actions and fine-grained violation and fulfillment markers in an open and flexible framework. Second, we show that for practical deontic reasoning, negation of an action cannot be the complement set of actions from the domain of action. Rather, we should use a notion such as antonym in natural language lexical semantics as we indicated in section 2.9.4.

The most general goal of the thesis should be kept in mind – a flexible, open framework and implemented tool in which we can express and exercise alternative definitions of the deontic concepts as applied to actions. In addition to our specific proposals, we discuss issues and problems for consideration in the design of the implementation. Thus, in this chapter, we are not giving a logic, and we have not addressed formal properties of the framework such as completeness or decidability for two reasons. First, the choice of definitions for the deontic operators in the language is still under discussion. Second, the formal properties which our tool, as an instance of a social simulator, ought to satisfy is also under discussion ([59], [61], [68], and [71]).

The structure of the chapter is as follows. The basics of a “generic” ScDL analysis are introduced, and the CTD structure is expressed in it. Then we show a range of problems for this analysis such as changing the primary obligation and action negation. We make the case for the difference between obligations on sequences and sequences of obligations. The ScDL analysis is briefly compared to the SwDL analysis. We then propose our reduction of the deontic operators on actions to actions plus articulated, fine-grained violation and fulfillment markers. It is demonstrated how this analysis solves the CTD problem in ScDL. We then review, analyse, and compare main prior ScDL proposals.

### 4.2 ScDL

In this section, we provide a minimal introduction to one version of ScDL – the Dynamic Deontic Logic (DDL) of [130] – focussing on the syntax. The objective here is to make clear our central issue of the reduction of deontic operators applied to complex expressions in a dynamic language, and more particularly, the application of the obligation operator to sequences of actions. This sets up our analysis of the CTD structure in an ScDL framework. For the moment, we focus on a syntactic presentation and discuss the semantics below.
4.2.1 Basic Elements of ScDL

[130] expresses the logic of obligation, permission, and prohibition on actions in dynamic logic. Dynamic logic is a weak modal logic like system K, but with extra axioms for actions. One of the key aspects of a dynamic logical system is that actions and assertions are strictly separated, which avoids paradoxes and counterintuitive propositions which appear in SDL [130, p.109].

4.2.1.1 Atomic Actions, Complex Actions, and Action Negation

In DDL, we have action names such as $\alpha$ and $\beta$, which are syntactic entities that we use to denote atomic actions. The action names denote abstract semantic actions $\alpha'$ and $\beta'$. We assume a set of Atomic Actions, of which $\alpha'$ and $\beta'$ are two. In DDL, there is no way to further specify properties of atomic actions or relations among them in terms of more basic attributes; atomic actions have no fine-grained structure. Complex actions are constructed from atomic (or complex) actions by action combinators. For example, given actions $\alpha$ and $\beta$ and the sequence combinator on two actions indicated by $';'$, then $(\alpha;\beta)$ is the sequence formed by first executing $\alpha$ and then executing $\beta$. We also assume propositions $\phi$ and $\psi$.

Given an action name $\alpha$ and a proposition $\phi$, we may form the proposition $[\alpha]\phi$. As an expression of the form $[\alpha]\phi$ is a proposition, we can apply the propositional connectives to it, such as negation in $\neg[\alpha]\phi$. We suppose that the action $\alpha'$ of $[\alpha]\phi$ can be executed in a state in which the weakest preconditions defined by the action, whatever they are, hold and results in a state in which the postconditions, whatever they are, hold along with $\phi$. For our purposes, we assume that the action can only be executed where the preconditions hold and, if the action is executed then the postconditions hold. Furthermore, we assume that action execution is not necessary wherever the preconditions hold. However, as we mention later, action execution is not itself defined in DDL, so these are assumptions outside the scope of DDL itself but useful in the course of discussion. Action negation, the negation of action $\alpha$ indicated by $\overline{\alpha}$, is given axiomatically; intuitively, the negation of an action denotes the complement set relative to $\alpha$ of actions of the domain of actions ([29]). A proposition of the form $[\overline{\alpha}]\phi$ we take to mean for any action $\beta$ in $\overline{\alpha}$, $[\beta]\phi$. The issue of action negation is itself a substantive issue; for our immediate purposes, we follow this intuitive semantics of action negation, showing the problems it gives rise to. We discuss alternative notions of action negation (e.g. [29], [106], and antonyms) later in this chapter.
4.2.1.2 Deontic Operators on Atomic Actions in ScDL

Deontic operators apply directly to action names of atomic or complex actions. Where the action name is atomic, the expression is reduced to an action and a violation marker; where the action name is complex, the expression is reduced to some complex of actions and violation markers. In [130], the violation marker is the special propositional letter V (which first appeared in [7]). Given an arbitrary atomic action name \( \alpha \), a state \( \sigma \), and deontic operators \( \text{OB} \) (obligation), \( \text{PR} \) (prohibition), and \( \text{PE} \) (permission) which apply to the action name, we have the following:

**Definition 9**

a. \( \sigma \models \text{PR} \alpha \) iff \( \sigma \models [\alpha](V) \)

   It is forbidden to do \( \alpha \) in \( \sigma \) iff when one performs \( \alpha \) in state \( \sigma \), there is always a violation in the resulting state.

b. \( \sigma \models \text{OB} \alpha \) iff \( \sigma \models \text{PR} \overline{\alpha} \) iff \( [\alpha](V) \)

   It is obligatory to do \( \alpha \) in \( \sigma \) iff it is forbidden to do \( \overline{\alpha} \) in state \( \sigma \) – doing \( \overline{\alpha} \) always results in a violation. In other words, doing anything other than \( \alpha \) leads to a violation.

c. \( \sigma \models \text{PE} \alpha \) iff \( \sigma \models \neg \text{PR} \alpha \) iff \( \neg [\alpha](V) \)

   It is permissible to do \( \alpha \) in \( \sigma \) iff it is not forbidden to do \( \alpha \) in state \( \sigma \). It is false that doing \( \alpha \) always leads to a violation. It says nothing in particular about what is implied if one does \( \alpha \) (other than what follows from \( \alpha \) itself) or if one does something other than \( \alpha \). No deontic markers necessarily follow from the performance or non-performance of \( \alpha \).

Note that the definition of permission does not specify a violation or fulfillment marker in the postcondition; permission is deontically underspecified. In this system, the notion of prohibition is basic, and the notions of obligation and permission are defined in terms of prohibition and the notion of not doing the action.

Given the single marker of violation, V does not differentiate among who executed which action with respect to which deontic specification; no distinctions can be made among what follows should a violation hold.
4.2.1.3 Deontic Operators on Sequences of Actions in ScDL and the CTD Structure

In DDL, deontic operators on complex action names are reduced to deontic operators on atomic action names. It is a *theorem* of DDL that a sequence of obligations, where one obligatory action follows another obligatory action, is equivalent to an obligation on a sequence of actions.

**Theorem 1**  \( OB(\alpha;\beta) \equiv OB(\alpha) \land [\alpha](OB(\beta)) \equiv [\alpha](V) \land [\alpha](\beta)(V) \)

[131] and [132] relate this theorem to the CTD structure, which can be expressed as follows:

**Example 95**

a. \( OB(\alpha) \)

b. \( [\alpha](OB(\beta)) \)

c. \( [\alpha](OB(\gamma)) \)

Where \( \alpha \) is executed, the primary obligation \( OB(\alpha) \) is *not* violated; in this subsequent state, \( OB(\beta) \) holds. Where an action among \( \beta \) is executed, \( OB(\alpha) \) is violated; in this subsequent states, \( OB(\gamma) \) holds. Thus, the reasoning in the CTD structure appears.

Note, following Theorem 1, that Examples (95a) and (95b) imply \( OB(\alpha;\beta) \).

4.2.2 Background for Problems with the CTD Structure for ScDL

Having given elements of ScDL, we introduce three problems. First, we show that in different cases, we get counterintuitive results (similar to our discussion of SwDL in chapter 3). We show how this problem is related partitioning of the action space. Finally, we demonstrate that a sequence of obligations is *not* equivalent to an obligation on sequences. Thus, the “solution” to the CTD problem in ScDL as in Example (95) is unsatisfactory.

While these subtopics have been discussed in some respects in the literature ([30], [130], [156], [174], [28], [29], [106], and [105]), the relationships between the subtopics have not been shown, the particular issues concerning action negation have not been addressed, and a distinction between OOS and SOO has not been clearly characterised.
We provide an intuitive discussion of the CTD problem in ScDL in terms of an example, provide the formal structure, and indicate what considerations we leave aside. These are followed by our discussions of the problems.

4.2.2.1 An Intuitive Example of the Basic CTD Problem in ScDL

Suppose we have the following set of statements, which all hold consistently in one state. We comment below on our choice of a CTD example.

Example 96

a. It is obligatory that Bill move the left toggle up.

b. If Bill does move the left toggle up, then it is obligatory that Bill move the right toggle left.

c. If Bill does not move the left toggle up, then it is obligatory that Bill move the right toggle right.

d. Bill does not move the left toggle up.

We assume that there are only two toggles and the only possible ways to move a toggle are up, down, left, or right.

There is a clear intuition that from (96d) and (96c), we can infer:

Example 97

It is obligatory that Bill move the right toggle right.

Additional intuitions are associated with Example (96). From Example (96a) and Example (96d), we may say that Bill has not done what he has been obligated to do. Let us assume that instead of moving the left toggle up, as he was obliged to do, he has moved the left toggle down. Thus, we can say that he has violated his obligation to move the left toggle up by moving the left toggle down. We then have the obligation to move the right toggle right as in Example (97), which is an obligation which is implied where someone has done something which has violated another obligation. We can say that Example (96a) is the primary obligation and Example (97) is a secondary obligation, for it arises relative to the
primary obligation. From execution of the action which gives rise to a violation, other consequences may follow.

We may consider that had Bill instead moved the left toggle up, as he was obligated to do, then it would follow, in such a circumstance, that he would be obligated to move the right toggle left. In addition, in this alternative circumstance, we would say that Bill had fulfilled his obligation. Consequences may follow from Bill’s having fulfilled his obligation such as he incurs a secondary obligation. In the alternative circumstance, we have a different secondary obligation. For our purposes, we do not discuss it further at the moment except to point out that different secondary obligations arise relative to the execution of actions which relate to violations or fulfillments of the primary obligation.

Finally, given Theorem 1, it follows that we have an obligation on a sequence:

**Example 98**

It is obligatory that Bill move the left toggle up, and then move the right toggle right.

### 4.2.2.2 A Formalisation of the Basic CTD Structure in SwDL

Consider an analysis of (96) in ScDL. Let us assume eight actions \( \alpha_U, \alpha_D, \alpha_L, \alpha_R \) and \( \beta_U, \beta_D, \beta_L, \beta_R \), which we associate with the different ways of moving the toggles, where \( \alpha \) is moving the left toggle, \( \beta \) is moving the right toggle, and U is up, D is down, R is right, and L is left. We can suppose that Bill is our agent, but abstract from this in our representations. Thus, \( \alpha_U \) is an abbreviation for Bill moves the left toggle up.

In order to represent our core issue in ScDL, we make assumptions about two issues which are discussed in greater depth in sections 4.4 and 4.5 as well as in chapter 5.

First, for our purposes, we have to assume the notion of the execution of an action, which appears in Example (99d). In ScDL, we have action names, such as \( \alpha_U \), and modal propositions formed using an action name, such \([\alpha_U](\phi)\), which intuitively mean that were action \( \alpha \) to be executed in a state where the preconditions of \( \alpha_U \) hold, then in the state which would result from the execution of the action, \( \phi \) must hold. Notice that \( \alpha_U \) is an action name and \([\alpha_U](\phi)\) does not specify that the action is executed, just that were it to be executed, what must hold in the resultant state. Thus, we can reason abstractly about actions. Yet, there is a distinction between reasoning about what would hold conditionally from what does hold. Indeed in ScDL, the execution of an action is not expressible, so Example (99d) is introduced for our purposes to reason clearly about what does hold as
a consequence of the execution of the action. In chapter 5, we express execution of an action. Our assumption here is for the sake of discussion; we assume some intuitive notion that an action such as $\alpha U$ is executed in some state and results in some state.

Second, we assume that $\pi U$ denotes the set of actions of the domain of actions other than $\alpha U$, namely the other seven actions. We assume that the expression execute an action that is an element of $\pi U$ picks out and executes one of these seven actions. We discuss this further in section 4.5 and in chapter 5. With these assumptions, we have the following formalisation.

**Example 99**

a. $OB(\alpha U)$

b. $[\alpha U](OB(\beta L))$

c. $[\pi U](OB(\beta R))$

d. Execute an action which is an element of $\pi U$.

Given that an action in $\pi U$ is executed, it follows that in the subsequent state, a violation marker holds which represents the violation of $OB(\alpha U)$. As some action in $\pi U$ has been executed, in this subsequent state, $OB(\beta R)$ holds.

We assume a notion of inertia (or persistence), which we discuss more specifically in chapter 5. For our purposes here, inertia means that if some property or proposition holds of the state before the execution of some action, then the property or proposition holds of the state after the execution of the action, unless the action directly and explicitly affects the property or proposition. In Example (99), the obligation $OB(\alpha U)$ in Example (99a) holds of the state before the execution of whatever action given in Example (99d). Since this action does not specifically negate $OB(\alpha U)$ and there is no other reason why $OB(\alpha U)$ is inconsistent with other properties or propositions of the state, we presume that $OB(\alpha U)$ persists after the execution of the action and holds of the state after the execution of the action. Note that we make no assumptions in this chapter concerning the relationship between the persistence of an obligation and whether the execution of an action has fulfilled or violated the action; resolution of this issue is not central to our concerns in this chapter. Issues relating to the management of deontic specifications (i.e. the introduction and elimination of expressions of the form $OB(\alpha U)$ are discussed further in section 4.4.2.4 and further in chapter 5. By the same token $[\alpha U](OB(\beta L))$ and $[\pi U](OB(\beta R))$ persist as well. The manipulation of action expressions are discussed
further in chapter 5. For our purposes in this chapter, we assume inertia as we have outlined it.

4.2.2.3 Structures Not Considered

[30], [132], and [62] discuss a range of CTD cases. The one we consider here is referred to as the forward version of the Chisholm set by [132]. What they call the parallel version incorporates the Gentle Murderer Paradox and the backwards version incorporates sequence of tense issues. As remarked in chapter 3, we are not considering examples with adverbial modification, sequence of tense, as well as obligations on stative expressions. In any case, however these issues are treated, they are not clearly related to our central investigation.

4.2.3 The Problems with the Analysis of the CTD Structure in ScDL

At this point, let us consider our problems.

4.2.3.1 Problem One: Changing the Primary Obligation

Let us consider the problems which arise when we change the primary obligation. We start with an informal discussion to clarify the intuitions, then follow with a formalisation.

4.2.3.1.1 Informal Presentation

Consider the following set of expressions:

Example 100

a. *It is obligatory that Bill move the left toggle down.*

b. *If Bill does move the left toggle up, then it is obligatory that Bill move the right toggle left.*

c. *If Bill does not move the left toggle up, then it is obligatory that Bill move the right toggle right.*

d. *Bill does not move the left toggle up.*

On the model of reasoning above, we would conclude that Example (101) holds.
Example 101

*It is obligatory that Bill move the right toggle right.*

The problem is not with drawing the inference, but with how we consider it in light of the theories we have. In particular, from the theory, we make inferences which do not seem intuitively plausible; thus, we could characterise them as paradoxes.

Consider that Bill’s not moving the left toggle up could be one of a variety of actions: for example, he moves the left toggle down, or he moves the left toggle left. When he moves the left toggle down, he has fulfilled the primary obligation. However, when he moves the left toggle left, he has violated the primary obligation since by hypothesis of the meaning of obligation and action negation, doing anything other than moving the left toggle down leads to a violation.

In addition, in that circumstance where Bill moves the left toggle down, we would then satisfy the requirements of Theorem 1, namely, that the following obligation on a sequence holds as well:

Example 102

*It is obligatory that Bill move the left toggle down, and then move the right toggle right.*

There are two problems. First, it intuitively seems wrong to say of Example (100) that violations or fulfillments of the primary obligation arise, particularly in comparison with Example (96), where it seems plausible that they do arise. The relationship between the primary obligation, the action executed, and the secondary obligations is somehow disrupted in Example (100). Second, while it is disputable whether we must accept the inference to the obligation on a sequence in Example (98), it is unacceptable in Example (102). We discuss each of these in a formal analysis.

4.2.3.1.2 Formalisation of Problem One  We formalise the CTD structure in Example (100) as:
Example 103

a. \( OB(\alpha D) \)

b. \([\alpha U](OB(\beta L))\)

c. \([\bar{\alpha} U](OB(\beta R))\)

d. Execute an action which is an element of \( \bar{\alpha} U \).

After execution of an action from \( \bar{\alpha} U \), we may infer that \( OB(\beta R) \) holds. As \( \alpha D \) is an element of \( \bar{\alpha} U \), the action executed could be \( \alpha D \), in which case, we infer that the primary obligation has been fulfilled. Since \( \bar{\alpha} D \) and \( \bar{\alpha} U \) intersect, some action other than \( \alpha D \) could be executed, for example \( \alpha U \), in which case the primary obligation has been violated; indeed, any action other than \( \alpha D \) gives rise to a violation marker. Thus, \( OB(\beta L) \) arises in states which are marked as violations relative to the primary obligation, while \( OB(\beta R) \) can arise either in a state which is marked either as a violation or as a fulfillment relative to the primary obligation. Moreover, since \( \alpha D \) is an element of \( \bar{\alpha} U \), we can infer that after \( \alpha D \) is executed \( OB(\alpha D; \beta R) \) holds.

As we pointed out intuitively in the first problem, it seems intuitively unreasonable to infer that the states, no matter how they arise, are definitely marked with either violation or fulfillment relative to the primary obligation. Rather, our reaction is that we cannot say anything about violation or fulfillment in these cases. Nor, by the same token, does it seem correct to claim that secondary obligations arise relative to the primary obligation, though it does seem right to say what follows from the execution of one action or the other. Even where we execute an action which violates the primary obligation, does this imply that the secondary obligation is a CTD obligation? Finally, as mentioned in the second problem, it seems intuitively odd to infer that we have an obligation on a sequence \( OB(\alpha D; \beta R) \) where \( \alpha D \) is executed. Both problems are similar to our problems with the SwDL analysis of CTDs.

The general issue is not with the logic per se, but that the logic does not correlate with our intuitions; the logic allows us to draw definite inferences which do not seem intuitively plausible. As an abstraction, the logic may serve a purpose, but as a model of natural legal reasoning, it seems to be overdetermined. It appears that the link between the primary and secondary obligations as well as the violation marker in Example (99) has been broken in Example (103). We see that the issue arises where we change primary deontic specifications, a problem prior proposals for SwDL or ScDL have not previously accounted for.
4.2.3.2 Problem Two: Partitioning the Action Space

The reason the structures Example (99) and Example (103) are treated alike is related to the treatment of action negation and the partitioning of the action space.

Suppose one obligation and four ways of moving the left toggle.

Example 104

*It is obligatory that Bill move the left toggle up.*

Given the definition of obligation and our assumption of action negation, it seems clear that Bill’s moving the left toggle up *fulfills* the obligation, while *any other action – moving it down, or left, or right* – *violates* it. In other words, no matter what Bill does, he either violates or fulfills his obligation. By the same token, as soon as Bill *incurs* his obligation, and supposing he must do something, then he induces either a violation or a fulfillment. Similarly, were Bill to bear two obligations each on a different action which could not both be simultaneously satisfied, then he is sure to violate one or the other, for the action which fulfills one obligation would also be an action that violates the other. While in an abstract and theoretical domain, these conclusions might not seem unreasonable, in any application or for real world reasoning, they are untenable.

The underlying reason for the problem is that, relative to a particular obligation, one can consider the domain of actions *partitioned* between those actions which fulfill the obligation (the explicitly given action) and those which violate it (the complement set of the given action) ([130] and [156]). There are no actions which are *underspecified* with respect to the obligation such that executing that action induces neither a violation, nor a fulfillment.

Consider a case where one *would* want such deontically underspecified actions. Suppose a pizza deliverer has a contract which specifies that he is obligated to deliver pizzas for six hours once a week. Let us consider one of those weeks. As we want to *enable* our deliverer to meet the obligation, we assume, *ceteris paribus*, the requisite conditions are satisfied such that he is able to meet his obligation: the pizzas are baked, weather conditions permit delivery, and so on. Delivering (enough) prepared pizzas over the six hours fulfills the obligation, while delivering no (or not enough) prepared pizzas in the six hours violates it. Clearly, there are subtle issues here concerning contextually specified values for “enough”, but our point is the following. We do not want just *any* action other than those which *specifically* fulfill the obligation to *instead* violation it. More broadly, as recognized in
jurisprudential theory [144], it is in practice unreasonable that every possible action is legally defined for a given domain. For example, eating an apple, going to the toilet, or many other actions would seem to be deontically underspecified. We want a more refined abstract analysis.

4.2.3.3 Problem Three: The difference between Obligations on Sequences and Sequences of Obligations

Let us return to the claim concerning Theorem 1 which determines the relationship between an obligation on a sequence (OOS) as in (105a) and a sequence of obligations (SOO) as in (105b).

Example 105

a. It is obligatory that Bill move the left toggle up, then move the right toggle left.

b. It is obligatory that Bill move the left toggle up. After having moved the left toggle up, it is obligatory that Bill move the right toggle left.

Theorem 1 ([130] and [156]) proves that Examples (105a) and (105b) are equivalent in DDL and so mean the same thing. In contrast, [106] claim they are intuitively distinct – an obligation on a sequence is an obligation on an action which is distinct from its components; [106] introduce a deontic operator for an obligation on a sequence which is not equivalent to deontic operators on the parts. However, [106] do not discuss the problem it introduces for [130], elaborate on the intuition, nor provide a full formal analysis. We are unaware of any subsequent discussion of the distinction.

4.2.3.3.1 Introduction or Elimination of Obligations

One argument for the claim that OOS and SOO are not equivalent is the following. Suppose an OOS, $\text{OB}(\alpha;\beta)$, implies an SOO, $\text{OB}(\alpha) \land [\alpha](\text{OB}(\beta))$. As we discuss in section 4.4.2.4, we can provide rules which introduce or eliminate deontic expressions in a state-wise analysis. Along these lines, we can separately stipulate that after the execution of $\alpha$, $\text{OB}(\beta)$ does not hold; that is, the execution of $\alpha$ removes the obligation on the execution of $\beta$. Then OOS, which implies that $\text{OB}(\beta)$ must hold after the execution of $\alpha$ is not consistent with the stipulation that it does not. The problem is that OOS requires an obligation to do $\beta$ follows from the execution of $\alpha$, which is not necessary.
4.2.3.3.2 Different Implications from Different Violations

Another way to see that an OOS and a SOO are not equivalent is to consider the violation conditions and what follows from them. In the ScDL of [130] and [106], there is no distinction between violation markers. Thus, no distinction between violating an OOS and violating an SOO.

However, this is not intuitively so. In an obligation on a sequence, there are different actions that can be executed, but which violate one and the same obligation – that obligation on the sequence per se. In contrast, in an sequence of obligations, the same actions can be executed, but which intuitively violate distinct obligations – those on each of the separate actions. The ScDL analysis with one violation marker cannot make this distinction. Here we provide examples which highlight these different intuitions; we present our formal analysis in section 4.3.

4.2.3.3.3 Examples Illustrating the Differences between OOS and SOO

To highlight the intuitions, consider the consequences which follow from the violations or fulfillments. If OOS and SOO were equivalent, then it ought not to be possible to provide different circumstances where we find different implications.

Suppose the following, where we have co-indexed expressions of the form It is obligatory $P$, for $P$ a proposition, and the deictic expression “this obligation” to make it clear exactly which obligation we are referring to. We suppose that the statements in Example (106) hold in one state and those in Example (107) hold in another. Suppose that given a particular fulfillment or violation implies that Bill gets a particular colored chip. These sets of statements are coherent and consistent within a state, but under no executions of actions do the implications in one state imply those in another state as shown in [193].

Example 106

a. [It is obligatory that Bill move the left toggle up.]
   i. If he fulfills [this obligation], then he gets a blue chip.
   ii. If he violates [this obligation], then he gets a red chip.

b. [It is obligatory that Bill move the right toggle left.]
   i. If he fulfills [this obligation], then he gets a green chip.
   ii. If he violates [this obligation], then he gets a yellow chip.
Chapter 4

4.3 A Proposal for Fine-grained Violation and Fulfillment Markers

Example 107

a. [It is obligatory that Bill move the left toggle up, and then the right toggle left.]_k

i. If he fulfills [this obligation]_k, then he gets a white chip.

ii. If he violates [this obligation]_k, then he gets a black chip.

Thus, we have a concrete, natural language counterexample to the presumed equivalence between OOS and SOO.

4.2.4 Comparisons between SwDL and ScDL

[30] and [130] do not agree in how to analyse the deontic concepts. [30] claim that DDL does not handle deontic specifications on static expressions (but see [46] and [191] for solutions). [130] argues that SDL uses the problematic concept of ideality, incurs a host of paradoxes, and cannot accommodate state change. The claim is that ScDL does not suffer the paradoxes of SwDL because the deontic operators apply to actions, not propositions. Therefore, the problems which arise in SwDL due to inferences in Propositional Logic do not appear in ScDL. We agree with [130] on these points. However, we believe that [30] maintain a key insight that is obscured in ScDL, namely a reference to the context-sensitivity of secondary obligations. Moreover, both SwDL and ScDL need to represent complex violation and fulfillment markers.

4.3 A Proposal for Fine-grained Violation and Fulfillment Markers

To address the problems outlined above, we propose a reductionist analysis of the deontic operators on actions which provides fine-grained violation and fulfillment markers. It is not a logic, though we do provide a semi-formal analysis, which is made more explicit and rigorous in the implementation in chapter 5. In addition, we should note that issues related to action negation are discussed separately in section 4.5.

In the following sections, we introduce the basic elements of the complex markers. We present linguistic examples to support the interpretation. Then, the analysis is given a semi-formal presentation. Alternative definitions for obligations with respect to sequences are provided. We present how our analysis solves basic issues for the CTD structure.
in ScDL. Comparable analyses are outlined. We end with a discussion of a range of outstanding issues.

### 4.3.1 Basic Elements of Fine-grained Markers

We draw forward from earlier sections the definitions for the obligation operator on actions and for the obligation operator on sequences of actions.

**Definition 10**

\[ a. \ OB_{\alpha} \equiv [\alpha](V) \]
\[ b. \ OB_{(\alpha; \beta)} \equiv [\alpha](V) \land [\alpha](\beta)(V) \]

In contrast to these definitions, in addition to marking for a violation after failing to execute the obligated action (V), we mark as well for fulfilling the execution of the action (F). More importantly, in addition to marking for fulfillment or violation, we mark what the deontic operator was, the action which was deontically specified, and the complex action combinator (if any) that combines the actions. We do this by *subscripting* the V and F propositions.

Thus, rather than Definitions (10a) and (10b), we have Definitions (11a) and (11b).

**Definition 11**

\[ a. \ OB_{\alpha} \equiv [\alpha](F \ OB_{(\alpha)}) \land [\alpha](V \ OB_{(\alpha)}) \]
\[ b. \ OB_{(\alpha; \beta)} \equiv [\alpha](V \ OB_{(\alpha; \beta)}) \land [\alpha]([\beta](F \ OB_{(\alpha; \beta)}) \land [\beta](V \ OB_{(\alpha; \beta)})) \]

Below we given the intuition behind these propositions; in section 4.3.3, we provide a semi-formal interpretation of these subscripted propositions.

### 4.3.2 Intuitive Interpretations of the Complex Propositions

We read the expressions on the right of the equivalence as follows, respectively:
Chapter 4  4.3 A Proposal for Fine-grained Violation and Fulfillment Markers

Example 108

a. Doing $\alpha$ results in a fulfillment of the obligation on the execution of $\alpha$ and doing an action from among $\bar{\alpha}$ results in a violation of the obligation on the execution of $\alpha$.

b. Doing an action from among $\bar{\alpha}$ results in a violation of the obligation on the execution of the sequence action $\alpha; \beta$. Doing $\alpha$ gives rise to a circumstance where doing $\beta$ results in a fulfillment of the obligation on the execution of the sequence action $\alpha; \beta$, and where doing an action from among $\bar{\beta}$ results in a violation of the obligation on the execution of the sequence action $\alpha; \beta$.

While the V marker in Definition (10) is to be understood as violation, the complex marker in (11a) is understood as a violation of the obligation on the execution of action $\alpha$. Similarly, where F stands for a fulfillment marker, the complex marker in (11a) is understood as a fulfillment of the obligation on the execution of action $\alpha$. More central to our proposal is the definition in (11b), where we find fulfillment or violation of the obligation on the execution of the sequence which is comprised of executing first $\alpha$ followed by $\beta$. While we also claim that there are alternative formulations of obligations on sequences, these are only discussed further in section 4.3.3.3.

Let us clarify these complex markers with reference to natural language expressions. Simply put, the markers are the formal analogs to the natural language expressions of the violation and fulfillment. Here we allow agents, such as Bill, actions such as move the left toggle up.

Example 109

a. Bill is obligated to move the left toggle up.

b. Were Bill to move the left toggle down, a violation of an obligation with respect to Bill on the movement of the left toggle up would be recorded. Were Bill to move the left toggle up, a fulfillment of an obligation with respect to Bill on the movement of the left toggle up would be recorded.

We decompose the verbal form of Example (109a) into its correlated nominal form in Example (109b). The key task is to provide a systematic decomposition along these lines.
4.3 A Proposal for Fine-grained Violation and Fulfilment Markers

4.3.3 A Semi-formal Analysis of Complex Violation and Fulfilment Markers

We break the presentation of the analysis down into several subsections. First, we present an analysis of deontic operators as they apply to atomic actions. We then discuss deontic operators on complex actions.

4.3.3.1 Deontic Operators and Atomic Actions

To decompose and then formalise the analysis, we postulate several sorts of abstract objects: deontic specifications, deontic operators labels, action labels, agent labels, and deontic flag labels. The deontic specifications are functionally related to these other objects. The noun phrase in Example (110a) is expressed as Example (110b), where we just consider the violation marker to simplify the presentation. The have relation is expressed functionally below.

Example 110

a. A violation of an obligation with respect to Bill on the movement of the left toggle up.

b. There is a deontic specification which has a deontic operator object obligation, an action object move left toggle up, an agent object Bill, and a deontic flag object violation.

Note that the labels are systematically related to but distinct from the expressions they are derived from; that is, while we have $\text{OB}_\alpha$ on the left side of the equivalence in Definition (11a), the same form on the right side which appears as a subscript to $V$ as in $V_{\text{OB}_\alpha}$ is a nominalised correlate. Indeed, $V_{\text{OB}_\alpha}$, where $\alpha$ is Bill moving the left toggle up, is an abbreviation for Example (110b). As we adopt a reductionist analysis, $\text{OB}_\alpha$ on the left side of the equivalence is reduced to the complex expression on the right side, which has no further reduction.

4.3.3.1.1 The Component Elements

Having presented the intuition behind the analysis in natural language, let us return to represent our more abstract example in Definition (10a), though we add the agent so as to more closely relate our definitions with
4.3 A Proposal for Fine-grained Violation and Fulfillment Markers

our examples. While we leave to chapter 5 several of details of our analysis, we sketch the main elements here. We suppose the following disjoint sets of objects:

a. DS, the set of deontic specifications.

b. DOL, the set of deontic operator labels, of which “obligation”, “permission”, and “prohibition” are the only individuals.

c. AcL, the set of action labels, of which “α”, “β”, and “α;β” are individuals (among others).

d. AgL, the set of agent labels, of which “Bill” and “Jill” are individuals (among others).

e. DFL, the set of deontic flag labels, of which “violation”, “fulfillment”, and “null” are the only individuals.

We assume that ds1 and ds2 are variables of sort DS. In addition, suppose the following functions:

a. deonticOpF, a function from objects in DS to objects in DOL.

b. actionF, a function from objects in DS to objects in AcL.

c. agentF, a function from objects in DS to objects in AgL.

d. deonticFlagF, a function from objects in DS to objects in DFL.

With these objects and functions, we represent Example (110b) as:

Example 111

\[ \exists x \in DS \ [ \text{deonticOpF}(x) = \text{“obligation”} \land \text{actionF}(x) = \text{“α”} \land \text{agentF}(x) = \text{“Bill”} \land \text{deonticFlagF}(x) = \text{“violation”}] \]
4.3.3.1.2 Definitions of the Deontic Operators

Our deontic operators are defined in terms of such deontic specifications. We use $\text{OB}_{\text{atomic}}$ to indicate the obligation operator on atomic actions. Obligation operators on complex actions are discussed below.

**Definition 12**

$$\text{OB}_{\text{atomic}}(\alpha, \iota) = \text{def}$$

$$[\alpha](\exists ds1 \ (\text{deonticOp}(ds1) = \text{“obligation”} \land \text{action}(ds1) = \alpha \land \text{agent}(ds1) = \iota \land \text{deonticFlag}(ds1) = \text{“fulfillment”})) \land$$

$$[\overline{\iota}](\exists ds2 \ (\text{deonticOp}(ds2) = \text{“obligation”} \land \text{action}(ds2) = \alpha \land \text{agent}(ds2) = \iota \land \text{deonticFlag}(ds2) = \text{“violation”})),$$

for $\alpha \in AcL$, $\iota \in AgL$, $ds1$ and $ds2$ are variables with values in DS, and $ds1 \neq ds2$.

In subsequent definitions and examples, we assume the relevant sorts and restriction which are provided in the last line. We understand that the agent $\iota$ which is obligated with respect to the action $\alpha$ is the bearer of violation or fulfillment. Notice the difference between which action is executed and the action associated with a violation or fulfillment marker. In particular, where $\alpha$ is executed, the violation marker indicates that the obligation was on the execution of $\alpha$ rather than (an action among) $\alpha$. The rationale is that we keep track of the information about the source obligation; that is, it is an obligation on the execution of $\alpha$.

We do not, therefore, take the markers in Definition 11 to be atomic propositional symbols of which there are as many as there are distinct propositions which the deontic operators apply to. Rather, the markers in Definition 11 are abbreviations of these complex expressions (though missing the agent).

As the violation and fulfillment markers are constructed in this way, we can see systematic relationships between them. For instance, we have inference. In ScDL, for any action label $\alpha$, $[\alpha](\phi \land \psi)$ implies $[\alpha](\phi)$. Thus, we have the following implication:

**Property 1**

$$\text{OB}_{\text{atomic}}(\alpha, \iota) \rightarrow [\overline{\iota}](\exists ds2 \ (\text{deonticFlag}(ds2) = \text{“violation”}))$$

We take the consequent as the ScDL expression $[\overline{\iota}](V)$ in Definition (10a). Thus, our analysis subsumes the standard ScDL analysis.
By the same token, we can represent **prohibitions** and **permissions**. For **prohibition**, the execution of the action gives rise to a marker of violation. We could add the clause which indicates a fulfillment of such a prohibition, but as we have pointed out in chapter 2, it is not clear we need this. For reasons we discuss later, we assume that for **permission**, execution of the action or its negation gives rise to a marker which does not indicate violation or fulfillment, but “null” instead. We provide these definitions for illustration.

**Definition 13**

\[
PR_{\text{atomic}}(\alpha, t) \overset{\text{def}}{=} \\
[\alpha](\exists ds_1 (\text{deonticOpF}(ds_1) = \text{“prohibition”} \land \text{actionF}(ds_1) = \alpha \land \text{agentF}(ds_1) = \text{“i”} \land \text{deonticFlagF}(ds_1) = \text{“violation”}))
\]

Our definition of the **permission** operator is as follows. Note that it uses a “null” label as a deontic flag, which we take as a way to represent that no violation or fulfillment flag is indicated.

**Definition 14**

\[
P_{E\text{atomic}}(\alpha, t) \overset{\text{def}}{=} \\
[\alpha](\exists ds_1 (\text{deonticOpF}(ds_1) = \text{“permission”} \land \text{actionF}(ds_1) = \alpha \land \text{agentF}(ds_1) = \text{“i”} \land \text{deonticFlagF}(ds_1) = \text{“null”}) \land [\alpha](\exists ds_2 (\text{deonticOpF}(ds_2) = \text{“permission”} \land \text{actionF}(ds_2) = \alpha \land \text{deonticFlagF}(ds_2) = \text{“null”}))
\]

### 4.3.3.2 An Extensible and Flexible Framework

We should again emphasise that our point is not to provide definitive semantic interpretations of the deontic operators, but rather to provide a means to express alternative formalisations which represent different intuitions of how the deontic operators apply to atomic and complex actions. We have claimed that the point of our study is to provide a tool with which one could experiment with alternative definitions of the deontic concepts which would suit different interpretations or purposes, yet could be compared and contrasted.

One could define alternative or additional interpretations of the deontic operators in this framework. For instance, the approach is clearly extensible and flexible, as one can add...
other sorts of objects which are related to the deontic specification by additional functions or relations.

With this in mind, we mention additions. For example, if one wanted to identify the object of the action of the deontic specification, then we would be able to differentiate violations on moving left toggles versus right toggles. Similarly, we could identify the direction of that object to mark violations on moving things up or down. While additional functionality would have to be incorporated into the system, one could provide a means to represent the violation or fulfillment of a deadline by which the obligation must be met. The additional functionality would have to compare the current time of the execution of the action against this deadline time.

As we show in the next section, the framework allows alternative definitions of obligations on sequences, which is another way that the basic framework above can be extended.

### 4.3.3.3 Alternative Definitions of Obligations on Sequences

So far, we have only considered the violation markers on atomic actions. Let us consider the violation markers for complex actions such as sequences in Definition (11b), which takes us closer to our central topic of CTDs, obligations, and sequences. Following on from our previous considerations, suppose that among the sort of labels for actions, we also have those which represent complex actions. Here we consider only sequence: where the complex action is \( \alpha;\beta \), its correlated nominal is “\( \alpha;\beta \)” This is based on the natural language relationship between the verbal form in (112a) and nominal form the sequence with reference to its parts (112a) in:

**Example 112**

\[
\begin{align*}
a. & \quad \text{Bill moved the left toggle up, and then Bill moved the right toggle left.} \\
b. & \quad \text{The sequence comprised of Bill’s moving the left toggle up followed by Bill’s moving the right toggle left.}
\end{align*}
\]

An obligation on this sequence would appear as:

**Example 113**

\[
\text{It is obligatory that Bill move the left toggle up, and then Bill move the right toggle left.}
\]
As we discuss further below, there are several alternative interpretations of the obligation on such a sequence. We first provide our various interpretations, and then justify them.

**4.3.3.3.1 Interruptable Obligations** One interpretation we call the *interruptable* obligation on a sequence, represented with $OB_{int}$. One interpretation of Definition (11b) is as an abbreviation of:

**Definition 15**

\[
OB_{int}(\langle \alpha;\beta \rangle;\iota) =_{def} \\
[\alpha](\exists ds3 \ (\text{deonticOpF}(ds3) = \text{“obligation”} \land \text{actionF}(ds3) = \alpha;\beta \land \text{agentF}(ds3) = \iota \land \text{deonticFlagF}(ds3) = \text{“violation”})) \land \\
[\iota](\exists ds4 \ (\text{deonticOpF}(ds4) = \text{“obligation”} \land \text{actionF}(ds4) = \alpha;\beta \land \text{agentF}(ds4) = \iota \land \text{deonticFlagF}(ds4) = \text{“fulfillment”})) \land \\
[\beta](\exists ds3 \ (\text{deonticOpF}(ds3) = \text{“obligation”} \land \text{actionF}(ds3) = \alpha;\beta \land \text{agentF}(ds3) = \iota \land \text{deonticFlagF}(ds3) = \text{“violation”}))
\]

This says that doing $\overline{\alpha}$ rather than $\alpha$, the first action of the sequence, gives rise to a marker of a violation of one’s obligation on executing the *sequence*. If one does $\alpha$, then there are two courses of action: either one executes the next action in the sequence $\beta$, which gives rise to a marker of one’s fulfillment of one’s obligation on executing the sequence; or, one executes an action other than $\beta$ giving rise to violation of the obligation on the sequence. In other words, there is one way to fulfill the obligation on the sequence and two ways to violate it.

We call this the *interruptable* interpretation because it is possible that there are actions other than $\beta$ which can be executed between the execution of $\alpha$ and $\beta$, yet which do not give rise to violation. However, this depends on how one interprets action negation, which we discuss further in section 4.5. In particular, let us suppose for the moment that the negation of an action need not be the set-theoretic complement of actions, but some subset of the complement set which we call the *antonyms*. To be concrete, suppose that there are five actions in our domain: $\alpha$, $\beta$, $\gamma$, $\delta$, and $\sigma$, where $\alpha$ and $\gamma$ are antonyms and $\beta$ and $\delta$ are antonyms. Given this, suppose an agent bears the obligation on the sequence in Definition (15). The agent executes the first action in the sequence $\alpha$. In this state, neither a fulfillment nor violation of the obligation arise since these only arise after the execution of $\beta$ or its antonym $\delta$. The agent could execute $\sigma$ *without incurring a fulfillment*...
or violation marker. This is the case where the agent can execute other actions between the actions of the sequence and which have no bearing on the obligation.

We think this is the most natural interpretation of an obligation on a protocol for human behaviour since it is unreasonable to expect all actions in a given state to give rise to violation or fulfillment; that is, there may be obligatory sequences on actions which allow some actions to intervene without being considered to give rise to a violation. For example, where a pizza deliverer is obligated to take a pizza freshly made to order and deliver it to the address on the order, between taking the pizza and delivering it, there may be a range of actions the deliverer executes which do not give rise to a violation on the obligation on the sequence.

The interruptable interpretation may be appropriate for certain interpretations. However, it is rather essential that we be careful which expressions under what interpretation we are considering. For example,

**Example 114**

\[ a. \text{Bill is obligated to kill and after to run away from the country.} \]
\[ b. \text{Bill is obligated to put the letter in the mail and after to phone saying that he has already posted the letter.} \]

As part of a job specification, it is plausible to interpret these as obligations on sequences. But, another equally plausible interpretation has the temporal clauses and after... semantically appearing outside the scope of the deontic operator. This is much like our analysis of the Good Samaritan Paradox in chapter 3. Thus, these are not counterexamples to our analysis.

**4.3.3.2 Collective Obligations** An alternative to the interruptable interpretation of obligation is what we call a collective interpretation, where the action which is executed is the sequence itself as one uninterruptable chain of actions, as if it were an action executed in one step.\(^1\)

\(^1\)The intended interpretation is distinct from an obligation that is collective with respect to agents ([60]), which we do not discuss.
Chapter 4  4.3 A Proposal for Fine-grained Violation and Fulfillment Markers

Definition 16

\[ OB_{col}((α;β),ι) =_{def} \]
\[ [α;β](∃ds3 (deonticOpF(ds3) = “obligation” ∧ actionF(ds3) = “α;β” ∧ agentF(ds3) = “ι” ∧ deonticFlagF(ds3) = “violation”)) \]
\[ [α;β](∃ds4 (deonticOpF(ds4) = “obligation” ∧ actionF(ds4) = “α;β” ∧ agentF(ds4) = “ι” ∧ deonticFlagF(ds4) = “fulfillment”)) \]

Here, the sequence can be regarded as a transaction which has no “visible” subactions and cannot be interrupted, following [106]. We take a transaction to be an action of the same type as the basic actions. As such, they can be negated as in α;β. A question here is What actions are the negation of a sequence? As we discuss later, action negation is itself a significant issue. We have supposed a notion of action antonym, which is a specified subset of actions. The negation of complex actions raises a central issue, particularly for an implementation: how can one find in the lexical space of actions the antonym of a sequence, where a sequence is comprised of any two arbitrarily sequentiable actions? With the atomic actions, it is not unreasonable to assume that the antonym of an action is stipulated. For complex actions, we must develop functions which calculate the antonym from the input actions and the action combinator. We take this up further in the implementation, where we solve this problem with lexical semantic functions.

4.3.3.3.3 Distributive Obligations  Yet another interpretation is to allow the obligation operator to distribute over the actions in the sequence, assuming our interpretation of OB in Definition (12):

Definition 17

\[ OB_{dist}((α;β),ι) =_{def} \]
\[ (OB_{atomic}(α,ι) ∧ [α](OB_{atomic}(β,ι))) \]

Indeed, an interpretation along these lines is found in both [130] and [106]. The intuition is that if one is obligated to move the left toggle up and then the right toggle left, one has

\[^2\text{An interesting alternative is to allow the distributive obligation operator to apply to β if β has the from of β1;...;βn. However, the definition is a bit more complex since we want the distributive interpretation where the action is complex and the simple interpretation where the action is not complex. We have opted for the simpler expressions for the purposes of illustration.}\]
an obligation to move the left toggle up, and after having done so, an obligation to move the right toggle left. These are separate obligations that are fulfilled or violated in a given order. However, note that violating one or the other obligation does not signal a violation of the sequence per se.

We have several alternative interpretations of obligations with respect to sequences. Several of these interpretations clearly have plausible natural language correlates, while others may seem more questionable, though at some other point a use may be found for them. However, as we have pointed out, our objective is to provide for a tool which allows for a range of expressions in order to study them in the implementation.

### 4.3.4 Solutions to Problems in the CTD Structure

Having discussed several problems which arise with ScDL accounts of CTD cases, let us provide our solution to them using complex violation and fulfillment markers. First, in contrast to [130] (or by the same token [30]), secondary obligations do not follow from actions (or propositions) directly, but rather from the articulated violation or fulfillment markers themselves; we make the markers syntactically active in the language.

Rather than representing Example (96) as Example (99), we represent it as follows:

**Example 115**

a. $\text{OB}(\alpha U)$

b. $F_{\text{OB}_a U} \rightarrow (\text{OB}(\beta L))$

c. $V_{\text{OB}_a U} \rightarrow (\text{OB}(\beta R))$

d. Execute an action which is an element of $\overline{\alpha} U$.

The significant move here is that rather than representing the action or proposition which gives rise to the secondary obligation as in Example 96 and Example 99, we reason with the representation of the violation or fulfillment markers. We are making explicit the information which was implicit, but which is the most significant. It is not simply that a particular action was executed, but rather that the action is related to the deontic specification in the state and gives rise to a violation or fulfillment marker from which a secondary obligation follows.
If one of the actions in $\alpha U$ is executed, the violation flag of $\text{OB}(\alpha U)$ holds in the subsequent state. The implication in Example (115c) applies, and we infer $(\text{OB}(\beta R))$; if, in an alternative state, $\alpha U$ is executed, then the violation flag of $\text{OB}(\alpha U)$ holds in the subsequent state and implies $\text{OB}(\alpha L)$. These observations are consistent with intuitions.

Furthermore, the analysis addresses a problem discussed earlier: since Example 115 has violation and fulfillment markers rather than actions, we cannot derive an OOS, obligation on a sequence, from it.

Let us consider the other problematic case, where we changed the primary obligation as in Example (103).

**Example 116**

\begin{itemize}
  \item \textbf{a.} $\text{OB}(\alpha D)$
  \item \textbf{b.} $F \text{OB}_{\alpha U} \rightarrow (\text{OB}(\beta L))$
  \item \textbf{c.} $V \text{OB}_{\alpha U} \rightarrow (\text{OB}(\beta R))$
  \item \textbf{d.} Execute an action which is an element of $\alpha U$.
\end{itemize}

As before, the sequence of obligations does not imply an obligation on a sequence. More interesting is that no matter what action is executed with respect to this set of expressions, no secondary obligation arises. This accords with our intuition. The reason is that any action which is executed gives rise to a violation or fulfillment marker relative to the primary obligation. These markers indicate a violation or fulfillment relative to the action $\alpha D$. However, these markers are not the antecedents of either Example (116b) or Example (116c), which are markers relative to $\text{OB}(\alpha U)$ and which does not appear in this set of expressions. We still need to address issues related to action negation, but because we are using explicit violation and fulfillment markers which give rise to the secondary obligations, it does not have a bearing on the problem. We see this as an important advantage.

We have argued that articulated violation and fulfillment markers allow a more accurate analysis of CTD cases in dynamic theories. In our view, the goal is to make inferences to secondary obligations depend on properties of the state which are ascribed relative to the primary obligation. Informally, it is not an action itself which induces the secondary obligations, but the action in relation to its deontic specification.

In the following subsection, we briefly mention several related proposals and highlight how they are different from our proposal.
4.3.5 Comparisons

As we have already pointed out, the presence of violation markers for deontic operators is not itself a novel idea in both the SwDL and ScDL approaches. In the earlier literature on violation markers ([102], [7], [6], [130], [106], and [105]), we find violation marked with an atomic proposition. In [99] and [30], violation markers appear alongside ideality; however, the relationship between ideality and the violation marker is not entirely clear. More articulated versions of the violation marker have appeared in several previous works. [186] and [187] specify violations relative to the relation (a verb) and its arguments (i.e. subject and object). In [46], the marker is indexed with a number. In [132], the marker is informally indexed with the action relation. In [174], agents and actions are indexed. [62] also allows fulfillment markers. In [29], there are violation markers indexed with respect to the deontic operator: \( V_{PR} \) for violation of a prohibition, \( V_{OB} \) for violation of an obligation, and \( V_{PE} \) for lack of an explicit permission. [162] have a judge logic where violations are identified; however, violations are not explicitly represented in the syntax of the language.

Our analysis is related to but distinct from these proposals. We do not suppose there is an atomic or indexed violation or fulfillment proposition. Rather, a violation or fulfillment marker is constructed as a deontic specification. Given the conjunctive construction, we can have systematic implications and relationships among the markers, which is not clearly available in previous approaches. Our approach generalises the previous approaches in that it marks all the previous parameters for agent, action, violation or fulfillment, and the deontic operator. It adds additional parameters to represent the construction of the action, whether an atomic action or complex action, which we have claimed captures the notion of a violation or a fulfillment of a sequence of actions as a protocol. Moreover, it provides a clear path for the addition of parameters should they be needed; the structure of the markers is intrinsically open, flexible, and yet systematic. Finally, by adopting a reductionist approach, we refine the violation and fulfillment markers rather than the logic of the operators as in [30] and [148].

4.3.6 Outstanding Issues

There are a range of outstanding issues that our proposal would have to address – interdefinability, consistency, and implication. We outline them, with reference to related proposals, and indicate how we would approach them. However, as we have said, we are not providing a fully developed language or logic.
4.3.6.1 Interdefinability, Consistency, and Implication

Having such articulated violation and fulfillment markers means that we have lost interdefinability of the deontic operators as in Definition (9), which hold as a consequence of the logic, following the alethic operators. Our view is that the interdefinability of the operators, while formally attractive, is misleading, based as it is on a presumed analogy with alethic modal operators and applied to abstract propositions. Furthermore, there are reasons not to accept some of the equivalences; this is the view taken in [106], where there is no equivalence between OB and PE. In particular, we consider the definition of obligation in terms of prohibition. For us, obligation includes some indication of having fulfilled it, which can consequently be rewarded; however, obligation is defined in Definition (9) in terms of prohibition, yet it is intuitively hard to accept that one can fulfill a prohibition and consequently reap a reward.

The second substantive problem is that we have lost consistency among sets of deontically specified expressions that arise because of the interdefinability. While clearly \( \text{OB}_{\text{atomic}}(\alpha,\iota) \) and \( \neg \text{OB}_{\text{atomic}}(\alpha,\iota) \) are, together in one state, inconsistent as a matter of logic, there is nothing inconsistent about \( \text{OB}_{\text{atomic}}(\alpha,\iota) \) and a prohibition expression \( \text{PR}_{\text{atomic}}(\alpha,\iota) \) both holding. We believe that this is an advantage and not a problem, as it is related to conflicts of deontic specifications; an agent can in fact have antagonistic requirements, and the agent must simply choose what to do. One might say that ideally there would be no such conflicts, but then, were the states ideal, deontic specifications would be unnecessary. As [106] point out, the deontic operators are useful not to filter out undesirable behaviour, but to enable us to reason with it where it occurs.

Third, the system loses implicational relationships such as \( \text{OB}_{\text{atomic}}(\alpha,\iota) \) implies \( \text{PE}_{\text{atomic}}(\alpha,\iota) \).

In any case, our point here is that we cannot assume what operators are interdefinable, mutually consistent, or imply another other as a matter of general properties such as is provided in ScDL. Rather, these hold relative to the specific definitions of the operators. Where we want such relationships, we must provide either alternative definitions of the operators or some alternative way to relate the operators. As pointed out in [193], one way to structure the domain is to use Meaning Postulates [136], which amount to constraints on the relationships between lexical items and how they apply to complex expressions.
4.3.6.2 A Proposal

We review a proposal where similar issues are discussed, comparing and contrasting it to our formalisation and considerations.

[29] points out that the following hold consistently in the logic of [130], but claims that they ought to be inconsistent. In the following expressions, we use the language of [130].

Example 117

a. $PE(\alpha;\beta) \land \neg PE(\alpha)$, a sequence of $\alpha;\beta$ can be permitted even if the first action is not permitted.

b. $\neg PR(\alpha;\beta) \land PR(\alpha)$, an action $\alpha$ can be prohibited, even if the sequence of which it is the first part is not prohibited.

c. $OB(\alpha;\beta) \land \neg OB(\alpha)$, a sequence of $\alpha;\beta$ can be obligatory even if the first action is not obligatory.

[29] claims that these are intuitively inconsistent: a permission on a sequence implies that every action of the sequence is permitted; if a sequence is not prohibited, then no action of the sequence can be prohibited; and if an action is obligatory, then every action of the sequence is obligatory.

While [29] and [174] discuss these in terms of process norms, in contrast to result norms where the violation or fulfillment marker appears in the state which results from the execution of the action, we do not make such a distinction since it rests on just how one defines the interaction between the deontic operators and the complex actions. As we do not propose a fixed set of such definitions, the distinction is moot. Moreover, the discussion in [29] of the alternative interpretations is stipulative rather than argued for on the basis of evidence or logic.

In our language, expressions in Example (117) could also hold, so we have to constrain what the sequences imply such as:

Property 2

$$PE(\alpha;\beta) \rightarrow (PE_{\text{atomic}}(\alpha) \land [\alpha](PE_{\text{atomic}}(\beta)))$$
Other constraints would have to be given for what one wants to hold between deontic specification on a complex expression and deontic specifications on parts ([29]). However, these constraints may be guided by the particular interpretation of the operators and action combinators one has in mind.

On the other hand, [29] denies the basic equivalences of Definitions (9) such as:

**Example 118**

a. \( PE(\alpha) \equiv \neg PR(\alpha) \)

b. \( PR(\pi) \equiv OB(\alpha) \)

Given our definitions, these do not hold simply in virtue of logical form. Moreover, as we discuss below, we agree with [29] that we should allow that some actions are deontically underspecified; there ought to be some actions that are executable which are neither permitted, prohibited, nor obligated.

[29] also claims that certain deontic specifications ought to be inconsistent and provides a constraint to rule them out:

**Example 119**

a. \( \neg (PE(\alpha) \land PR(\alpha)) \): no action can be permitted and forbidden at the same time.

b. \( \neg (OB(\alpha) \land OB(\beta)) \): no action can be obligated, while an action that is different is also obligated.

While we might accept Example (119a), we would reject Example (119b). Certainly in the domain of contracts, the agents may bear a range of obligations at the same time. As we discuss below, the problem arises largely because in [130], the execution of any action other than the obligated action gives rise to a violation, so that without the constraint, the agent bearing two different obligations always violates one of them. While a theory which required deadlines could help to allow simultaneous obligations by differentiating them, the alternative is to allow some actions to be deontically underspecified.

In [29], where the deontic operators on actions are reduced to actions and violation markers which indicate the operator, Example (119a) is constrained as in Example (120), where it is assumed that after the execution of a permitted action \( \neg V_{PE} \) holds and after execution of
a prohibited action $V_{PR}$. As we have not accepted that permissions give rise to violation or fulfillment markers and were we to accept the intuition that an action cannot be both permitted and prohibited, then we express a constraint more along the lines of Example (119a).

**Example 120**

$$\neg (\neg V_{PE} \land V_{PR})$$

Concerning Example (120), notice that in systems with richer violation and fulfillment markers, it is not logically necessary that such expressions are inconsistent. The deontic concepts are primarily deployed to help us reason about the consequences of actions: doing $\alpha$ will violate a prohibition, which in turn implies some sanction; therefore, the agent should not do $\alpha$ if she wants to avoid that sanction. Thus, an agent confronted with $(PE(\alpha) \land PR(\alpha))$ need not suppose a logical inconsistency arises. The deontic specifications do not help the agent reason about what course of action to take, unless the consequences which follow from the distinct violation markers are clearly provided.

Thus, we see that issues bearing on interdefinability, consistency, and implication appear in other research and are addressed by similar means. However, in contrast to much previous work, our goal is not to specify one logical theory to address all these issues, but to allow a flexible, open framework in which a variety of proposals could be proposed, compared, contrasted, and animated.

### 4.4 Two Variants of ScDL

In this section, we review and analyse some of the technical details of ScDL as found in [130] and [106]. We relate them to our proposal. In addition to providing a deeper understanding of ScDL, we also use this section to take into consideration elements which may or may not be relevant to incorporate into our implementation, which is presented in chapter 5. Furthermore, by contrasting the two proposals, we see how they differ on key issues we have discussed above – obligations on sequences and action negation.

Dynamic Deontic Logics developed from Dynamic Logic ([81], [90], and references therein). Natural language semantics also developed analyses in the Dynamic Logic vein, but for different purposes ([101], [35], [75], and [177]). Here we focus on issues related to deontic reasoning. We first present [130], then [106], highlighting similarities and differences. In
the course of the presentation, we indicate just those points which are relevant for the implementation. While there has been subsequent research in ScDL such as [28], the foundations have been little altered; in addition, our objective is a clear and to the point presentation of the issues; the technical and abstract issues raised in [28] are not clearly relevant for our purposes. As we have said, our objective is to make observations and gather issues which are useful in the design of our implementation.

4.4.1 Propositional Deontic Logic — [130]

Meyer [130] expresses the logic of obligation, permission, and prohibition on actions in dynamic logic. While the analysis has continued to be a live research area ([184] and [156], among others), we focus on the fundamental issues. [130] focuses on deontic specifications on actions, but proposals have been made to extend the theory to apply to propositions so as to accommodate state-wise expressions as well ([46] and [191]). As we have seen, the essential idea of the dynamic deontic analysis is that as a consequence of the deontic operator applying to the action, additional postcondition properties arise as the action is executed.

One of the key aspects of a dynamic logical system is that actions and assertions are strictly separated; according to [130, p.109], this avoids paradoxes and counterintuitive propositions which appear in non-dynamic deontic logics. The following point is rather important for our overall view, and it also helps us to clarify what the obligated-to-be expressions are about, in particular, why they implicitly include actions ([130, p.109]):

The philosophical idea behind separating actions and assertions is the simple observation that only assertions can be asserted and only actions can be acted or performed. So it is meaningless to state that the obligation O\(\phi\) of some proposition \(\phi\), such as in OO\(\alpha\), where \(\phi\) is taken to be the assertion stating that the action \(\alpha\) is obligatory. Furthermore, of crucial importance is the consideration that an action may change the current situation (world) and an assertion does not. Furthermore, the fact that actions change situations implies some notion of passing of time.... In our approach, accessibility is defined between a world before a certain action \(\alpha\) is performed and possible ones after \(\alpha\) has been done, i.e., between worlds with different “time-stamps”.

For us, there are two important points to emphasize about this passage. First, [130] is explicitly denying that obligations can be applied to assertions; it would appear that there

---

3We take world to be synonymous with state here.
are only ought-to-do expressions in this view and no ought-to-be expressions under the assumption that the same notion of obligation applies to both.\textsuperscript{4} The second point is that an action logic, as a form of modal logic, makes use of an accessibility relation. In a deontic action logic, the accessibility relation is defined with respect to a temporal relation and those states in which the result of the action holds.

\subsection{Basic Definitions}

We have action names such as $\alpha$ and $\beta$, which are syntactic entities that we use to define atomic actions. The action names denote abstract semantic actions $\alpha'$ and $\beta'$. For example the syntactic expression \textit{to walk} denotes whatever it is we call walking. Complex actions are constructed from atomic (or recursively defined) actions by using action combinators. Given an action name $\alpha$, we may form the assertion $[\alpha]\phi$, if $\phi$ is an assertion. We suppose that the action $\alpha'$ of $[\alpha]\phi$ applies in a state in which the weakest preconditions defined by the action hold and results in a state in which the postconditions, here $\phi$, hold.

[130] claims that the dynamic expression $[\alpha]\phi$ is like a conditional expression meaning that wherever action $\alpha$ is executed (i.e. the assertion $[\alpha]\phi$ holds in a state which meets the preconditions), then $\phi$ holds in the temporally subsequent state. However, it is unlike the conditional in that $\alpha$ is not a proposition. Most importantly, the deontic operators may apply to actions and not to propositions, and because of this, many of the paradoxes and counterintuitive examples of SwDL do not appear.\textsuperscript{5} Note that as we discussed in chapter 3, it is not our objective to address every paradox as we believe they raise a range of issues that are outside the scope of this thesis.

As we have atomic actions, we cannot say anything about exactly why a particular action, say $\alpha$, gives rise to some proposition $\phi$ as expressed in $[\alpha]\phi$; in other words, what is it about doing $\alpha$ that induces the property $\phi$ in the subsequent state. By the same token, the preconditions of actions are not explicitly provided; for example, we would have explicit preconditions if it was stated that the weakest preconditions for action $\alpha$ are that $\phi$ and $\psi$ hold. For ScDL, it is sufficient to provide the actions at this level of abstraction, where the preconditions and postconditions of the actions can be given but, in practice,

\textsuperscript{4}This point changes in [46], where there are both ought-to-do and ought-to-be expressions. However, each is expressed in a different logic, a dynamic logic for ought-to-do and a static logic for ought-to-be. Then the expressions are related. [191] argue against this, claiming that there are implicit actions where we place an obligation on a proposition.

\textsuperscript{5}The point is that many of the paradoxes follow from deontic operators which apply to propositions of propositional logic. The implications and equivalences give rise to the paradoxes. In a Dynamic Deontic Logic, similar paradoxes do not arise because the deontic operators apply to action expressions, and not to propositions. Thus, whatever the implications and equivalences follow in a Dynamic Deontic Logic must be given axiomatically, for they do not follow as in a modal propositional logic.
are left *implicit*. However, as the actions are atomic, there is no intrinsic relationship between them. As we have discussed in chapter 3, we want *lexical semantic* relationships such as *antonym* which provide us with *particular* actions. While in principle, one could have an *antonym* function from atomic actions to atomic actions, this becomes untenable for complex actions. Instead, as we propose in the implementation, we *calculate* the lexical semantic relationships for arbitrary actions. For this to work, we provide *explicit* preconditions and postconditions for the actions. However, further discussion on this is tangential to the presentation of ScDL.

Finally, in certain respects, a dynamic theory with expressions like \([\alpha]\phi\) bears a resemblance to the STIT operator *Sees-to-it-that* \(\phi\), \(E_i\phi\), as discussed in section 2.9.6. Whereas the STIT “action” abstracts over all actions to focus just on the result of the execution of some arbitrary action irrespective of passage of time or precondition properties, dynamic logic introduces additional atomic actions which focus on the result *in addition to* time and precondition properties. Moreover, STIT represents *agency*, which is not usually represented in ScDL approaches. And finally, ScDL actions and STIT differ in terms of the moment of evaluation regarding the performance of the action: for \([\alpha]\phi\), where the preconditions of the action are satisfied, the postconditions of the actions are defined in the resulting state, while for \(E_i\phi\), no consideration of preconditions apply. In general, the relationship between these notions of action appears to be one of relative *abstraction*.

### 4.4.1.2 Basic and Complex Types

We have a finite set \(A'\) of elementary semantic actions \(a', b', \) and so on. We have a finite alphabet \(A\) of elementary syntactic actions \(a, b,\) and so on. We have \(\alpha, \beta\) as variables over elements of \(A\). The variables \(\phi\) and \(\psi\) range over assertions, which we take to be literals. We define \(Act\) and \(Ass\) as sets of action expressions and assertion expressions, respectively; they are the smallest sets satisfying the clauses in Definition (18). From the elementary actions and assertions, we construct more complex actions and assertions, where the complex actions are composed by sequential, choice, joint, conditional, and negation operators, and complex assertions are composed by disjunction, conjunction, conditional, and negation.\(^6\)

\(^6\)There is also the conditional, read as If \(\phi\) then \(\alpha_1\) else \(\alpha_2\), which we leave aside: \(\phi \rightarrow \alpha_1/\alpha_2 \in Act\).
Definition 18

a. $a \in \text{Act}$, for every $a \in A$. Defines elementary actions as actions.

b. Constants $\emptyset \in \text{Act}$, $\_ \in \text{Act}$, and $\text{skip} \in \text{Act}$, pronounced as failure, whatever, and skip respectively.

c. The sequential composition of two actions is an action:
   \[ \alpha;\beta \in \text{Act}, \text{if } \alpha \in \text{Act} \text{ and } \beta \in \text{Act} \]
   This is read as “$\alpha$ followed by $\beta$."

d. The choice between two actions is an action:
   \[ \alpha \cup \beta \in \text{Act}, \text{if } \alpha \in \text{Act} \text{ and } \beta \in \text{Act} \]
   This is read as “$\alpha$ or $\beta$."

e. Two actions simultaneously performed is an action:
   \[ \alpha \& \beta \in \text{Act}, \text{if } \alpha \in \text{Act} \text{ and } \beta \in \text{Act} \]
   This is read as “$\alpha$ together with $\beta$."

f. A negated action is an action:
   \[ \neg \alpha \in \text{Act}, \text{if } \alpha \in \text{Act} \]
   This is read as “not-$\alpha$."

We assume $\text{Ass}$, the set of assertion expressions, has a fixed set of propositional letters, and a special propositional letter $V$, which is also an assertion. We also have modal action operators $[\alpha]$ and $<\alpha>$, defined from the action names; these operators together with an assertion are also assertions.

Definition 19

a. $\phi \lor \psi$, $\phi \land \psi$, $\phi \rightarrow \psi$, $\phi \leftrightarrow \psi$, $\neg \phi \in \text{Ass}.$

b. $[\alpha] \phi$, $<\alpha >\phi \in \text{Ass}$, if $\phi \in \text{Ass}.$

This defines the syntax of the basic action logic.

The special propositional letter $V$, which is the designated marker of violation appears first in Anderson and Moore [7] in a state-wise deontic logic. As we have discussed above the single marker of violation, it does not differentiate among who executed which action with respect to which deontic specification. Given this, it is hard to make distinctions among what follows should a violation hold.

\footnote{These actions are not relevant to our central investigation, so we only mention them where necessary.}
4.4 Two Variants of ScDL

4.4.1.3 Semantics

The semantics of propositional dynamic logic, which is the basis of ScDL, are presented in [90]. ScDL builds on this in ([130] and [156], among others). For simplicity and compactness of exposition, we follow [156]. The semantics can be given in two different ways: a semantics based on possible worlds semantics, and semantics in terms of what are called synchronicity sets, which are sets of actions which occur synchronously ([156, p. 17-41]). So far as we understand, these two approaches to the semantics are not explicitly linked. Nor, for that matter, is it clear why we need the semantics in terms of synchronicity sets, except that since actions are basic, we need some structured domain of meanings from which to systematically construct more complex expressions following the syntax ([156, p. 19]). Moreover, only the possible worlds semantics appears to capture the requisite notion of state change. We only provide some of the key points of [130].

4.4.1.3.1 Intensional Semantics

A model M for Propositional Deontic Logic is [156, p.33]:

Definition 20

\[ M = (A, W, R_\alpha, \mathcal{V}), \]

- where \( A \) is the set of actions,
- \( W \) a set of possible worlds,
- \( R_\alpha \) an accessibility relation associated with action \( \alpha \in \text{Act} \) between a world \( w \in W \) and worlds \( w' \in W \) which are accessible from \( w \) and which result from the execution of \( \alpha \),
- \( \mathcal{V} \) is a valuation function from propositions and worlds to truth values.

With this, we can provide the truth of a dynamic action \( \alpha \) relative to an outcome \( \phi - [\alpha]\phi \) – in a world.

Definition 21

\[ w \models [\alpha]\phi \text{ iff } \forall w' \in W \text{ such that } R_\alpha(w, w'), w' \models \phi, \]

Executing \( \alpha \) in \( w \) is such that \( \phi \) is necessarily true in all worlds \( w' \) which are temporally subsequent to \( w \) and are the outcome of performing \( \alpha \).
While \([\alpha] \phi \) means that the action \( \alpha \) always results in a state where \( \phi \) holds, \(<\alpha> \phi \) means that the action \( \alpha \) results in at least some state where \( \phi \) holds.

A sequence of actions \([\alpha; \beta] \phi \) relativizes the outcome of the second action to the outcome of the first action.

**Definition 22**

\[
 w \models [\alpha; \beta] \phi \quad \text{iff} \quad \forall w' \in W R_\alpha (w, w') \quad \text{and} \quad \forall w'' \in W R_\alpha (w', w''), \quad w'' \models \phi.
\]

Neither [130] nor [156] provide extensive intensional truth conditional denotations for the full range of action expressions, but instead they focus on *Synchronicity Sets*.

**4.4.1.3.2 Synchronicity Sets** We assume a set of atomic actions \( A = \{a_1, a_2, \ldots\} \). These syntactic actions stipulate the performance of a semantic action. We assume a set of semantic actions \( A' = \{a'_1, a'_2, \ldots\} \), where each syntactic action correlates to a semantic action (and vice versa). Semantic actions can be executed simultaneously. We take the powerset of \( A' \) to yield the *synchronicity sets* – those sets of actions which are executed simultaneously; each of the subsets is given between square brackets. The set of synchronicity sets is \( S \).

**Example 121**

\[
\text{Suppose } A' = \{a'_1, a'_2, a'_3\}, \\
S = \varnothing A' = \\
\{\emptyset, [a'_1], [a'_2], [a'_3], [a'_1, a'_2], [a'_1, a'_3], [a'_2, a'_3], [a'_1, a'_2, a'_3]\}
\]

Each of the synchronicity sets \( s \) indicates a step which is the execution of the actions in the set. For instance, if in a state the synchronicity set \([a'_1, a'_2] \) is executed, then both \(a'_1\) and \(a'_2\) are executed. Given this, sequences of synchronicity sets denote the courses of performances of actions. For instance, \([a'_1, a'_2]\) followed by \([a'_3]\) indicates that first the actions \(a'_1\) and \(a'_2\) are simultaneously executed, then followed by an execution of \(a'_3\).

The *meaning* of an atomic action \( a \) is that set of synchronicity sets \( s \) which have the semantic action as an element [156, p.25].
Chapter 4 4.4 Two Variants of ScDL

Definition 23

\[ [\alpha] = \{s \in S \mid \alpha \in s \} \]

For example,

\[ [a_1] = \{[a'1], [a'1, a'2], [a'1, a'3], [a'1, a'2, a'3]\} \]

Exactly which of these synchronicity sets is executed depends on the state.

We can give a set theoretic definition of the action combinators. For instance, simultaneity is expressed as follows.

Definition 24

\[ [\alpha \& \beta] = [\alpha] \cap [\beta], \text{ which is just set theoretic intersection of the synchronicity sets of the two actions.} \]

For example,

\[ [a_1 \& a_2] = \{[a'1, a'2], [a'1, a'2, a'3]\} \]

Negation of an action is given as that set of synchronicity sets which do not have the action. The action \textit{skip} is a designed action which does nothing; that is, it has no preconditions or postconditions, so changes nothing. It is included here in order to be consistent with the presentation in [156] and is otherwise not relevant to our presentation.

Definition 25

\[ [\bar{\alpha}] = [\alpha]^\neg, \text{ where this is set theoretic complementation of } [\alpha] \]

For example,

\[ [\bar{\alpha}] = \{[a'2], [a'3], [a'2, a'3], \text{[skip]}\} \]

In effect, this version of action negation says that not doing an action is to do any of the alternative actions. We have further discussion about action negation below with respect to [29].

Sequences of actions are given as function composition: each of the synchronicity sets of the first action are followed by the synchronicity sets of the second action [156, p.25].
Definition 26

\[ [\alpha; \beta] = [\alpha] \circ [\beta], \text{ where } \circ \text{ denotes function composition.} \]

For example,

\[ [a1; a2] = \{[a'1]/[a'2], [a'1]/[a'2, a'3], \ldots \} \]

This is conceptually very close to the intensional semantics for sequences of actions in Definition (22).

Here, we have provided the semantics of atomic and complex actions in terms of those actions which are synchronously executable. For our purposes, we find a semantics of actions defined this way to be obscure. Rather than decomposing the meaning of an action into some more primitive notion, it is expressed relative to other actions which it is synchronous with. More to the point for our purposes, it is difficult to see how lexical semantic restrictions can be placed on them. In the next subsection, we discuss further complex expressions in ScDL only syntactically.

4.4.1.4 Propositional Dynamic Logic

We present selected axioms of the system for the sake of providing additional considerations about ScDL. Notice that we have two sorts of logical connectives – the propositional logic connectives and the dynamic connectives which apply to the action expressions. However, in all cases but one (negation), the action connective is defined in terms of simpler action expressions and/or the connectives of propositional logic.

We have the axioms which define the complex actions of the basic system. Following this, we return to the application of deontic operators to complex actions. We comment on the axioms along the way. We assume that ScDL includes all the tautologies of propositional calculus.

We have Axiom K for actions.

Definition 27

\[ \vdash [\alpha](\phi_1 \rightarrow \phi_2) \rightarrow ([\alpha](\phi_1) \rightarrow [\alpha](\phi_2)) \]

In chapter 3, we explored issues relating to conditional obligations, where we claimed that Axiom K with deontic operators is not a well-formed expression in natural language.
Here a similar issue arises as to the natural language intuition behind the antecedent of the conditional as well as the well-formedness of the consequent. As with our previous discussion, we leave these issues aside for future research, noting that it implies we are not adopting ScDL as is.

We have sequences.

**Definition 28**

\[ \vdash [\alpha_1, \alpha_2](\phi) \equiv [\alpha_1]( [\alpha_2](\phi)) \]

If a sequence of actions bring about \( \phi \), then, when we perform the actions in sequence, the result is \( \phi \). Notice here that the complex action of sequence is decomposed into the simpler actions.

We have the choice combinator.

**Definition 29**

\[ \vdash [\alpha_1 \cup \alpha_2](\phi) \equiv [\alpha_1](\phi) \land [\alpha_2](\phi) \]

If there is a choice on which action to use to bring about \( \phi \), then doing either action alone brings about \( \phi \).

We have simultaneous actions, but the implication only goes one way.

**Definition 30**

\[ \vdash [\alpha_1](\phi) \land [\alpha_2](\phi) \supset [\alpha_1 \& \alpha_2](\phi) \]

If doing one action brings about a property and doing another action brings about the same property, then doing the actions simultaneously brings about the property, assuming that the actions have the same duration. Note that the implication does not go in the other direction. This would say that if both actions together bring about the property, then any single one does independently. Another way to put it is that we cannot decompose the complex action into the simpler components. In some sense, the very *combination* of the actions brings about the result.
We have a family of axioms about action negation, \( \alpha \), read as not-\( \alpha \). This is not the same as \( \neg [\alpha] \phi \), which says that it is false that executing \( \alpha \) always results in \( \phi \). In addition, \( \alpha \) is the one operator on actions which is not defined in terms of atomic actions and propositional logical operators. [130] provides action negation axiomatically, claiming the axioms are sufficient to define the behavior of the operator. We only give one axiom to give a flavor of the analysis, others are found in [130].

**Definition 31**

\[
\vdash [\alpha_1] (\phi) \lor [\alpha_2](\phi) \rightarrow [\alpha_1 \cup \alpha_2](\phi)
\]

The alternative to a choice among actions is that there is an alternative to at least one, where the actions have to be of the same duration.

Finally, we have two rules, *Modus Ponens* and *Action Necessitation*.

**Definition 32**

\[
\vdash \phi, \vdash \phi \supset \psi \\
\vdash \psi
\]

**Definition 33**

\[
\vdash \phi \\
\vdash [\alpha](\phi)
\]

In the next section we add deontic specifiers.

### 4.4.1.5 Propositional Dynamic Deontic Logic

In section 4.2, we already introduced the deontic operators on atomic and some complex actions. In section 4.2.2, we discussed issues bearing on sequences, CTDs, and how we address them. In this section, we focus on several remarks on theorems concerning permissions on actions, prohibitions on sequences, and obligations on sequences. Our point is that we claim that the logic does not adequately represent our intuitions. The proofs are based on the axioms and theorems of [130].
4.4.1.6 Prohibitions on Sequences

If $\alpha_1;\alpha_2$ is prohibited, it is prohibited from doing $\alpha_2$ in a context in which $\alpha_1$ has already been performed.

Definition 34

a. $PR(\alpha_1;\alpha_2) \equiv [\alpha_1]/(PR(\alpha_2)) \equiv [\alpha_1]/([\alpha_2]/(V))$

b. If one is prohibited from moving the left toggle up and then moving the right toggle left, then having moved the left toggle up, one is then prohibited from moving the right toggle left.

c. PROOF: $PR(\alpha_1;\alpha_2) \equiv [\alpha_1;\alpha_2]/(V) \equiv [\alpha_1]/([\alpha_2]/(V)) \equiv [\alpha_1]/(PR(\alpha_2))$

Under this interpretation of prohibition on a sequence, $F(\alpha_1;\alpha_2)$ is consistent with a circumstance where $\alpha_1$ and $\alpha_2$ are independently permitted, yet, it is prohibited from executing first $\alpha_1$, and then $\alpha_2$. A natural example of this is that It is prohibited to drink and then drive, though drinking and driving are independently permitted; the violation becomes significant where one drives after drinking. As we have already discussed for obligations on sequences, we would like finer-grained violation and fulfillment markers than Definition (34) provides and for similar reasons: we want to distinction violations of prohibitions on different sequences rather than homogenising them as this Definition provides.

A second point is that we believe there is another possible interpretation of a prohibition on a sequence, where we understand the sequence to be a protocol. Notice that in Definition (34), a violation can only arise with respect to the performance of the second action when it occurs after the first action; it does not associate any violation marker with the first action. But, it seems possible to consider that where there is a prohibition on a protocol such as moving the switches in a fixed order, then some indication of wrong-doing ought to be flagged as soon as one starts the sequence. Consider that in Definition (34b) it is necessary to have started down the “path” of a course of actions to attain the violation flag; if one does any action other than $\alpha$ and follows that with any action, then no violation flag arises. However, to attain the full “force” of the violation, one must first execute $\alpha$ and then follow this with $\beta$. Our point is that, for some purposes, some indication of having started along a prohibited course of actions might be useful. This could be indicated with a warning label rather than a full-fledged violation flag. It would be useful in the design of a security system which indicates successive approximations towards full-fledged violation
4.4 Two Variants of ScDL

and take appropriate response; as it is, Definition (34) would allow a signal of violation to arise after the worst action has been executed. This may not apply in all circumstances such as drinking and driving, but for others it may be appropriate. We can compare these interpretations of prohibitions on sequences with obligations on sequences.

4.4.1.7 Obligations on Sequences

We have already extensively discussed obligations on sequences. The purpose of this section is to provide the technical background and observations on that discussion, primarily by proving the equivalence between OOS and SOO. The proof is as follows, where the steps in the proof depend on axioms in [130] and not repeated here.

**Theorem 2**

a. \( OB(\alpha_1,\alpha_2) \equiv OB(\alpha_1) \land [\alpha_1](OB(\alpha_2)) \)

b. If one is obligated to move the left toggle up and then to move the right toggle left, then one is obligated to move the left toggle up, and then having moved the left toggle up, one is obligated to move the right toggle left.

c. PROOF: \( OB(\alpha_1,\alpha_2) \equiv PR(\overline{\alpha_1},\overline{\alpha_2}) \equiv \overline{[\alpha_1],[\alpha_2]}(V) \equiv \overline{[\alpha_1]}(V) \land [\alpha_1]([\alpha_2](V)) \equiv [PR(\overline{\alpha_1}) \land [\alpha_1](PR(\overline{\alpha_2}))) \equiv OB(\alpha_1) \land [\alpha_1](OB(\alpha_2)) \)

As we have commented, there is no differentiation between violations which may arise given the different obligations, whether on the first action, the second, or on the sequence as a whole. Comparing obligations on sequences to prohibitions on sequences, we see that the derivations lead to different distributions: prohibitions only specify what happens in the end action of the sequence, while obligations specify what happens in both portions. However, we claimed that there is an interpretation of prohibitions on sequences where some marker arises as a consequence of executing the first action. Note, in any case, the deontic operator applied to a sequence is always equivalent to the deontic operator applying to some atomic actions and a primitive violation marker.

In this section, we have reviewed key elements of [130] which are relevant for our purposes, as well as alternative interpretations of deontic specifications on complex actions. This gives us a clearer idea of the elements that are to go into an implementation. In the next section, we discuss [106], which is another version of ScDL and developed around the same time as [130], though it has not subsequently been as extensively developed. We do not
provide a full technical discussion of their approach because it is very similar to [130] in
the main, so would be redundant. Moreover, it addresses issues that are not central to
our line of investigation. Rather, our objective is to point out and describe those relevant
elements of [106] which were distinct from [130] and which were useful in the development
of our analysis and implementation.

4.4.2 Deontic Action Logic – [106]

[106] provide a closely related analysis of deontic operators on dynamic actions. The
intuitive starting point of [106] is that deontic logic is not only to specify the behavioral
constraints which entities in the given system must adhere to, but moreover to be able to
“...reflect reality a little more faithfully and to provide a means to indicate undesirable
behavior in order to avoid it and to report on it should it occur in an integral and uniform
manner.” Or, in other words, to be able to report on and reason with bad behavior rather
than trying to banish it altogether. Were we not to allow some representation of bad
behavior, then where it (inevitably) occurs, we will be unable to reason with it.

[106] also claim that “...actions should be modeled as always leading somewhere (total
functions in our models) no matter in which state they are performed.” Thus, actions
are deterministic. Because of this, we cannot model a While which does not terminate,
though it is not clearly relevant to do so in our domain of application of contract execution.
This is to contrast to theories in which an illicitly executed action may lead to a special,
undefined state; from this state, nothing can follow and no further actions can be executed
(i.e. programs where systems hang or crash are examples). One can say it is a situational
black hole. In contrast, in a theory in which actions are total functions, though leading to
states where undesirable behavior is recorded, we leave open the possibility of subsequent
actions from such states; we can introduce a recovery mechanism, which could not be
defined were actions to lead in some cases to the undefined state.

The view that we should allow the possibility of executing bad behavior and be able to
reason with it is in contrast with theories which want to rule out bad behavior. For
example, [4] define norm-compliant protocols which, in effect, use normative statements
to filter out those protocols which violate the norms. Such protocols are supposed to be
more efficient to execute because [4, p.47] “...norms tend to be hard to understand and...a
bit challenging.... It is not unlikely that in highly regulated systems agents (and humans
alike) might become overly cautious, trying not to violate any of the norms and thereby
seriously reducing their efficiency and even influence the outcome and success of their
goals....” Whatever the advantages or disadvantages of this approach, in our view and
for our purposes, it amounts to a repudiation of the fundamental intention of deontic specification, which is, as pointed out above, to allow violations in order to reason with them and to provide realistic models of social behavior. Our implementation is designed to specifically take this view into account.

As we have just said, we only point out and describe those elements of their theory which are said to be different from [130], and where we can, we bridge between the approaches. These are:

- A distinction between static information, action descriptions, and action prescriptions. [130] makes use of the latter two, but not static information.
- A designated proposition \( \eta \) which partitions states into normative and non-normative, where the normative states are those with \( \eta \) and the non-normative states are those with \( \neg \eta \). This relates to the violation V proposition in [130].
- A notion of permitted to refrain but no notion of action negation, though these are related notions.
- A distinction between obligations on sequences and sequences of obligations. This contrasts with the equivalence between them in [130].
- The introduction, maintenance, and elimination of deontic specifications.

As with [130], it is important to emphasise that we do not directly adopt [106] in the implementation, but rather use the differences between the two approaches as inspiration for the implementation. Some of these points we have already raised in section 4.3, others are made clearer in the implementation. In particular, we have discussed different ways to mark alternative contexts with violation or fulfillment and also distinguishing between obligations on sequences and sequences of obligations.

The other three elements are important in the implementation for the following reasons.

- We provide distinct levels at which processes in the implementation apply, distinguishing between what is applied within a given state and what is applied to change from state to state.
- From the notion of permitted to refrain, we are led to the more specific notion of action antonym provided by lexical semantic functions.
- We provide a specific mechanism for the introduction, maintenance, and elimination of deontic specifications.
4.4.2.1 Levels of Analysis

[106] distinguish between static information, action descriptions, and action prescriptions ([186] and [187] for similar distinctions). Action descriptions and prescriptions are conceptually the same as the actions and deontic specifications on actions as also found in [130], respectively.

The static information models the entities, properties, relationships, and constraints in the system; the static information is expressed in a sorted first-order predicate logic (SFOPL), which is not part of the presentation of [130]. The static information is used to describe system specifications, which are the invariant properties which hold of every state of the system.

We may have a range of static constraints, namely formulae which hold of all or some elements of the domain at all times in any context (more on contexts below); they are constraints in that they narrow the contexts in which expressions of the language may be true. For example, consider a telephone exchange system. We have a variable t over telephone individuals; it is (implicitly) universally quantified over. We have Receiver-up, Busy, and Connected, which are predicates of or relations between telephones. Such predicates and relations do not express change from context to context. We can then express the following sorts of constraints.

Example 122

\[ a. \, \text{Receiver-up}(t) \rightarrow \text{Busy}(t) \]
\[ b. \, \text{Connected}(t, t') \rightarrow (\text{Busy}(t) \land \text{Busy}(t')) \]

In all states, when the receiver is up, the phone is busy; two phones cannot be connected where one or the other is not busy. Another way of characterizing such statements is to say that they are true in all states at all times; they are static because they do not vary.

More relevant for our purposes in the implementation, we may also have static constraints on action names, where the difference between action names and actions is the same as that in [130]. Unlike [130], action names can have argument places which further specify the action. Given this, we can state relationships between action names, which then enable us to state relationships between certain actions. Suppose a is a variable over account identifiers, Debit and Credit are one-place action names, and Not-together is a predicate of type $<\text{Act,Act}>$ which is intended to mean that two action names cannot simultaneously
hold. Then the following is intended to mean that we cannot simultaneously debit and credit the same account:

**Example 123**

\[ \forall a \text{ Not-together(Debit}(a), \text{ Credit}(a)) \]

A range of such system constraints can be provided.

The action descriptions represent dynamic information in much the same way as in [130], though [106] explicitly represent an agent associated with the execution of the action:

An atomic modal formula \([A, \alpha]\phi\), where \(\alpha\) is an action name, \(A\) is an agent name, and \(\phi\) is a first order formula, says that if the action \(\alpha\) is performed by agent \(A\) in state \(s\), then the first order property \(\phi\) holds in the state \(s'\) reached by \(A\) performing \(\alpha\) in \(s\).

Before the execution of the action, we cannot assume that \(\phi\) holds in \(s'\); after the execution of the action, \(\phi\) holds in \(s'\).

As with [130], [106] also have a range of action combinators for *sequence*, *choice*, and *simultaneity*. While the definitions are not exactly the same, the differences are not relevant with respect to our main interest, so we do not undertake a comparison and contrast. In particular, we adopt the notion of *sequence* from [130], which is relevant and sufficient to our purposes.

The action prescriptions are deontic specifications on action descriptions. Rather than a designated violation proposition \(V\), [106] introduces a designated *normative* proposition \(\eta\). The normative states are those in which \(\eta\) holds and the non-normative in which \(\neg\eta\) holds; the states are partitioned between normative and non-normative states. We could this to correlate with \(V\): \(\neg\eta \equiv V\), and \(\eta \equiv \neg V\). Doing so allows for a readier comparison to [130].

We have the same symbols for permission \(\text{PE}\), obligation \(\text{OB}\), and prohibition \(\text{PR}\); however, as deontic specifications on action descriptions, the deontic operators are of type \(<\text{Agt,Act}>\). Thus, action prescriptions always represent the agent of the action.

---

8This is not quite right as \(\phi\) can be any formula of Deontic Action Logic.
A significant difference between [106] and [130] is that [106] do not define action negation. Rather, they provide a conceptually related notion of refraining from an action, which essentially means to execute some other action than the action one refrains from. This is an explicit version of what we earlier interpreted $\overline{\alpha}$ to mean. In terms of this, obligation on an action means that it is not permitted to refrain from doing the action; doing some action other than the obligated one leads to a state which is non-normative.

We focus on obligations and sequences primarily, but include the others for related observations. In particular, we construe [106] to propose three different interpretations of obligation, similar to what we expressed as $\text{OB}_{\text{atomic}}$, $\text{OB}_{\text{dist}}$, and $\text{OB}_{\text{coll}}$ in section 4.3; however, as they have no articulated violation or fulfillment markers, we indicate them with $\text{OB}_{\text{atomicKM}}$, $\text{OB}_{\text{distKM}}$, and $\text{OB}_{\text{collKM}}$, where we replace our complex violation markers with the atomic violation proposition $V$, respectively. We present [106], correlating it with these alternative operators.

### 4.4.2.2 Deontic Action Logic

We present the relevant portion of the Deontic Action Language (DAL) of [106], followed by discussion. We note where we have deviated from DAL for clarity. As with [130], a model is a set of states and each action is interpreted as an accessibility function over these states. It is, then, essentially a state transition system in which states have structure and transitions are labeled by actions.

**Definition 35 (Syntactic Categories)**

- a. We have a sort $\text{Act}$, which is the set of actions.
- b. We have a sort $\text{Agt}$, which is the set of agents.
- c. We have a sort $S(\text{Act})$, each element of which is a finite (possibly singleton) sequence of actions, where a sequence is given as list of ordered actions.
- d. Variables $\sigma^1, \ldots, \sigma^n$ of the sort $S(\text{Act})$.
- e. Predicate symbols $\text{PE}$, $\text{OB}$, $\text{PR}$, and PerRef of type $<\text{Agt}, \text{Act}>$, and $\text{OS}$ of type $<\text{Agt}, S(\text{Act})>$.
- f. Logical constant $\eta$, which represents the normative proposition.

Note the distinction between types here and the function symbols in (d). We use $\text{OB}$ where they use $O$. 

172
d. Function symbols $\langle \rangle$, which is of type $<\text{Act}, S(\text{Act})>$, and the comma character, which is of type $<S(\text{Act}), \text{Act}, S(\text{Act})>$. These function symbols are used to form concatenated and sequenced actions.

g. Logical connectives $[ ]$, which represent the function from expressions of type $<\text{Agt}, S(\text{Act})>$ to action descriptions. It is the operator which corresponds to $[ ]$ that is used to make an action name a modal operator, but is used for sequences.

In addition, we assume that the clauses on quantifiers, variables, and equality as defined in SFOPL apply to the sort $S(\text{Act})$. Notice that at this point in the language we do not have action combinators as in [130]. Rather, we have action sequences which indicate that one action follows another. The difference between them is that while the sequence combinator in [130] is a function from two actions to an action, while a sequence of actions is not such a function. The difference becomes significant in the distinction between $\text{OB}_\text{distKM}$ and $\text{OB}_\text{colKM}$, which [130] does not make.

**Definition 36 (Formation Rules)**

a. Terms: Any constant or variable of type $S(\text{Act})$ is a term.

b. Atoms: The SFOPL clauses about the construction of atoms and equality apply to $\text{PR}, \text{PE}, \text{PreRef}, \text{OS}$.

c. Formulae: All clauses of SFOPL. The constant $\eta$ is a formula. Where $A$ is a term of sort $\text{Agt}$, $\sigma$ is a term of sort $S(\text{Act})$, and $\phi$ is a formula of DAL, then $[A, \sigma] \phi$ is a formula.

**Definition 37 (Axioms)**

a. $[A, <\alpha>] \phi \leftrightarrow [A, \alpha] \phi,$
   where $<\alpha>$ is of type $S(\text{Act})$ and of length one.

b. $[A, <\sigma, \alpha>] \phi \leftrightarrow [A, <\sigma>] [A, \alpha] \phi,$
   where $<\sigma, \alpha>$ is of type $S(\text{Act})$, $<\sigma>$ is of type $S(\text{Act})$ (possibly unary), and $<\alpha>$ is of type $S(\text{Act})$ (unary). In other words, $\sigma$ ranges over sequences of actions and $\alpha$ ranges over atomic actions.

c. $\eta \rightarrow (\text{PE}(A, \alpha) \leftrightarrow [A, \alpha] \eta)$

d. $\neg[A, \alpha] \eta \rightarrow [A, \alpha] \neg \eta$
4.4 Two Variants of ScDL

e. $OS(A, <\alpha>) \leftrightarrow OB(A, \alpha)$

Same comment as (37a). This is the case where OS applies to an agent and a unary sequence of actions. In this case, OS correlates with $OB_{atomicKM}$.

f. $OS(A, <\sigma, \alpha>) \leftrightarrow (OS(A, \sigma) \land [A, \sigma]OB(A, \alpha))$, where $\sigma$ ranges over sequences of actions and $\alpha$ ranges over actions. In this case, OS correlates with $OB_{distKM}$, where the operator distributes over the actions in the sequence of actions.

g. $PerRef(A, \alpha) \leftrightarrow \exists \beta ((\beta \neq \alpha) \land PE(A, \beta))$

h. $OB(A, \alpha) \rightarrow PE(A, \alpha)$

i. $OB(A, \alpha) \rightarrow \neg PerRef(A, \alpha)$,

i.e. $OB(A, \alpha) \rightarrow \forall \beta ((\beta \neq \alpha) \rightarrow \neg PE(A, \beta))$

j. $PR(A, \alpha) \leftrightarrow \neg PE(A, \alpha)$

While [106] do not discuss prohibitions, we assume they are the negation of permissions, for the execution of a prohibited action ought to imply that the subsequent state is non-normative; thus we have included Axiom (37j). We see here that [106] include Axiom (37h), which we earlier argued against (and which [32] abandon).

In the following discussion section, we point out similarities and differences between DAL and [130]. As it is unclear which to choose from, we remain agnostic and instead provide an implementation in which these differences can be expressed.

4.4.2.3 Discussion

In the following, we discuss the relation between the normative proposition and permission, action negation and $PerRef$ as well as the difference between obligations on sequences and sequences of obligations.

We have already introduced the normative proposition $\eta$. In (37c), we see that the normative proposition implies that the permitted actions are those actions which lead to a state in which the normative state holds. As earlier, we correlated the normative proposition with the negation of the violation proposition, Axiom (37c) is close to the interpretation of permission in [130]. There are two issues to observe here. First, as with [130], it is unclear whether there is a useful distinction between permission on and availability of actions, where the latter means that the actions are among those which are executable in a given state. Moreover, Axiom (37d) says if it is false that an action gives rise to the normative proposition in the subsequent state, then that action gives rise to the non-normative

174
proposition in the subsequent state. Thus, it forces models to have only states with $\eta$ or $\neg \eta$. Deontic specification on an action determines which of $\eta$ or $\neg \eta$ follows from the execution of the action. Note that these two Axioms distinguish between permitted actions and actions which are not deontically specified: all permitted actions lead to normative states while those actions which are not deontically specified can lead to either non-normative states or to normative states. Yet, given Axiom (37j), in fact, all such actions are implicitly either a permitted action or a forbidden action; that is, in sum, in a model all actions are deontically specified and give rise to either $\eta$ or $\neg \eta$.

A second point is that Axiom (37c) only implies that if the current state is normative, then it holds that permitted actions are equivalent to actions which lead to normative states. In non-normative states, all bets are off, and one can define deontically specified recovery actions such as one might find in a CTD case which lead from a non-normative state to a normative state. Nonetheless, the distinction between $\eta$ and $\neg \eta$ is too coarse to define what deontic specifications apply with respect to a particular violation.

A significant difference between [106] and [130] is that [106] do not have action negations, for it is conceptually difficult to determine what counts as not executing $\alpha$, particularly in any realistic or natural context. Moreover, where action negation $\overline{\alpha}$ is the set which is the complement set relative to $\alpha$ and a domain of actions, we have interpreted expressions of the form $[\overline{\alpha}](\phi)$ to mean that executing any action in $\overline{\alpha}$ subsequently give rise to $\phi$. This interpretation is not overtly expressed in [130] nor is it explicitly part of the language. The problem is particularly important with respect to obligations, where not doing the obligated action leads to a violation marker.

To make this interpretation of action negation explicit without providing an action negation operator, [106] use an operator which represents permission to refrain – $\text{PerRef}$ – which is defined in terms of permission and negation. $\text{PerRef}$ of an action $a$ is true where there is at least one action $b$ which is distinct from $a$, and $b$ is permitted; in other words, to be permitted to refrain from executing an action means that one is permitted to execute some other action. In terms of this, where action $\alpha$ is obligatory, then it is false that it is permitted to refrain from executing $\alpha$; all actions other than $\alpha$ lead to non-normative states. This represents explicitly the notion of executing alternatives, but without action negation.

For our purposes, the significance of this move is that it explicitly introduces a conception of alternatives of a given action as particular actions rather than as a predefined set of actions (also see [16] for discussion of obligations and alternative actions). However, the formulation in [106] uses universal quantification over these alternatives, and one might
take it to be too strong. After all, as we have previously discussed, were not doing the action to simply denote the set theoretic complement of the set of actions relative to \( \alpha \), then doing any other action than \( \alpha \) would lead to a violation. Yet this is too stringent a requirement, for surely an agent may bear an obligation and execute any number of actions without violating the obligation; only some actions will count towards violation. Indeed, it is relatively straightforward to introduce this notion, while keeping to [106], for rather than Definition (37i), we have:

Property 3

\[
OB(A, \alpha) \rightarrow \exists \beta((\beta \neq \alpha) \land \neg PE(A, \beta))
\]

This says that if one is obligated to execute \( \alpha \), then there is some action \( \beta \) other than \( \alpha \) such that it is not permitted to execute \( \beta \). Notice that it does not say that any action other than \( \alpha \) is not permitted, which is the interpretation which we have previously criticized. This alternative formulation comes closer to the intuition above in that we do not want all actions other than \( \alpha \) to be not permitted; we want some actions to be deontically underspecified. However, this does not capture the particular actions which might not be permitted. For example, if one is obligated to do \( \alpha \) and not permitted to do \( \gamma \), this would satisfy the obligation in Definition (3) but not our intuition that there must be some relationship between \( \alpha \) and \( \gamma \). It does not satisfy our earlier observation of symmetrical obligations: if Bill is obligated to do \( \alpha \) then doing \( \alpha \) fulfills the obligation and \( \beta \), say, violates it; on the other hand, if Bill is obligated to do \( \beta \), then doing \( \beta \) fulfills the obligation, while doing \( \alpha \) ought to violate it. This identifies a very tight relationship between the obligated action and that which violates it. This relationship is not captured in Definition (3). Nonetheless, this alternative notion takes us partway to the lexical semantic functions, which are functions which select some actions which satisfy the consequent. What needs to be refined is just which actions those are. We discuss action negation further in section 4.5.

[106] distinguish between sequences of obligations and obligations on sequences. This in turn is based on the distinction between elements of type S(Act), which are sequences of actions, and the result of applying the sequence combinator \( ; \) on a pair of actions, where this result is function composition. Where the obligation operator in [106] applies to a sequence of two (or more) actions \( OS(A, <\alpha, \beta>) \), it correlates with the \( OB_{distKM} \) interpretation. Both of these interpretations are found as well in Meyer [130]. However, [106] also have an expression of the form \( OS(A, \alpha;\beta) \), where \( \alpha;\beta \) is regarded as a atomic action sequence since the result of function composition is a unary action. This correlates
with our \( \text{OB}_{\text{collKM}} \) interpretation. Importantly, they attend to \textit{structural information} in the syntax of the expression; that is, it is \textit{an} action, though comprised of two \textit{component actions}.

The action named \( \alpha;\beta \) is to be viewed as just that, \textit{an} action, and not a sequence of actions. It is not difficult to find examples of where the indivisibility feature of this combinator is useful. The generic name given to sequences of this nature is \textit{transaction}, a term arising from the area of data and knowledge bases....

A transaction is an action constructed out of component actions such that no intervening actions can be executed and there is no \textit{visible} intervening state. In these respects, it \textit{appears} much like an atomic action. In view of this, an obligation on a sequentially composed action is not equivalent to an obligation on a sequence of actions. An obligation on a sequentially composed action is as if it is an obligation on a primitive action.

Property 4

\[
\text{OB}[A, \alpha;\beta] \neq \text{OS}[A, <\alpha,\beta>]
\]

In fact, given the interpretation of \( \alpha;\beta \) and \textit{obligation}, neither \( \alpha \) nor \( \beta \), nor any other action, can be permitted if \( \text{OB}[A, \alpha;\beta] \) holds. As [106] note, when there is an obligation on a sequentially composed action, “a transitory suspension of all other permitted actions is imposed until the obligation has been discharged.” While we find \( \text{OB}_{\text{atomicKM}}, \text{OB}_{\text{distKM}}, \) and \( \text{OB}_{\text{collKM}} \) interpretations of obligation, we do not find \( \text{OB}_{\text{intKM}} \). The essential reason is that every action is either an action which fulfills or violates the obligation; there is no possibility for deontically underspecifying an action. Note, finally, that [106] do not consider CTDs or the relation between obligations on sequences.

4.4.2.4 Management of Deontic Specifications

Finally, [106] discuss issues related to the introduction, maintenance, and elimination of deontic specifications. One observation is that \textit{obligations} appear to be transitory; indeed, since \textit{obligations} block all other permitted actions (i.e. that no action other than the obligated action can be executed without introducing violation), were \textit{obligations} persistent, then one could only \textit{ever} execute the obligated action in order to avoid violation. This is
plainly undesirable in terms of contracts which model human behaviour. By introduction of an obligation $\text{OB}[A, \alpha]$, they mean that at one state $\neg\text{OB}[A, \alpha]$ holds and subsequently $\text{OB}[A, \alpha]$ holds; maintenance means that $\text{OB}[A, \alpha]$ holds from one state to the next; while elimination means that at one state $\text{OB}[A, \alpha]$ holds, but in the subsequent state $\neg\text{OB}[A, \alpha]$ holds. We can assume that obligations are introduced or eliminated following the execution of some action, while obligations can be maintained either by inertia or the execution of some action which reintroduces the obligation.

We provide an example of obligation change in the sample telephone exchange specification briefly introduced above. We assume that the telephone receiver is up initially, during which the telephone exchange indicated with $\text{Ex}$ has the obligation that the telephone is connected with some other telephone. Furthermore, where the action of putting the receiver down is executed, the obligation that the phones are connected no longer holds. Note here that what is described is an abstract specification, not the means to introduce or eliminate obligations.

**Example 124**

\[
\begin{align*}
\text{Receiver-up}(t) & \land \text{OB}(\text{Ex}, \text{Connect}(t,t')) \\
& \land \neg\text{OB}(\text{Ex}, \text{Connect}(t,t'))
\end{align*}
\]

We believe that such manipulations of deontic specifications are key in the solution of the CTD problem.

### 4.5 Action Negation

In this section, we draw together several of our discussions concerning action negation, adding additional considerations from [29].

In section 2.9.4, we pointed out that a lexical semantic notion of *antonym* is relevant to deontic specification, which is *symmetrical* relationship for our purposes. Furthermore, in principle, we want to allow underspecified actions, meaning that for a given obligation there must be actions which fulfill the obligation and which violate it, but in principle there could be actions which neither fulfill nor violate the obligation. This point is made again and more concretely in section 4.2.3.2. We have also pointed out the symmetrical relationship of obligated actions and their violation and fulfillment conditions. In section 2.9.6, we discussed the notion of *refraining* in a STIT framework. Refraining is clearly
related to the lexical semantic notion in that reference is made in both to whether or not a
given property holds as a result of the execution of the action. In STIT, refraining is seeing
to it that it is not seen to that a given property holds. However, STIT makes no reference
to particular actions nor to state change. We have also pointed out in section 4.2.3.2 that
we do not necessarily want the negation of an action to be defined as the complement set
of actions. In section 4.4.2, we discussed the notion of permission to refrain. The key
observation there was the introduction of quantification over the domain of actions and
the notion of alternative actions. We saw that the definition of obligation made use of
a universal quantification over actions, which has the same drawback in section 4.2.3.2.
However, with this explicit quantification, we could provide an alternative, stating that
an obligation on an action holds if some alternative to the obligated action is prohibited.
In this case, we have deontically underspecified actions.

[29] makes somewhat similar claims and adds some additional observations, most as stip-
ulations which are not supported by intuitive evidence or argument. For [29], the notion
of action negation must have an intuitive interpretation as a combinator on actions, does
not impose restrictions on other action combinators such as sequence, and which reflects a
notion of refraining. As well, the negation of an action $\alpha$ should be different from and an
alternative to $\alpha$. It is claimed that [130] does not satisfy all these requirements, primarily
because the algebraic approach in [130] leaves open a variety of alternative interpretations
of action negation. [106] is not cited. [29] addresses issues that are not directly relevant
to our line of inquiry, namely, free choice permission and more particularly relativized
action negation. For the latter, [29] provides several different versions of dynamic logic
with respect to different modal logical frames; for each version the action combinators are
defined, among them action negation. In other words, the definition of action negation is
relativised to alternative definitions of the other action combinators. Semantically, actions
are relations between states and we can refer to the reachable state-space of an action as
those states attained as a consequence of the execution of the action. In all the alternative
definitions of action negation, it is understood in terms the “relational complement with
respect to all possible relations over the reachable state-space” and defined as “refraining of
$\alpha$ to be the act of ‘doing anything but $\alpha’$.” [29]. Relativised action negation is then required
as the reachable state-space varies with respect to the modal logical frame. However, given
that [29] makes use of complement, it is unclear how deontic specifications that make use
of action negation then leave some actions deontically underspecified. Rather, the concep-
tion of action negation would appear to suffer from problems to those discussed above. As
we discuss below, this is somewhat misleading since it is not relational complement with
respect to the total space of actions, but to the relativised alternatives. A final point here
is that it is unclear how to give a *specific* action that is the negation of another, which would be important in any implementation.

[29] observes that action negation ought not to impose any restrictions on other action combinators. In other words, for any action, simple or complex, we ought to be able to apply action negation to it. In particular, the negation of a sequence of actions ought to yield some action that is somehow considered to be the negation of the sequence. For our purposes, it is crucial to be able to determine such an action for the definition of collective obligation. One caveat is that this should hold on the condition that action negation is a *total* function: for every action (simple or complex), there *is* some action which is the negation of that action. It may be for a *practical* domain that action negation is partial; there may be actions for which action negation is not defined.

The relevance of this to deontic specification is that where obligations on an action imply that the negation of that action is prohibited, i.e. there is no such negation, there cannot be an obligation on the action. Intuitively, if *Bill is in the room* and there is *no possibility* *not be in the room*, say he is a prisoner in a secure facility, then *Bill is obligated to be in the room* as an obligation on Bill is vacuous since he cannot execute any alternative action such that he is not in the room. None of the actions Bill executes in the room can violate this obligation. Where an obligation cannot be violated, it does not hold.

In defining alternative actions, [29] presents several requirements. It is pointed out that we must specify action *identity* and the relationships between actions. Two actions $\alpha$ and $\beta$ are identical in the semantic sense if and only if $\alpha$ and $\beta$ are interpreted as the same relation over the state-space. They are different otherwise. We observe that this is an *extensional* notion of action identity. However, an *intensional* difference may be important: for example, the actions of *buying* and *selling* are extensionally equivalent in terms of the states they reach, but are intensionally distinct. One way to intensionally distinguish actions is to introduce a *label* or *name*. But this is a somewhat minor point for our purposes.

An additional stipulation is that *alternatives to an action are only possible if the primary action itself is possible* [29]. We interpret this to mean that an action and the negation of an action must both be executable from the same states, though they reach different states. This point relates tightly to our notion of antonyms: remaining in the room and leaving the room are antonyms in that both have the same state in which they are executable (being in the room), though the results (being in the room v. not being in the room) are distinct.
Another requirement is that different actions may not be alternative actions; in particular, actions which are in strict subset relations are different but not alternatives. Here [29] illustrates the point with explicit reference to preconditions and postconditions. Suppose propositions $P$, $Q$, $R$, $S$, and $\alpha$ and $\beta$ are actions. Furthermore, suppose $\alpha$ has preconditions \{P, Q\} and postconditions \{R, S\}, while $\beta$ has preconditions \{P\} and postconditions \{R\}. Intuitively, if Bill is obligated to run in the room, then Bill’s moving in the room is a different action in that it is more general, but it cannot be an action which would be prohibited or result in a violation since Bill’s running in the room implies his moving in the room.

A final requirement is that the negation of an action is specific and unique, though it is unclear exactly what motivates this or how to determine it in the framework of [29], though empirically this often appears to be the case as shown by Thesauri ([72]).

Where deontic specification is given using this a notion of action negation relative to alternative actions, it implies that there are actions which are not necessarily deontically specified, which are those actions that do not meet the specification of alternative action.

Considering obligations on actions, we specify many of the same requirements. However we provide these in terms of lexical semantic functions which, for any action, calculate the antonym of an action, if there is one.

4.6 Summary

In this chapter, we introduced ScDL. We made clear a range of interlocking problems for ScDLs with respect to the central issue in Deontic Logic, the CTD structure, which have not previously been discussed. These problems bear on changing primary obligations, action negation, and the relationship between obligations on sequences and sequences of obligations. To address these problems for the CTD structure, we introduced a novel proposal for deontic operators in an ScDL analysis. The proposal reduces deontic operators on atomic and complex actions to fine-grained markers for violation and fulfillment which are sensitive to the structure of the action they apply to. It is a flexible and extensible framework. Along those lines, we provided novel, alternative definitions for obligations on sequences. We used our analysis of the deontic operators to account for the CTD structure without homogenising obligations on sequences and sequences of obligations. A range of outstanding issues were discussed and potential solutions suggested. We reviewed and analysed two basic, key proposals for ScDL, highlighting differences and similarities as well as identifying those elements to carry over into our implementation, discussed in
chapter 5. Finally, we discussed action negation, which is a key problem in ScDL. We reviewed the issues and concluded that our approach would be based on lexical semantic notions such as antonym rather than the logical notion of propositional negation or set-theoretic complementation.

4.7 Next Chapter: Introduction to the Implementation

In the next chapter, we introduce the implementation. We first relate the structure of the program and results to the problems discussed earlier in the thesis. This is followed by an overview of the programming language (Haskell) and of the implementation. Each module of the program is presented by way of code and examples.
Chapter 5

The Abstract Contract Calculator

5.1 Introduction

To this point, we have considered a range of aspects of contracts, from the broad issues of the elements of the organization of legal contracts and the analysis of the language of expressions which appear in legal contracts, to narrower issues of logical form and semantics that arise in SwDL and ScDL, particularly as they bear on the central CTD structure and the analysis of sequences of obligations and obligations on sequences of actions. We have concluded that it is not clear what the definition of the operators should be in general or with respect to a particular application, as different applications may require different definitions – those in mathematics or engineering applications being different from those for electronic commerce or legal contracts expressed in natural language. There is, a priori, little reason to expect that there is but one definition of the deontic operators that suits all purposes at all times; the diversity of opinions with the research literature attests well-enough to this. Moreover, considered abstractly, the deontic operators are but one way in which the executions of actions may be associated with some value, leading to a more general view of value operators that apply to actions. However, as these values are not intrinsic to the action, but arise as a consequence of the application of a value operator to them, we can speak of secondary properties associated with the action. Furthermore, considerations of the CTD structure lead us to suppose that deontic specifications are mutable for they can be introduced or eliminated. Finally, whether an action gives rise to a violation or fulfillment, as well as what deontic specification holds, is context dependent as we discussed in section 4.4.2.4.
Our earlier analyses and explorations have lead to a conclusion that we do not yet have an analysis of the deontic operators which is formalisable in the logical sense. Hence, before embarking on such a formalisation, we need a tool which can be used to explore the space of alternative formulations where deontic specifications are mutable, context dependent, secondary properties which are associated with the executions of actions. It is, then, desirable and useful to have an engine which allows one to exercise the different definitions of the value operators in order to explore the outcome, as well as to have a tool that could be extended to consider other issues. We provide a tool to experiment with different definitions of value operators which generates data as a result of executing actions relative to some value specifications and rules that update the value specifications. The data represents which agents executed which actions and when as well as whether the agent’s actions fulfilled or violated a particular deontic specification. We then have a clear measure of risk, interpreted as the number of violations, for a particular agent relative to a set of value specifications and rules. Given the results one desires, one can then fix a system and provide an axiomatisation, though this is not our goal here.

In this chapter, we provide an overview of the Abstract Contract Calculator (ACC), which is a tool that allows one to execute actions relative to a contract, thus operationalising a contract ([192]). Discussion of the tool is related to model theory rather than proof theory; that is, we provide particular models rather than axioms and proofs. The tool itself provides schematic models which, according to one’s choices of how to define components, can be instantiated. Actions can then be executed with respect to the instantiation. The choices one makes are distinct from the tool in the sense that one determines what properties one wants of the system, which can then be represented in the tool and subsequently executed. One can understand these as meta-constraints that one realizes in the tool. However, certain choices are built-in and not amenable to such constraints.

Two examples will help to clarify these points. The latter point first. As a basic choice, the execution models action execution relative to a contract. It is, then, in contrast to a state-wise modal logic where operators are semantically defined with respect to accessibility relations between possible worlds. The second point shows that one can operationalise alternative definitions; the tool allows one different ways to define deontic specifications on complex actions which introduce complex violation and fulfillment markers. Or, alternatively, one could have simple markers or have only violation markers. Different choices in this space would determine different models of the notions one has in mind.

In the course of the presentation, we distinguish what the tool does and does not do, what that capability is used for, and what alternatives are available.
Chapter 5  

5.2 Bridge to the Abstract Contract Calculator

We have considered a range of observations and issues related to reasoning with the deontic concepts. In section 5.2, we review some of the key points we have drawn in previous chapters and how they relate to our tool. In the subsequent section 5.3, we introduce the main elements of the tool, a walk-through example, and a diagram of the flow of information. As deontic specifications can be understood as particular instantiations of *value operators applied to actions*, main elements are presented more generally. In section 5.4, we then briefly discuss the programming language, Haskell, used in the implementation. Following this, in section 5.5, we provide a more detailed discussion of each of the modules, giving the design decisions for the fixed aspects of the tool, meta-decisions that guide the particular instantiation of the tool, and examples. However, to keep a focus on the discussion, we target only obligations on actions and, when we discuss complex actions, only sequences. Having provided information about the operation of the program, we address general issues related to the formalisation of the domain of application in section 5.6. Finally, we discuss related systems in section 5.7. Given space constraints, we note where additional topics are not presented in the body of the thesis.

5.2 Bridge to the Abstract Contract Calculator

In this section, let us recall key elements of the previous chapters to highly the relationships between the topics previously discussed and the ACC. We present issues relating to dynamic languages, *obligation* reduced to violation and fulfillment, obligations on sequences, action negation, and negations of deontic specifications.

5.2.1 A Dynamic Language

Our first basic choice is to adopt a dynamic rather than a static language. As we have said, one reason for the choice is that the dynamic approach avoids the family of paradoxes that arise in the static approach, in large measure because a dynamic approach does not import the problems of propositional modal logic. Moreover, we have argued on linguistic grounds to reject some of the key defining clauses of a static logic, such as Axiom K or Necessitation. In other cases, we have provided linguistic analyses of paradoxes such as the *Good Samaritan Paradox* and the *Gentle Murderer Paradox*. In yet other cases, we have argued that, for legal contracts in natural language, it is untenable to maintain that there are no conflicting obligations or that there are vacuous deontic specifications, which are those which cannot be executed. In this light, *permissions* need to be understood
as rights; however, as we do not consider interactions between agents, we cannot have a representation of rights, and therefore settle for a vacuous representation of these.

However, we have adapted, not adopted, previous accounts of dynamic languages. Most importantly, our analysis is tied to the interlocking notions of deontic specification, complex action, action negation, and lexical semantics. To clarify this, we discuss each of these elements.

5.2.2 Directed Violation and Fulfillment

We argued that deontically specified actions should be analysed in terms of violation and fulfillment so that we reason with the markers and their consequences directly in the language. For our purpose, we have deontic specifications directed at specific individuals rather than allowing deontic specifications with collectives or quantified expressions. However, in contrast to previous approaches, and as we have argued earlier, we do not want to represent such individuals as incorporated into the operator, for example, as in the SDL representation OB_{bill}P, meaning that Bill is obligated with respect to P. Rather, we want another means to associate an individual with the operator. Such a representation ought to, in principle, allow an extension of the system to address quantifiers over a domain of individuals and groups, but we do not consider this in the tool.

5.2.3 Obligations on Basic Actions

In this view, we want distinct representations of abstract forms which correspond to the expressions in Example (125), where one agent has one obligation and another agent has another obligation.

Example 125

\begin{enumerate}
\item Bill is obligated to push the left toggle up.
\item Jill is obligated to push the right toggle right.
\end{enumerate}

Here it is important to emphasise that these examples provide intuitive, informal, and natural language analogs to what appear in the tool. In particular, we are not providing representations of the semantics of pushing the left toggle up or pushing the right toggle right, but rather an abstract action represented in terms of a labeled transition function.
This is similar to the abstraction in Propositional Logic, where intuitions are derived from sentences, but only propositional variables appear in the formalism, and similarly in action languages, which abstract over natural language action expressions. While we only consider natural language predicates such as *pushing the right toggle right* as our basic unit of analysis, following formal natural language practice, we could provide an even more fine-grained analysis. We believe the tool could be extended to support such decomposed and articulated fine-grained distinctions, leaving the overall architecture intact.

### 5.2.4 Spelling out Violation and Fulfillment

In a reductionist analysis, a violation or fulfillment marker arises as a consequence of the execution of one action or another. In addition, we have argued that we want a fine-grained analysis of the violation and fulfillment markers, indicating at least the agent, the deontically specified action, and the operator. Where the deontic operator applies to a complex action, and we choose to represent the marker relative to that complex action, then this too is included in the marker. To realize this, we distinguish the execution of the action per se from the association of that action with a violation or fulfillment marker. Informally, we have examples for a simple case.

**Example 126**

a. Bill’s *pushing the left toggle up* is associated with a fulfillment of his obligation to *push the left toggle up*. Bill’s not *pushing the left toggle up* is associated with a violation of his obligation to *push the left toggle up*.

b. Jill’s *pushing the right toggle right* is associated with a fulfillment of her obligation to *push the right toggle right*. Jill’s not *pushing the right toggle right* is associated with a violation of her obligation to *push the right toggle right*.

### 5.2.5 Obligations on Sequences

As we have discussed, there appear to be several ways to construe a deontic specification on a complex action such as a *sequence of actions*, and we have argued that, in the face of the alternatives, we want a tool that does not commit us to one representation or the other, but allows the alternatives to be represented and executed. The other complex action combinators are *choice, combination*, and *negation*. The issues at stake are exactly of what the violation or fulfillment marker marks a violation or fulfillment and when that
marker arises. These distinctions appear in the *distributive*, *collective*, and *interruptable* interpretations of obligations on sequences. We articulate Example (127a) as in Example (127b).

**Example 127**

a. Bill is obligated to push the left toggle up and then to push the right toggle right.

b. Bill’s not pushing the left toggle up is associated with a violation of his obligation. After pushing the left toggle up, Bill’s not pushing the right toggle right is associated with a violation of his obligation. After Bill’s pushing the left toggle up, Bill’s pushing the right toggle right is associated with a fulfillment of his obligation.

The different interpretations arise from what we understand *his obligation* to be. In the distributive reading, it refers to an obligation on *each action separately*, while in the collective and interruptable interpretations, it refers to the obligation on the sequence *per se*. Furthermore, interruptability arises where we do not require that the sequence be executed in one step and that there are alternative actions that may be executed in the intervening time, which do not introduce violations or fulfillments of the sequence.

We have argued that we should distinguish between sequences of obligations and obligations on sequences. The main difference between them is with respect to the violation or fulfillment markers: a sequence of obligations has violation and fulfillment markers relative to the particular actions each deontic specification applies to; an obligation on a sequence has violation and fulfillment markers relative to the sequence *per se* which is a marker on a complex expression. We view this analysis as an in depth case study of and argument for complex violation and fulfillment markers. While we believe the argument could be made to bear on other topics such as time and tense, obligations and sequences raise central issues that have been extensively discussed and also connect separate approaches to the deontic concepts. In addition, making the case for such complex markers and providing a means to generate and reason with them opens the way for extensions to these related topics.

### 5.2.6 Action Negation

We have discussed *action negation* and argued against taking this as set-theoretic complementation with respect to the domain of actions. Rather, we want a localized definition. Our model for this is a *lexical semantic* relation of *antonym* found in natural language.
One reason for this is that we want to partition the space of actions into those which induce a violation, those which induce a fulfillment, and those which have no deontic consequence. We also want a definition which fulfills the symmetry of obligation, whereby the actions which fulfill and violate Bill’s obligation with respect to an action are symmetrical with those actions which fulfill and violate Bill’s obligation on the antonym of that action. Finally, it ought to be possible to calculate the antonym of complex actions.

In defining obligation on an action, we suppose that the given action fulfills the obligation while the antonym of the given action violates it. We see this in the following two reductions.

**Example 128**

a. Bill is obligated to push the left toggle up: Bill’s pushing the left toggle up fulfills his obligation and Bill’s pushing the left toggle down violates his obligation.

b. Bill is obligated to push the left toggle down: Bill’s pushing the left toggle up violates his obligation and Bill’s pushing the left toggle down fulfills his obligation.

Bill’s obligation to push the left toggle up is reduced to two cases – one of pushing the left toggle up and another of pushing the left toggle down. Similarly, we have Bill’s obligation to push the left toggle down is reduced to pushing the left toggle down and pushing the left toggle up.

### 5.2.7 Relationship between Notions

Let us return to the interlocking notions of deontic specification, complex action, action negation, and lexical semantics. Every obligated action provides a partition of the space of actions into those which, if executed, either fulfill or violate the obligation or have no deontic consequences. In principle, supposing a finite set of atomic actions, we could have a function which maps an input action to its negation. For this purpose, we could use natural language actions.

However, for complex actions, the negation of the complex action depends on the two input actions and the combinator. Though combinations of more than two are in principle possible, we only discuss a complex action comprised of two actions.

To calculate the negation of a complex action, we have a function that calculates the negation of any action (should there be one), given the explicit precondition and postcondition
properties of the actions and those of the combinator. Furthermore, we pointed out that we want to allow deontically underspecified actions, which ought to be differentiated from operators which have a vacuous secondary property. It is unclear how one would express such an analysis without having definitions of actions that do not explicitly represent the precondition and postcondition properties in relation to the deontic notions and markers. An additional advantage of representing the actions in this way is that the actions have a structure whereby we can specify actions which subsume other actions; the deontic specifications can be sensitive to such actions. Moreover, if we introduce a novel action into our model, we can calculate its negation (supposing there is one).

5.2.8 Negations on Deontic Specifications

In addition to action negation, we may want negations on deontic specifications. In section 4.3.6.1, we discussed interdefinability, consistency, and implications among deontic specifications. These notions would require some conception of negations of deontic specifications themselves. In this section, we briefly review these matters and suggest how they relate to the implementation, though this is not presented in the thesis.

We argued that there are two distinct negations: we want a negation which relates to a deontic specification which is not a component of a given contract, but does not imply some other deontic specification which is; and we want a negation which allows us to express definitional relationships between the deontic operators. For the first, we assume a form of negation which essentially tests whether the given deontic specification is an element of the contract; where it is not, we do not want to imply any other deontic specification. For example, suppose the contract is comprised of Bill is obligated to push the left toggle up and Jill is obligated to push the right toggle right. With respect to this contract, it is true that it is not the case that Bill is permitted to push the left toggle down, for this permission does not appear in this contract. Under such an interpretation of negation on a deontic specification, we do not want to infer that Bill is forbidden to push the left toggle down; rather, it is underspecified. This is like action negation in that we explicitly want to allow that there are some expressions which are not deontically specified.

On the other hand, we might well want to infer in other cases that indeed a negated deontic specification implies another expression: If Bill is not permitted to push the right toggle right, then we may want to infer that Bill is forbidden to push the right toggle right. We express these as meaning postulates, which constrain the operators to relationships between two expressions. These are not general logical relationships, but must be stipulated to allow a representation of one’s view of the relationship. We could, for instance, adopt the SDL
or DDL equivalences of the operators. However, given limitations of our presentation, we
do not discuss negation of deontic specifications further in the thesis.

5.2.9 CTDs

We want our implementation to address CTDs. First, we want to make deontic specifica-
tion on an action context dependent; that is, in one context, Bill’s obligation to move
the left toggle up holds, but not in another context. Moreover, the specific change should
depend on what has been executed. In other words, supposing that Bill’s obligation holds
in a context, then, where he moves the left toggle down, we can say that Bill’s obligation
is removed and another obligation is introduced. While the discussion in the literature
focuses on CTD structures, we view CTD structures as but one way in which deontic
specifications can be introduced and eliminated from a contract. We want to allow ac-
tions which are not deontically specified to do so as well. Exactly how such updates of
deontic specifications are provided depends on what one wants the system to represent;
the tool provides a means to provide and execute such alternative representations.

5.3 Conceptual Overview of the Tool

In this section, we provide an overview of the tool in three parts. First, we give a walk-
through simple example of an obligation on an atomic action. We make several assump-
tions here for the sake of presentation, though alternatives are possible. We then outline
the main components. Then, a schematic representation is given of the execution of an
action relative to a contract and a state-of-affairs.

There are two elements we do not overview in this section: how we create the deontic
specifications on atomic or complex actions, though their essential structure was provided
in section 4.3.3; how we provide antonyms of actions, which hinges on the structure of
actions and the lexical semantic functions. These elements are presumed in this section,
but discussed in depth in section 5.5. Finally, for the purposes of the walk through, we
suppose natural language actions such as move the left toggle up rather than the abstract
actions we present in the tool. To make the components clearer in the following section,
we indicate our abbreviations.

We have also chosen to represent the overview in terms of the more abstract and general
notion of values, where the deontic concepts are but one possible representation of the
realisation of such values.
Chapter 5 5.3 Conceptual Overview of the Tool

5.3.1 A Basic Example

In this section, we consider a simple example of an obligation on an action. We first provide the elements of the example, then a walk through with a graphic.

5.3.1.1 Deontic Specification on an Action and a State-of Affairs

Suppose our working example is *Bill is obligated to move the left toggle up*. Furthermore, we suppose that the antonym of *Bill’s moving the left toggle up* is *Bill’s moving the left toggle down*. We call both *agentive actions* (AGAC), where *Bill* is the agent (AG), and *moving the left toggle up* and *moving the left toggle down* are actions (AC). We have a *state of affairs* (SOA) represented in terms of current properties and time. We have not represented or used a *world* index here, though that appears in the tool to allow for other extensions.

We adopt a *reductionist* approach to deontic specifications on actions: *Bill’s obligation to move the left toggle up* is reduced to expressions which appear in a *valued specification list* (VSL), which is a list of *valued action specifications* (VAS). VASs are statements flagging actions to violation and fulfillment markers. For our purposes here, we suppose *Bill is obligated to move the left toggle up* reduced to a VSL comprised of two VASs: *Bill’s moving the left toggle up fulfills his obligation to move the left toggle up* and *Bill’s moving the left toggle down violates his obligation to move the left toggle up*. Thus, doing the obligated action is flagged with fulfillment of the obligation, while doing the antonym of the obligated action is flagged with violation of the obligation.

We distinguish between the action per se (moving the toggle up or down) and the secondary effects which it gives rise to due to its deontic specification (violations and fulfillments of obligations). In other words, if *Bill’s moving the left toggle were not* deontically specified, then just those changes usually induced by the action would follow since there are no associated components of a deontic specification. We can say that the deontic concept ascribes a value to the action executed by an agent. Thus, the language of violations and fulfillments allows us to flag states according to the secondary effects of actions relative to a value on that action and agent.

5.3.1.2 The History

Suppose that *Bill* only moves the left toggle either up or down. We assume that the context is such that he can move the toggle up or down; say the SOA specifies that the
left toggle is in neutral. He does one of them. We record not only which action Bill has executed, but also the associated VAS, if any. The mechanics of this association are achieved by matching the action which is executed to components of a VAS. To record the result, we suppose a history (H), which is a list of historical records (HR) that represent what has happened. For example, supposing that Bill executes the action move the left toggle up. We record in the history the historical record that he has moved the left toggle up. In addition, we record in the history the historical record that Bill’s moving the left toggle up has fulfilled his obligation to move the left toggle up. Furthermore, Bill’s action has changed the SOA; the left toggle is up.

Note that we distinguish what holds in the SOA from what holds in the H. This facilitates processing, for the properties in the SOA change (are introduced or eliminated) while we only add records to the H. By the same token, the time of the SOA is changed as a result of the execution of an action, while the HRs have a time indication (e.g. Bill moved the left toggle up at time 2).

5.3.1.3 The Rule Function

To this point, an action which has been executed and the associated VAS have been recorded in the history. With the CTD structure, we have been concerned with the consequences which follow from the VAS; that is, it being the case that Bill has violated his obligation to move the left toggle up, some other deontic specification comes to hold. In jurisprudential terms, this represents the relationship between a contractual breach and its remedy.

We make the following simplifying assumptions for the purpose of illustration. We have two rules which appear in a rule function (RF). If Bill’s moving the left toggle up fulfills his obligation to move the left toggle up appears in H, then Bill is obligated to move the right toggle right is introduced into the VSL and Bill is obligated to move the left toggle up is removed from the VSL. If Bill’s moving the left toggle down violates his obligation to move the left toggle up appears in H, then Bill is obligated to move the right toggle left is introduced into the VSL and Bill is obligated to move the left toggle up is removed from the VSL. There are two assumptions in this example.

First, we assume that moving the right toggle left and moving the right toggle right are antonyms. Thus, the VASs can be spelled out; for instance, the VASs of Bill is obligated to move the right toggle right fulfill his obligation
to move the right toggle right and Bill’s moving the right toggle left violates his obligation to move the right toggle right.

Second, we assume that introduction and removal of obligations as given in the VSL; this is for illustration, and there are other alternatives available. But, it is this operation which animates the CTD structure. We can say that the rules in RF are given according the body of law or pragmatically and are not a general or systematic matter. We concern ourselves only with the abstract formal language which allows one to express the sorts of relationships the designer might want.

5.3.1.4 Updating the VSL with respect to a Triggered RF

Let us consider the relationship between the rules and the VSL. The rules are triggered by what appears in the history H (although other possibilities are available, as we show later). Thus, if we find the HR in H Bill’s moving the left toggle left toggle up fulfills his obligation to move the left toggle up, we trigger the rule with that HR. Consequently, as given above, Bill is obligated to move the right toggle right is introduced into the VSL and Bill is obligated to move the left toggle up is removed from the VSL.

Note that we have three components (SOA, H, and VSL) which can be changed, but each in distinct ways: the SOA has updates of properties that follow from the action itself and of time; the H records the actions which are executed as well as the deontic information that follows, neither of which change; the VSL represents the deontic specifications, which can change. We have one component which, by hypothesis, does not change the RF. However, this is a working assumption of our model and could allow further manipulation.

As we see, the CTD structure is treated in terms of a cascade of triggers, implications, and revisions of states, some of which are “external” (as in temporal update) and some “internal” (no temporal update, but expressions within the state are modified).

5.3.1.5 The Contract and the Value Specified Context

In our view, the notion of contract is comprised of the valued action specifications and the rule function. Thus, the contract represents the key elements of the CTD structure. The only element which is missing is the particular action that is executed with respect to the contract. More accurately, the contract is the RF along with that VSL as it is initialised and before the execution of any action. Each action execution which changes the VSL represents a contract state. Given a finite initial VSL, a finite set of rules in the RF, and
a finite set of executable actions, we can generate the possible contract states. Each of the
branches through the space of contract states that result from the execution of an action
represents a \textit{trail of execution of the contract}.

The most general component of our implementation is the \textit{Value Specified Context (VSC)},
which is comprised of the \textit{SOA}, the \textit{VSL}, the \textit{RF}, and the \textit{H}. An action is executed with
respect to a \textit{VSC} using an execution function. Thus, what we are calling an action
represents not the action itself, but the description of the action.

5.3.2 A Graphic Representation

In Figure 5.1 we have a graphic representation of two alternative trails of execution from
an initial \textit{VSC} – \textit{VSC1}. \textit{VSC1} represents a state in which the \textbf{Properties} represent that
the \textit{left toggle} is in a neutral position, the \textbf{Time} is 0, nothing has previously occurred to be
marked in the history \textit{H}, and the \textit{RF} and \textit{VSL} contain the rules and deontic specification
outlined in section 5.3.1. We abbreviate \textit{left toggle} with “LT” and \textit{right toggle} with “RT”.
The alternative trails of execution are outlined below, the first where the obligation is
violated and the second where it is fulfilled.

5.3.2.1 Violating the Obligation

Following the left branch, we are following the branch in which \textit{Bill moves the left toggle
down} rather than up. This is the action which violates the obligation to move the left
toggle up, which is represented in the \textit{VSL} in \textit{VSC1}.

The result is \textit{VSC2}: the \textbf{SOA} indicates that the property the left toggle is down and
the time is updated with 1; the \textbf{H} represents that \textit{Bill has moved the left toggle down}
and, relative to the \textit{VSL}, \textit{VAS2} holds, which represents \textit{Bill’s moving the left toggle down
violates his obligation to move the left toggle up}.

As a consequence of what holds in \textbf{H} and relative to the rules in \textbf{RF}, we update the
obligation in \textit{VSL} a per the rule which is triggered. This gives us \textbf{VSC2.1}. Note that
neither the \textbf{SOA}, the \textbf{RF}, nor the \textbf{H} are modified. However, the \textit{VSL} shows that the
representation for \textit{Bill is obligated to move the left toggle up} has been removed and the
representation for \textit{Bill is obligated to move the right toggle left} has been introduced.
Chapter 5  
5.3 Conceptual Overview of the Tool

**VSC 1**

**SOA**

Properties = \{LT neutral\}

Time = 0

**VSL**

VAS1 = Bill's moving the LT up fulfills his obligation to move LT up

VAS2 = Bill's moving the LT down violates his obligation to move LT up

**RF**

If VAS1 in H, then add to VSL that Bill is obligated to move RT right

If VAS2 in H, then add to VSL that Bill is obligated to move RT left

H

Nothing has happened to be recorded in history

---

**VSC 2**

**SOA**

Properties = \{LT down\}

Time = 1

**VSL**

VAS1 = Bill's moving the LT up fulfills his obligation to move LT up

VAS2 = Bill's moving the LT down violates his obligation to move LT up

**RF**

If VAS1 in H, then add to VSL that Bill is obligated to move RT right

If VAS2 in H, then add to VSL that Bill is obligated to move RT left

H

Bill has moved LT down at time 0

VSL2 at time 0

---

**VSC 2.1**

**SOA**

Properties = \{LT down\}

Time = 1

**VSL**

VAS3 = Bill's moving the RT left fulfills his obligation to move RT left

VAS4 = Bill's moving the RT right violates his obligation to move RT left

**RF**

If VAS1 in H, then add to VSL that Bill is obligated to move RT right

If VAS2 in H, then add to VSL that Bill is obligated to move RT left

H

Bill has moved LT down at time 0

VAS2 at time 0

---

**VSC 3**

**SOA**

Properties = \{LT up\}

Time = 1

**VSL**

VAS1 = Bill's moving the LT up fulfills his obligation to move LT up

VAS2 = Bill's moving the LT down violates his obligation to move LT up

**RF**

If VAS1 in H, then add to VSL that Bill is obligated to move RT right

If VAS2 in H, then add to VSL that Bill is obligated to move RT left

H

Bill has moved LT up at time 0

VAS1 at time 0

---

**VSC 3.1**

**SOA**

Properties = \{LT up\}

Time = 1

**VSL**

VAS5 = Bill's moving the RT right fulfills his obligation to move RT right

VAS6 = Bill's moving the RT left violates his obligation to move RT right

**RF**

If VAS1 in H, then add to VSL that Bill is obligated to move RT right

If VAS2 in H, then add to VSL that Bill is obligated to move RT left

H

Bill has moved LT up at time 0

VAS1 at time 0

---

**Figure 5.1: Graphic for Actions Executed Relative to Contract**
5.3.2.2 Fulfilling the Obligation

Starting over again from VSC1, we follow the alternative, right branch. We suppose that Bill moves the left toggle up, which is that action which fulfills the obligation to move the left toggle up.

The result is VSC3. We have indicated temporal update with 1", though this could have been indicated with a world index instead (which is not included here). In VSC3, we see the Properties of the SOA as well as the H have changed. In particular, VAS1 holds, which represents Bill’s moving the left toggle up fulfills his obligation to move the left toggle up.

Given this, a different rule of the RF is triggered, updating the VSL as in VSC3.1. This represents the removal of the obligation to move the left toggle up and the introduction of the obligation to move the right toggle right.

5.3.2.3 Output VSCs and Complex Actions

The resultant VSCs are available for the execution of the next action such as moving the right toggle right or left. Depending on the action, Bill either violates or fulfills his obligation. As no rules in the RF can be triggered, no subsequent VSL can be provided. We might have presumed an RF which removes the remaining obligations, leaving the VSLs empty. In any case, if Bill moves the left toggle again, it does not indicate subsequent violations or fulfillments.

In this example, we have not considered complex actions. The essential idea is that we can represent such deontic specifications on complex actions by enriching the representation of VSAs as discussed in chapter 4 or by adding further rules to the RF.

5.3.3 The Components

In this section, we given an abstract idea of the components which appear in the implementation as well as a bird’s eye view of the architecture of the main stages of execution of the tool. While we present this in more abstract terms of values, the specific examples are in terms of our deontic operators.
5.3.3.1 Basic Components

We first provide the basic components:

- \( L \) represents the set of literals, which is the set of positive and negative atomic propositions such as \( \{prop1, prop2, ..., neg-prop1, neg-prop2\ldots\} \).

- \( AG \) represents an agent, which is understood to be an individual who executes an action. For example, we have a set of agents such as \( \{Bill, Jill, Phil, \ldots\} \).

- \( VOL \) represents a value operator label, which is a label for a value operator. For example, we have a set of value operator labels such as \( \{Obligated, Prohibited, Permitted, NULLVOL\} \).

- \( VF \) represents a value flag, which is a flag for a value and is associated with the execution of an \( AC \) by \( AG \) relative to a \( VOL \). For example, we assume a set of value flags such as \( \{violation, fulfillment, NULLVF\} \).

- \( ACLabel \) represents an action label, which is a label for an action. For example, we have a set of action labels labels such as \( \{Action1, Action2, Action3, \ldots, NULLAC\} \).

- \( AC \) represents an action, which is a labeled transition from a set of consistent literals of \( L \) (the preconditions) to a set of consistent literals of \( L \) (the postconditions). The label is drawn from \( ACLabel \). We assume some set of actions.

- \( AGAC \) represents an agentive action, which is an agent associated with an action. We assume some set of agentive actions.

- \( Time \) and \( World \) are, respectively, time and world indices. We assume a set of time indices \( \{0, 1, 2, \ldots\} \) and world indices \( \{0, 1, 2, \ldots\} \).

5.3.3.2 Complex Components

In terms of these, we describe the complex components, starting with the most inclusive.

- \( VSC \) is a value specified context, which is comprised of a \( SOA \), a \( VSL \), an \( H \), and an \( RF \).

- A \( SOA \) is a state of affairs, which is comprised of a set of consistent literals along with time and world indices.

\[1\text{We include the world index for development of the ACC, but do not utilise it yet.}\]
• **VSL** is a *valued specification list*, which is a list of **VASs**.

• **VAS** is a *valued action specification*, which is a record that associates an action, an agent, a value operator label, and a value flag. Valued action specifications are used to trigger a **VSL** update.

• **H** is a *history*, which is a list of **HR**.

• **HR** is a *historical record*, which is a record consisting of an action label (the action which occurred), the agent who executed it, the time it was executed, and the value operator label and value flag associated with the agent’s execution of the action, if there are any.

• **RF** is a *rule function* from elements in the **HR** to an output **VAS** which is the result of applying an *update function* to the input **VAS**. The elements of the **HR** in the function act as triggers for updating the **VAS**.

### 5.3.3.3 Functions on the Components

We have functions which apply to these components, giving examples for clarity. We provide details of the functions later, particularly those utility functions which are not specifically named here.

• **Do** is a function on agentive actions and a **VSC**. We test whether the agentive action is executable with respect to the **SOA** of the **VSC**. If the action cannot be executed, we *return* the **VSC**. If the action can be executed, it is executed, and we *update* the **SOA** relative to the action.

• Having updated the **SOA**, we have a function which tests whether the **VAS** associated with the agentive action is in the **VSL** of the **VSC**. If not, then we *update* the **H** of the **VSC**, indicating that the agentive action has been executed, and *return* the resulting **VSC**. If so, we *update* the **H** of the **VSC** relative to the **VAS** by adding to the **VAS** that the action has been executed.

• We have a function which tests whether the **HR** associated with the **VAS** matches a *trigger* in the **RF**. If not, then we *return** **VSC**. If so, then we *update* the **VSL** as given by the **RF** relative to the particular **VAS**. And then we *return** **VSC**. For example, suppose a **VSC** with **VSL** which contains the **VAS** *Bill is obligated with respect to Action1 and doing that action would flag for fulfillment*. Where the **RF** contains a mapping from *Bill has executed Action1 which is obligated and*
flagged for fulfillment to a function that removes from the current VSL the VAS representing Bill’s obligation with respect to Action1. The update is triggered where the HR contains the representation of Bill has executed Action1 which is obligated and flagged for fulfillment.

5.3.3.4 Graphic Representation

This sequence of processing steps is followed for any execution of any action, simple or complex, value specified or not. Figure (5.2) represents a schematic of the flow of control.
In the next section, we outline some of our reasons for adopting Haskell as the programming language for implementing the tool.

## 5.4 Haskell as the Language of the Implementation

We have implemented the tool in Haskell, which is a declarative, purely functional, lazy, statically typed programming language. As a declarative language, it emphasises what is to be calculated as opposed to how the calculation is to be carried out, as in imperative programming languages such as Java. Speaking broadly, it implements the Typed Lambda Calculus. Lazy evaluation means that expressions are only evaluated when they are needed and that if an expression is shared in several places, then when the expression is evaluated, the value appears in all those places. Static typing allows typing errors to be caught at compile time; typing also helps to document the system and facilitate understanding of the program since it is easy to identify the arguments into and results from a given function.

The main reason why we chose Haskell is personal; as we had experience using the Typed Lambda Calculus for our linguistic research on the syntax and semantics of adverbial modification ([189], [190], and [195]), it was a “natural” programming language for us and enabled us to express the tool in a manner and form with which we were more accustomed. We found it a good language in which to write a prototype; as we are not concerned with running an application over the internet or making it part of a module in a business application, we were not concerned with using a language such as Java. In addition, as a programming language for computational semantics ([63] and [176]), we found logical and representational tools which were familiar to us already implemented. By comparison, Prolog does not “natively” use the Lambda Calculus ([18]) or typing.

In addition, contrasting it to more widespread programming languages, Haskell has been argued to produce smaller, clearer, and more elegant programs which can be quickly understood as well as modified ([151]). As a functional language with types, it is a programming language which is close in form to the formal language used to express a logic, and thus, it is easier to present to readers with a formal, mathematical background.

In the following, we give an overview of relevant aspects of Haskell, leaving more particular aspects to be presented concretely in the body of the text. We provide examples of the main elements of the Haskell language which we use: types, functions, records, list comprehensions, pattern matching, and set functions.
In Haskell, as a functional programming language, all computations are evaluations of expressions, which are syntactic terms, to yield values, which are abstract entities. Values are associated with a type. Examples of atomic expressions are integers, such as 5, and of atomic types, such as `Integer` and `Char`; we can have user-defined and complex types as well. Functions are expressed as equations, where the value of a function applied to a value yields, or is reduced, to some other expression or value. Functions also have a type signature declaration. For example, we can define the function `increment` as `inc`, where the type signature appears in the first line, meaning `inc` is a function from expressions of type `Integer` to expressions of type `Integer`, and the equation appears in the second line.

```haskell
Code 1  inc :: Integer → Integer
        inc n = n + 1
```

One can also introduce types which are based on basic Haskell types. For example, we introduce the type `Time` which has type `Int`. While both `Int` and `Integer` are whole numbers, those of type `Int` are limited in size to 32 bits, while those of type `Integer` are unlimited in size (apart from memory limitations). While this seems redundant, it clarifies what our functions do by labeling the inputs and outputs appropriately.

```haskell
Code 2  type Time = Int
```

We make extensive use of record types, which are comprised of labels and fields that hold values. For instance, in `Code 3`, we define a type `SOA`, which is a record with labels for `properties` with a field of type `PropList`, `time` with a field of type `Time`, and `world` with a field of type `World`. We define these types later. We can access particular fields of a given record via the label. For instance, in `Data 1`, we have an example, where `input>` and `output>` stand for the input and output at the prompt. The expression `#properties` amounts to a function from the `SOA` record to the value associated with that label in the record. In this case, we want the value of the `properties` field, which is a list of strings.

```haskell
Code 3  type SOA= Rec (properties :: PropList, 
                        time :: Time, world :: World)

Data 1  input> #properties (properties = [prop1, prop7, prop5, neg-prop3], 
                          time = 2, world = 4) 
        output> [prop1, prop7, prop5, neg-prop3]
```
We can apply these access functions to draw out the values of fields which are embedded in other records, as we see later.

We use list comprehensions, which essentially provide us the list of expressions, relative to a generator, which satisfy a given property or relation. They are expressions of the form \[ [ x \mid x \leftarrow P ] \] and are analogous to the set-builder notation of set theory, where for \( S = \{ x + 2 \mid x \in \{1, \ldots, 5\} \land \text{odd}(x) \} \), the result is \( S = \{3, 5, 7\} \). List comprehension works much the same way, but using lists rather than sets. In Data 2, the input says to form a list of ordered pairs, where the first element is generated from the list \([1,3,5]\), the second element is generated from the list \([2,4,6]\), and the first element is strictly less than the second element. The output gives the result.

Data 2  
\[
\text{input}\to ([x,y]\mid x\leftarrow [1,3,5], y\leftarrow [2,4,6], x<y]
\]

\[
\text{output}\to [(1,2),(1,4),(1,6),(3,4),(3,6),(5,6)]
\]

Haskell makes use of pattern matching: in a function application, the pattern of the definition of the function is matched to the actual parameters applied to the function; where the actual parameters are of the right type, then the function is evaluated with respect to the values of the parameters; the result of the function application is produced; otherwise, the function application fails.

While lists are a basic Haskell data type, we often want to use sets, primarily to void any effects of ordering between lists but also to use set-theoretic functions rather than list functions. Essentially, we use functions that allow us to convert lists to sets (and vice versa) using modules from ([63] and [176]).

These are the main elements of Haskell. To facilitate readability, we have distinguished code snippets from data snippets, which illustrate runs of the program. Similarly, we remove the scare quotes from around expressions of type String. Other elements are introduced as needed.

5.5 Overview of the Implementation

In section 5.3, we gave an overview of the tool. In this section, we present the key elements of the code of the prototype implementation along with example executions. The main, relevant “auxiliary” functions are discussed and provided, referring to or omitting those which are less relevant. As a tool, it provides a language in which alternative notions of value specification on agentive actions can be systematically examined and animated.
5.5 Overview of the Implementation

From this section, one should take away a good understanding of the parts of the tool, how it functions, and how alternative definitions for the deontic notions or actions could be represented and reasoned with.

The implementation runs from the command line. Essentially, one enters at the command line an agentive action, a value specified context, and the execution function. The output is a value specified context, so another action can be executed with respect to it, which is done manually. We have example program executions.

We construct first the simplest elements and how we work with them, leaving the more complex constructions, such as execution of actions relative to deontic specifications on complex actions, until the end of the presentation. This facilitates understanding of what each component contributes to the overall functionality. We are explicit about what additional elements are found in the implementation and what are meta decisions which lead us to design the implementation in a particular way or express how we understand elements of the tool.

Thus, we start with states of affairs in section 5.5.1, then construct agentive actions in section 5.5.2, and then add lexical semantic functions in section 5.5.3. Deontic specification on actions (the valued action specifications), which depend on these preparatory elements, do not appear until section 5.5.4. Discussion of the execution of an action relative to a list of deontic specifications appears in section 5.5.5, after we represent how these are recorded in a history. Then, in section 5.5.6, we outline the interactions between the execution of an action, the list of deontic specifications, and a rule function. This provides our analysis and implementation of the CTD structure. Finally, we discuss deontic specifications on complex actions in section 5.5.7. The presentation has some deliberate redundancy which we intend to help the presentation by moving from simpler structures to the more complex. So, actions, execution functions, and deontic specifications are discussed more than once, but in increasingly elaborate ways.

5.5.1 States of Affairs

We construct many of our expressions from basic Haskell types for strings String, integers Int, and records, which are labels associated with values of a given type. In terms of these, we have several derived types. [String] gives the type which is a list of expressions of type String; [SOA] is a list of expressions of type SOA.
Our atomic literals are of type \texttt{String} such as \texttt{prop1} and \texttt{prop2}. Prefixing a string with \texttt{neg-} forms the negation of a literal. We have a \textit{double negation elimination} rule \texttt{negateOrDoubleNegationElimList} such that \texttt{neg-neg-prop1} in a list reduces to \texttt{prop1} in that list. Given a finite list of positive and negative literals, we generate the Cartesian product, representing the list of all sublists of the literals; these are lists of type \texttt{PropList}. We filter these sublists for consistency, meaning that no sublist has a literal and its negation in it. Maximal lists can be taken to be \textit{possible worlds}; we take the nonmaximal lists as abbreviations for those possible worlds of which it is a sublist. For our purposes, we do not have complex literals other than negated literals. Nor do we address inference with respect to literals. We have utility functions which we can use to further specify the list of literals by eliminating or requiring certain \textit{collocations} of literals; for example, for whatever reason, every set of literals we wish to consider may have to contain both \texttt{prop1} and \texttt{neg-prop4}. We can call such collocational restrictions \textit{constraints} on the properties of \texttt{SOAs}; they can be used to define \textit{admissible models}. Such filters can be understood to represent \textit{abstract properties} which hold of all states-of-affairs; they \textit{constrain} the logical space of models under consideration and used for processing. Literals appear in a lattice-theoretic structure, where the literals are the atoms of the semilattice and the sets of the lattice are formed by \textit{meet} and \textit{join}. The minimal element of such a lattice is the \textit{empty list}; the maximal element is a \textit{contradictory list} (containing all the contradictions of literals); and each maximal list of consistent propositions corresponds to a possible world. The constraints imply that the lattice is not \textit{complete} since there may be elements of the powerlist of the literals that are not an element of the constrained literals. However, for our purposes, this is not relevant for the definition of actions, and we need not consider this further.

States-of-affairs, which are of type \texttt{SOA}, are records comprised of a field for a list of literals which are consistent and fields for indices for world and time. We refer to the list of literals as the \textit{properties} of the \texttt{SOA}. We use non-maximal lists of literals, so a particular \texttt{SOA} is schematic of the class of all those \texttt{SOAs} that have maximal lists of properties and in which the particular non-maximal list is a sublist; it leaves the other literals \textit{underspecified}. This allows us to focus attention on those relevant parts which are affected by an action.
While we have included the *world* index in the implementation, we do not make use of it for the purposes of this presentation. Moreover, while we have a temporal index, we only use this as a “clock” to record change induced by the execution of an action. We do not implement functions to use the world index as in Modal Logic, nor the time index as in Temporal Logic.

An example *SOA* is:

**Data 3**  
(properties = \{prop1, prop7, prop5, neg-prop3\},  
time = 2, world = 4)

Lists of expressions of type *SOA* are of type $DBSoas$. These can be understood as alternative states-of-affairs.

### 5.5.2 A Lexicon of Basic and Agentive Actions

We provide a lexicon of basic and agentive actions, sketching the main points. These actions are not constructed with action combinators (*sequence*, *choice*, *simultaneous*, or *negation*). Basic actions are labeled, directed arcs; that is, they are functions from lists of consistent literals (preconditions) to lists of consistent literals (postconditions). As such they can be viewed as *transition constraints*.

Our basic actions and the associated execution function are similar to *STRIPS* ([73]): world models, which represent information about states, are represented as lists of (consistent) literals; actions are executed with respect to a world model; actions are associated with preconditions and postconditions which determine (in combination with the execution function) what literals are added or removed from a world model after execution of the action. Unlike *STRIPS*, we do not have a theorem prover or a search strategy, as our focus is on the deontic operators.

The preconditions and postconditions of actions are represented as sets of literals; thus, it facilitates the *lexical semantic functions*. Agentive actions are basic actions with a field for the agent of the action. It is useful to make a distinction between actions and agentive actions for the purposes of searching the lexicon.

As with *SOAs*, we generate a set of actions, then filter them to create a lexicon. The lexicon of basic actions is a subset of the logical space of actions, which is the Cartesian Product of the lists of consistent literals which are used to define the *SOAs*. The point is that the actions must be such that they apply with respect to the *SOAs*. To form a subset
of actions from the logical space of actions, the actions are constrained by filtering the preconditions and postconditions of actions with collocational restrictions; for example, while there may be a consistent list of properties of SOAs which contains literals prop3 and neg-prop5, we may impose the constraint that there is no action with both these literals in the precondition (or alternatively, the postcondition). Moreover, we can constrain the actions in terms of transitions, for instance, filtering out all actions in which the literal prop5 appears in the precondition and neg-prop7 appears in the postcondition. From the remaining actions, we may create a lexicon, which is some selection of actions that represent our hypothetical domain. The importance of such constraints and working with a lexicon is that the resultant space of actions can have gaps, which are particularly important for the definition of antonyms and then for the definition of obligations. In addition, the lexicon has some structure, as actions are related to one another with respect to subsumption on the sets of preconditions and postconditions.

We discuss further issues relating to the structure of actions and deontic specification in section (5.5.3) on the lexical semantics. Here we focus on the types of actions as well as the functions which allow the execution of an action drawn from this hypothetical lexicon.

5.5.2.1 Types for Basic and Agentive Actions

An action is a record of type Action, which has fields for a label of type String, preconditions xcond of type PropList, and postconditions ycond of type PropList. An action is used to express transitions from SOAs where the preconditions hold to SOAs where the postconditions hold. As with the properties of the SOAs relative to maximal lists of literals, we take the preconditions and postconditions to be schematic of maximal lists of literals. Thus, an action represents a transition from any maximal list of literals where the preconditions are a sublist to any maximal list of literals where the postconditions are a sublist. An action with an agent is of type AgentiveAction, which is a record with fields for an action and an Agent of type String. A list of agentive actions is of type DBAgentiveAction.
An example of an agentive action follows: we have distinguished the agent from the action, where the action is as above.

Data 4  
\[
\begin{align*}
\text{action} &= (\text{label} = \text{Action6}, \\
\text{xcond} &= [\text{prop1, prop7, prop5}], \\
\text{ycond} &= [\text{prop3, neg-prop4, neg-prop6}], \\
\text{agent} &= \text{Jill}
\end{align*}
\]

This represents an abstract agentive action, which contrasts with agentive actions found in natural language, such as Jill leaves. We work exclusively with abstract agentive actions since we can explicitly work with the properties which exhaustively define them. It is harder to do so with natural language expressions since it is not clear that we can either explicitly or exhaustively define them in terms of component properties. Nonetheless, we can refer to the natural language examples where useful.

For our purposes here, we assume a model in which any distinction in the definition of an action distinguishes one action from another. For instance, we suppose that two actions otherwise the same but differing only with respect to the label are distinct. In another model, one could make other assumptions, but this is not a relevant issue for our purposes. We also do not discuss issues related to causality, which is not central to our concerns.

Furthermore, there are additional issues related to the lexical semantics of actions for examples such as Bill kicked the door and Bill kicked the door open ([119] and [120]) that concern a fuller theory of actions than we need to provide for our purposes. However, as we make clear in section 5.5.3, our approach provides a structure for actions. It is straightforward to set-theoretically define such relationships in a natural way. The relationship between Bill kicked the door and Bill kicked the door open is that the latter is more specific than the former – a hypernym in lexical semantics terms. One might want to provide constraints on the ways that particular actions can be structurally defined, for
example to account for why *Bill kicked the door red is ill-formed. We also assume that it is well-formed for there to be actions which induce change: before Bill kicks the door open, the door is not open. In this case, the precondition of the action contains a literal which represents that the door is not open and the postcondition contains a literal which represents that the door is open.

Because of the ease of expressing the relationships among actions this way, we have chosen not to represent actions as conditions of the form $P \rightarrow [\alpha](Q)$, where $P$ is the precondition and $Q$ the postcondition. It is also important to emphasise that, while our tool makes reference to natural language lexical semantic intuitions, it is not intended to provide a full analysis or implementation of the lexicon of natural language, though we believe it does provide a framework in which issues could be formalised, implemented, and addressed. However, additional such issues will be left for future research, for our main concern is the representation of the deontic notions with respect to actions.

5.5.2.2 Executing an Action Relative to a SOA

We execute an agentive action relative to a SOA with an execution function. For clarity and simplicity of exposition, we introduce here a version of the execution function which does not take into account deontic specifications on actions. In section 5.5.6, we modify the execution function to take into account not only deontic specifications, but also how contract states are modified as a consequence of violating or fulfilling a deontic specification. The function $doAgentiveAction$ in Code 6 takes expressions of type SOA and AgentiveAction and outputs an expression of type SOA.

**Code 6**

\[
\text{type doAgentiveAction :: SOA} \rightarrow \text{AgentiveAction} \rightarrow \text{SOA}
\]

However, to define the function, we first need to define what properties it satisfies. In particular, relative to the postcondition, the function must satisfy the following properties, which are stated informally. Our inertia condition addresses basic issues of the Frame Condition (\cite{126}).

**Definition 38**

- **Consistency:** no action can induce an inconsistent SOA, though an action can change the valence of a literal from positive to negative or vice versa.

- **Non-redundancy:** no action can introduce redundant literals in the resultant SOA.
• **Compatibility:** no action can induce a SOA which is not in keeping with the constraints on the properties of SOAs.

• **Inertia:** whatever properties of the input SOA which are otherwise unaffected by the action are output in the resulting SOA.

• **Action Postconditions Introduction:** the literals of the postcondition of the action are introduced into the resulting SOA.

Note that we do not here present issues related to Compatibility, though this is built into the implementation. The following is a helper function for the doAgentiveAction function and meets these requirements. It takes a SOA and an Agentive Action to produce a list of properties. The expression \( \setminus \) is a function which, given two lists, produces the difference between them. The expression ++ is a function which, given two lists, produces the concatenation of them. We also see here that, given an input Agentive Action, we first find the action and then the postconditions:

\[
\text{Code 7} \quad \text{diffSOAActionPostcond} :: \text{SOA} \rightarrow \text{AgentiveAction} \rightarrow \text{PropList}
\]
\[
diffSOAActionPostcond \text{ inSOA inAgentiveAction } =
((\#\text{properties inSOA}) \setminus ((\#\text{ycond (action inAgentiveAction)}) ++
\text{negateOrDoubleNegationElimList}(\#\text{ycond (action inAgentiveAction)}))) ++
(\#\text{ycond (action inAgentiveAction)})
\]

For example we have the following:

\[
\text{Data 5} \quad \text{Input} > \text{diffSOAActionPostcond (properties=\[prop2,prop3\], time=1, world=2) (action=(label=Action182, xcond=\[prop2,prop3\], ycond=\[prop1,prop2,neg-prop3\], agent=Hillary)}
\]
\[
\text{Output} > \text{[prop1,prop2,neg-prop3]}
\]

There are two aspects of inertia: that with respect to actions and SOAs and that with respect to the other aspects of the context that hold as a consequence of the execution of the action relative to the contract state and rule function. Here we discuss just the first form. The other we discuss only after we present actions executed relative to deontic specifications and rules.

In the execution function, an action can be executed so long as the preconditions of the action are a subset of the properties of the SOA with respect to which the action
is to be executed. Following execution of the action, the properties of the SOA must fulfill $\text{diffSOAActionPostcond}$, and the time index of the resultant SOA is incrementally updated. In Code 8, we have the function which essentially takes an AgentiveAction and a SOA as input and outputs a SOA. The $[\text{PropList}]$, which represents a list of lists of literals that we have specified as being incompatible, is needed by the helper function $\text{consistentContextInconIncomPred}$ for Compatibility. We have a guard in this function: if the preconditions of the AgentiveAction are a subset of the set of properties of the SOA and (represented by &&) the result of executing the action does not result in an inconsistent list of literals, then we execute the action with respect to the SOA. Doing so means that the properties of the output SOA are set to the result of the action applied to $\text{diffSOAActionPostcond}$, plus an indication that the action was executed, and the input time is incremented. If the guard fails because the action cannot be executed or execution of the action would give rise to inconsistency, then the output SOA is the same as the input SOA.

```haskell
Code 8  doAgentiveAction :: AgentiveAction → [PropList] → SOA → SOA
doAgentiveAction inAgentiveAction inComp inSOA
  | (subSet (Set (#xcond (#action (inAgentiveAction))))
  (Set (#properties inSOA))) &&
  ((consistentContextInconIncomPred inComp)
  (diffSOAActionPostcond inSOA inAgentiveAction)) =
  (properties = ((diffSOAActionPostcond inSOA
  inAgentiveAction) ++
  [(#label (#action (inAgentiveAction)))++-Done]),
  time = (#time inSOA)+1), world = (#world inSOA))
  | otherwise = inSOA
```

Note that we have a highly restricted notion of “an agentive action can be executed”. Here it means that the preconditions of the action are a subset of properties of the input SOA and that the result of executing the action would not create an inconsistent set of properties. Any other notion of can be executed is not relevant for our purposes. In particular, no comment is made with respect to the abilities of the agent or any other factors which may disable the execution of the action.

Clearly, any action which meets the requisite restrictions can be executed. We have further functions which find all such actions, then allow a random choice among them as to which one gets selected for execution. Other meaningful ways to pick an action could, in principle
be designed. These functions are interesting and useful for modeling runs of executions of actions, but this is not the focus of our presentation here.

In Data 6, we have an example.

**Data 6**

\[\text{input} \gg \text{doAgentiveAction} \]
\[
(\text{properties} = \{\text{prop1, neg-prop3, prop5, prop7}\}, \\
\text{time} = 2, \text{world} = 4) \\
(\text{action} = (\text{label} = \text{Action6}, \\
\text{xcond} = \{\text{prop1, prop5, prop7}\}, \\
\text{ycond} = \{\text{prop3, neg-prop4, neg-prop6}\}), \\
\text{agent} = \text{Jill}) \\
\text{output} \gg (\text{properties} = \{\text{prop1, prop3, neg-prop4, prop5, neg-prop6, prop7}\}, \\
\text{time} = 3, \text{world} = 4) \]

### 5.5.3 Lexical Space and Lexical Semantic Functions

In this section, we discuss the relations among actions, in general, and then lexical semantic functions that we want in order to define deontic specification of actions. With the lexical semantic functions applied to actions, relative to a lexicon, we can functionally find actions in the lexicon which stand in the specified relationships. These functions are central to our definition of the deontic operators for the following reasons.

#### 5.5.3.1 Systematic Lexical Relationships

For the definition of obligation, we want to determine which specific alternatives of a given action induce fulfillment or violation. Other alternative actions which could be executed in a given context need not be relevant for our analysis. For instance, if Bill is obligated to move the left toggle up, and we have only the actions of moving the left toggle up or down, then moving the toggle up fulfills the obligation, while moving it down violates it. Moreover, unless specifically excepted, alternative ways of moving the toggle up also fulfill the obligation and alternative ways of moving the toggle down also violate the obligation; actions which are subsumed by a deontically specified action may be taken to fulfill the obligation, and similarly for violation. However, it is not necessarily true that the relationship holds in the other direction; that is, if it is obligatory to move the toggle up slowly, then other ways of moving the toggle up may not fulfill this obligation.
manner adverb could be represented as an additional literal in the definition of the action. Nor, by the same token, can the action of moving the toggle up without specifying the manner fulfill the obligation to move it up slowly since this appears to introduce alternative ways to execute the action, such as moving the toggle up quickly, which do not fulfill the obligation.

We also wish to account for the symmetry of obligations. Suppose two possible worlds with the same time, properties, and available actions. The actions which fulfill or violate Bill is obligated to move the left toggle up ought to be the ones which violate and fulfill (respectively) Bill is obligated to move the left toggle down. Furthermore, the actions which fulfill or violate a particular obligation must both be executable in the same SOA. This means that the alternative actions required by a particular deontic specification on an action have the same precondition properties. Another point is that, while in principle one could define a lexicon of actions such that every action has an antonym and the obligation operator can apply to any action, in natural language, neither are so.

In general, we want to allow that there are actions which cannot be deontically specified or happen to be deontically unspecified. We want these observations to hold for complex actions or novel actions as much as for basic actions, which necessitates rules which can calculate the appropriate actions relative to an input action in the given lexicon.

5.5.3.2 Unsystematic Lexical Relationships

Finally, in addition to these systematic relationships between deontic specifications and actions, there may be conventional or idiosyncratic relationships. For example, suppose Bill is obligated to move the left toggle up, and there is an additional action of turning a light on with a foot switch; it could (arbitrarily) be the case that where Bill does turn the light on is included among the actions which are taken as a violation of Bill’s obligation. However, we focus on the systematic cases. While natural language provides the intuitions behind the design of the lexical semantic functions, we implement them with respect to our abstract actions. We do not provide a general background discussion of natural language lexical semantics ([45], [72], [150], [64]), but only the relevant notions.

5.5.3.3 A Graphic Representation

We represent these issues graphically in Figure (5.3), where we have the items below. Some of the notions, such as coordinate actions, hyponyms, and synonyms, are discussed later.
• **DOM** – the total logical space of actions generated in a particular model (i.e. the actions generated given constraints on consistent literals and transitions). It is the domain of actions.

• **α** – a deontically specified action, as in $\text{Obligated}(\alpha)$. $\alpha$ is an action, $\alpha \in \text{DOM}$. Performance of this action marks the context which arises subsequently with fulfillment of the obligation.

• **Ω** – the set of all actions which “count as” actions which fulfill the deontic specification on $\alpha$. These include coordinate actions, synonyms, and hyponyms with respect to $\alpha$. $\alpha \in \Omega \land \Omega \subset \text{DOM}$.

• **Δ** – the set of all possible actions which are in opposition to the deontically specified action $\alpha$; these are given by the function $f$ from $\alpha$ to $\Delta$. Every action in this set has the same preconditions and postconditions as $\alpha$ except that the actions in $\Delta$ have the negation of at least one of the literals found in the postcondition of $\alpha$. $\Delta \subset \text{DOM}$.

• **β** – a set of actions which are lexically realized in the model and are a subset of $\Delta$, $\beta \subset \Delta$.

• **ι** – the set of actions which are conventionally specified to be in opposition to the deontically specified action $\alpha$. Performance of an action in $\iota$ leads to violation. These are given by the function $g$ from $\alpha$ to $\iota$. $\iota \subset \text{DOM}$.
• \( f \) – a function from \( \alpha \) to \( \Delta \), given by the Cartesian Product of the precondition of \( \alpha \) with \text{altContextsGen} on the postcondition of \( \alpha \) and filtered with respect to the action restrictions of the model. In effect, the function gives us antonyms.

• \( g \) – a function from \( \alpha \) to \( \iota \), which are socially or conventionally defined actions the performance of which count as a violation of the given obligation. These are arbitrary, and there is no way to calculate them.

• The remainder of the actions, \( \gamma \), are the actions which are deontically unspecified with respect to \( \alpha \); that is, while execution of \( \alpha \) leads to fulfillment and execution of actions in \( \beta \) or \( \iota \) lead to violation, execution of actions in \( \gamma \) are unmarked for either violation or fulfillment. \( \gamma = \{ x : x \in \text{DOM} \land x \notin (\Omega \cup \Delta \cup \iota) \} \).

We can say that the set \( \Omega \) is the set of deontically specified actions, the set comprised of \( \Delta \cup \iota \) is the set of alternative actions with respect to the deontically specified actions, and \( \gamma \) is the set of deontically underspecified actions. Note that the difference between actions in \( \iota \) and those in \( \Delta \) is that those in \( \Delta \) are in a functionally determined with respect to antonymy relationship with respect to \( \alpha \), while those in \( \iota \) are not.

### 5.5.3.4 Lexical Semantic Functions

Given a lexicon of actions, we want lexical semantic functions to provide those actions which are abstractly described above. These functions are based on the following definitions:

**Definition 39** HyponymPost: An action \( \alpha \) is a hyponymPost of \( \beta \) if and only if the precondition of \( \beta \) is the same as the precondition of \( \alpha \), and the postcondition of \( \beta \) is a proper subset of the postcondition of \( \alpha \), and the number of elements in the postcondition of \( \beta \) is one less than that of \( \alpha \).

**Definition 40** Antonym: An action \( \alpha \) is an antonym of \( \beta \) if and only if the precondition of \( \beta \) is equal to the precondition of \( \alpha \), and the postconditions of \( \alpha \) and \( \beta \) differ only with respect to one literal which are contraries (e.g. prop1 and neg-prop1).

Note that alternative definitions of lexical semantic functions are possible, as for instance, restrictions to differences of one literal in the postconditions; the definitions suit the lexical semantic intuitions and are minimally plausible. Furthermore, as justified in section 2.9.4,
we want the function for antonyms to be *symmetrical*, such that, if the antonym for action α is β, then the antonym for β is α. The function for finding hyponyms with respect to the postcondition returns a set of actions. In addition, we have a function for *hypernyms* to identify more specifically defined actions in the lexicon.

### 5.5.3.5 Implementation

We have a function `findOpposites`, which is a function from an action to those actions which are antonyms of that action. It uses `findHyponymsPost`, which is an auxiliary function from an action to those actions which are hyponyms of that action relative to the postconditions (i.e. Definition 39). While there are, in principle several ways to provide the lexical semantic relationship or functions, we have modeled our approach on the lexical semantic relationships discussed in the literature on natural language for examples of the opposition between *leave* and *remain*.

Let us suppose a partial lexical semantic function `findOpposites`, which is a function from an `Action` to a list of `Actions`. It is partial in that not every action need have an antonym in the lexicon. For processing, it takes a lexicon and some constraints; the functions depend on several helper functions, which we will not present. For example, suppose `findOpposites` applied to the action labeled *Action6* yields the list which contains *Action7* and vice versa. The function outputs actions which are the same as the input action but for the negation of one of the postcondition literals. As an illustration, we have the following:

**Data 7**

| input > findOpposites (label = Action6, xcond = [prop1, prop7, prop5], ycond = [prop3, neg-prop4, neg-prop6]) | output > (label = Action7, xcond = [prop1, prop7, prop5], ycond = [prop3, neg-prop4, prop6]) |

Three things are important about the function `findOpposites` for our purposes. First, we can calculate specific alternative actions which give rise to violations. As discussed earlier, it is unintuitive that just any action other than the obligated action should give rise to violation. Second, as a calculation, we can find an opposite for any action *where the lexical structure allows one*. For the purposes of deontic specification, it need not be the case that every action has an antonym (although one could define a function and lexical space to allow this). Crucially, this holds for atomic as well as complex actions. And
finally, the function `findOpposites` is defined so as to provide reciprocal actions; that is, the opposite of Action6 is Action7 and vice versa. Thus, the function closely models the natural language case discussed above.

The function `findHypernymsPost` is a function from an action to a list of actions, where the output actions, if they exist in the lexicon, have the same list of preconditions and lists of postconditions with one more literal in them.

```
Data 8 input> findHypernymsPost (label = Action6, 
   xcond = [prop1, prop7, prop5], 
   ycond = [prop3, neg-prop4, neg-prop6])
output> (label = Action9, xcond = [prop1, prop7, prop5], 
   ycond = [prop3, neg-prop4, neg-prop6, prop2])
```

Both functions return the set of actions which meet the conditions. At this point, we are prepared to consider our deontic specification.

### 5.5.4 Deontic Specifications

In this section, we provide the function for specifying the `ContractFlags` (i.e. a valued action specification), `ContractFlagStates` (i.e. a valued specification list), and then the obligation operator on actions. At this point, we focus on deontic specification on basic actions. For brevity, discussion of permission and prohibition are mentioned, but not implemented in full. We first discuss deontic specification on simple actions to create VSLs, then we present manipulations on VSLs.

#### 5.5.4.1 The Creation of Value Specification Lists

A deontic specification on a simple action is a record that indicates which action is deontically specified, which actions can fulfil the deontic specification, which actions violate the deontic specification, and the fulfilment and violation flags which follow the execution of the given actions in each case.

To define the deontic specifications, we provide a type `ContractFlag`. This type is a record having fields for: the action which is executed (indicated by the label), the deontic specification on the action (i.e. obligated, permitted, or prohibited), the action which is deontically specified (indicated by the label and which can be distinct from the action
that is executed), whether execution of the action flags for violation or fulfillment, and the agent which executes the action. Lists of contract flags are of type \textit{ContractFlagState}.

\textbf{Code 9} 

\begin{verbatim}
type ContractFlag = Rec (actionDone::String, deonticSpec::String, onSpec:: String, valueFlag::String, agent::Agent)
type ContractFlagState = [ContractFlag]
\end{verbatim}

The violation and fulfillment flags, which are \textit{String} types that are values of \textit{valueFlag}, are key in reasoning what follows from a particular flag. In other words, that an agent has violated an obligation on an action may imply that the agent incurs an additional obligation.

A deontic operator is a function from an \textit{AgentiveAction} to a list of results in a \textit{ContractFlagState}. A list of actions \textit{DBAction} and literals \textit{PropList} are also input in order to search the lexicon and find an antonym. Here we provide the function for \textit{obligation}. The functions for \textit{prohibition} and \textit{permission} are similar. For \textit{prohibition}, we do not search for antonyms, but only for hypernyms, which are flagged with a violation marker with respect to the deontic specification. \textit{Permission} is similar, but the value flag is \textit{NULLVF}, which signals that permissions are neither violated nor fulfilled.

\textbf{Code 10} 

\begin{verbatim}
type obligatedCompFlag :: AgentiveAction \rightarrow DBAction \rightarrow [PropList] \rightarrow ContractFlagState
\end{verbatim}

In \textbf{Code 11}, a \textit{ContractFlagState} is calculated relative to an input agentive action \textit{inAgentiveAction} (along with a lexicon and compatibility constraints).

We discuss the code relative to the line numbers in \textbf{Code 11}. Lines 1-2 constitute a \textit{guard} on the function: if the action from the input agentive action has an opposite (i.e. is a non-empty list), only then do we return a non-empty \textit{ContractFlagState} list. Thus, we can block vacuous obligations (if we choose to do so). Otherwise, we return the empty list (line 14). This reflects the conceptual point that there can only be obligations on an action where the obligation can be violated. Thus, where we return a non-empty list, there is some action in opposition to the input action. All the actions in the list are associated with the same input agent, input action, and deontic operator. We find the input action along with the hyponymPosts and antonyms of the action, which are marked accordingly with fulfillment or violation markers. In lines 3-7, we create a list of type \textit{ContractState} which represents the \textit{fulfillment} of the obligation on the action. In lines 8-13, we find
all those actions which are hyponymPosts of the input action and similarly indicate that they are associated with fulfillment of the obligation. In lines 14-19, we find the opposite to the input action and use it to create a list of type \texttt{ContractState} which represents the violation of the action. We use `++` to conjoin these to lists to produce a list of type \texttt{ContractFlagState}.

\begin{verbatim}
\textbf{Code 11} \quad \texttt{obligatedCompFlag inAgentiveAction inDBAction inComp}
\begin{verbatim}
1 | ((findOpposites (#action inAgentiveAction)
2   inDBAction inComp) \neq []) =
3   [([actionDone=(#label (#action inAgentiveAction)),
4      deonticSpec=Obligated,
5      onSpec=(#label (#action inAgentiveAction)),
6      valueFlag=Fulfilled,
7      agent=(#agent inAgentiveAction))] ++
8   [([actionDone=(#label x), deonticSpec=Obligated,
9      onSpec=(#label (#action inAgentiveAction)),
10     valueFlag=Fulfilled,
11    agent=(#agent inAgentiveAction))]
12   | x ← (findHyponymsPost
13     (#action inAgentiveAction) inDBAction)] ++
14   [([actionDone=(#label x), deonticSpec=Obligated,
15      onSpec=(#label (#action inAgentiveAction)),
16     valueFlag=Violated,
17    agent=(#agent inAgentiveAction))]
18   | x ← (findHyponymsPost
19     (#action inAgentiveAction) inDBAction [])]]
20 | otherwise = []
\end{verbatim}
\end{verbatim}

To illustrate, let us assume that we apply \texttt{obligatedCompFlag} to an agentive action labelled \texttt{Action6} with agent \texttt{Jill}.

\texttt{Chapter 5} \hspace{1cm} 5.5 Overview of the Implementation

\texttt{}
Data 9  \[
\text{input} \succ \text{ obligatedCompFlag} (\text{action} = (\text{label} = \text{Action6},
\ xcond = [\text{prop1}, \text{prop7}, \text{prop5}], \ ycond = [\text{prop3}, \text{neg-prop4},
\ \text{neg-prop6}], \ agent = \text{Jill}) \text{ testLexActions02 }] \\
\text{output} \succ [(\text{actionDone} = \text{Action6}, \ agent = \text{Jill},
\ \text{deonticSpec} = \text{Obligated}, \ onSpec = \text{Action6}, \ valueFlag = \text{Fulfilled}),
\ (\text{actionDone} = \text{Action7}, \ agent = \text{Jill}, \ deonticSpec = \text{Obligated},
\ onSpec = \text{Action6}, \ valueFlag = \text{Violated}), (\text{actionDone} = \text{Action8},
\ agent = \text{Jill}, \ deonticSpec = \text{Obligated}, \ onSpec = \text{Action6},
\ valueFlag = \text{Violated})]
\]

This is of type \text{ContractFlagState}. It associates Jill and \text{Action6} as well as the hypernym \text{Post Action7} with a fulfillment of Jill’s obligation on \text{Action6}. On the other hand, it associates Jill and \text{Action8} with a violation on Jill’s obligation on \text{Action6}.

5.5.4.2 Manipulations on \text{ContractFlagStates}

At this point, we have lists which correspond to our reductions of the deontic operators. However, we have only considered those lists which represent one deontic specification with respect to one action and one agent. Clearly, we would like more complex lists which can be taken to represent deontic specifications on a range of actions and agents; such a complex list could be understood, with caveats from earlier, as a \text{contract state}, for different agents have deontic specifications which bear on the actions they are to execute. Furthermore, looking in advance, we will want to allow the lists of deontic specifications to \text{change over time}, and in particular, in \text{reaction} to what actions have been executed and their deontic specifications. We use such changes to represent the CTD structure.

We have two ways to change a \text{ContractFlagState} – adding or removing other \text{ContractFlagStates}. Since every \text{ContractFlagState} corresponds to a deontic specification on an agentive action, such additions or subtractions amount to the addition or subtraction of a deontic specification. The function \text{addToContractFlagState} simply concatenates two lists, each a \text{ContractFlagState}. The function \text{subFromContractFlagState} is set difference between two sets of type \text{ContractFlagState}.
5.5 Overview of the Implementation

**Code 12**

```haskell
addToContractFlagState :: ContractFlagState -> ContractFlagState -> ContractFlagState
addToContractFlagState inContractFlagStateA inContractFlagStateB = inContractFlagStateA ++ inContractFlagStateB

subFromContractFlagState :: ContractFlagState -> ContractFlagState -> ContractFlagState
subFromContractFlagState inContractFlagStateA inContractFlagStateB = [ x | x <- inContractFlagStateB, not (elem x inContractFlagStateA)]
```

For example, the following represents Jill’s obligation with respect to Action6 added to Bill’s prohibition with respect to Action9.

**Data 10**

```text
[(actionDone = Action6, agent = Jill,
  deonticSpec = Obligated, onSpec = Action6,
  valueFlag = Fulfilled),
  (actionDone = Action7, agent = Jill,
  deonticSpec = Obligated, onSpec = Action6,
  valueFlag = Violated),
  (actionDone = Action9, agent = Bill,
  deonticSpec = Prohibited, onSpec = Action9,
  valueFlag = Violated)]
```

Manipulations of `ContractFlagState` expressions are crucial for modeling contract change, which is key to the analysis and implementation of Contrary-to-Duty Obligations.

### 5.5.4.3 Additional Functionality

A range of other issues are relevant to the creation and manipulation of `ContractFlagStates`. We briefly mention the additional functionalities of the program. However, while important, they are outside the main line of development of the chapter, so we only provide an outline discussion.

We can have functions which can constrain the `ContractFlagState`. As we are providing a tool, we allow these constraints to be defined by the system designer to suit their idea of how the deontic concepts should be reasoned with. For example, suppose no `ContractFlagState` should allow violation and fulfillment to be defined with respect to the same agent, deontic specification, and action. We write a predicate which returns `True` where this constraint is not violated.
5.5 Overview of the Implementation

Code 13  \[
\text{consistentContractFlagState} :: \text{ContractFlagState} \rightarrow \text{ContractFlagState} \\
\rightarrow \text{Bool}
\]
\[
\text{consistentContractFlagState} \text{ inContractFlagState} = \\
\lambda x \rightarrow \text{every inContractFlagState} (\lambda y \rightarrow \\
(\text{not} \\
((\text{elem} (\text{actionDone} = (#\text{actionDone} y), \text{agent} = (#\text{agent} y), \text{deonticSpec} = (#\text{deonticSpec} y), \text{onSpec} = (#\text{onSpec} y), \text{valueFlag} = "Violated") \\
\text{inContractFlagState}) \&\& \\
(\text{elem} (\text{actionDone} = (#\text{actionDone} y), \text{agent} = (#\text{agent} y), \text{deonticSpec} = (#\text{deonticSpec} y), \text{onSpec} = (#\text{onSpec} y), \text{valueFlag} = "Fulfilled") \\
\text{inContractFlagState})))
\]

\[
\text{consistentContractFlagStatePred} :: \text{ContractFlagState} \rightarrow \text{Bool}
\]
\[
\text{consistentContractFlagStatePred} \text{ inContractFlagState} = \\
\text{consistentContractFlagState inContractFlagState}
\]

If values are given for the deontic specifier, then we could have a function similar to Code 13 to block, for example, an obligation and a prohibition on the same action by the same agent. In this way, the system designer can provide tests for consistent contracts as they define it.

5.5.5 Histories

To model reasoning for CTDs, we enrich our SOAs to include expressions of type \text{ContractFlagState} as well as histories of type \text{History}. Expressions of type \text{History} are lists of records of type \text{HistoryFlag} which have labels and fields for what was done, when, by whom, and whether this is associated with a value flag of fulfillment or violation relative to a deontic specification. Expressions of type \text{History} and \text{ContractFlagState} are similar in many respects; the key difference is in how they are processed. In particular, while expressions of type \text{ContractFlagState} can be modified, by adding or removing a record of
type ContractFlag, we cannot remove something of type HistoryFlag from a History. Thus, we can say that ContractFlagStates are non-monotonic, while Histories are monotonic.

**Code 14**

```haskell
type HistoryFlag = Rec (actionDone::String,
                        deonticSpec::String, onSpec:: String,
                        valueFlag::String, agent::Agent,
                        world::World, time::Time)

type History = [HistoryFlag]
```

As we want our execution function to modify the SOA relative to a ContractFlagState and a History, we revise our notion of SOA.

**Code 15**

```haskell
type SOAHistorical = Rec (properties::PropList,
                          actionDone :: String, history :: History,
                          contractFlagState :: ContractFlagState,
                          world :: World, time :: Time)
```

Actions are executed with `doAgentiveActionSOAHistContract`, which is a function from SOAHistorical to SOAHistorical. We have adapted our previous execution function to take into account the different type. Supposing that the action is executable in the SOAHistorical, then the crucial issue is to test the action relative to the ContractFlagState. If the action is among those in the ContractFlagState, then we record in the History that action which is executed, along with the deontic information. Otherwise, we simply record that the action has been executed. This is first expressed informally, then the function is provided.

If

The input action is deontically specified in the contract state.

Then update the SOA

The Properties are the result of

`doAgentiveActionSOAHist`

The ContractFlagState is inherited from the input SOA

The History is inherited from the input SOA,
and we update it with the deontic specification associated with the agentive action executed and stamped along with the world and time when it was executed.

The Time is inherited.

The World is inherited.

Otherwise

The Properties are inherited.

The ContractFlagState is inherited.

The History is inherited from the input SOA, and we update it to indicate the agentive action executed and stamped along with the world and time when it was executed.

The Time is inherited.

The World is inherited.

We see that at every time, we have a `ContractFlagState`; we can easily define a function to return that `ContractFlagState` so as to report what is “currently” obligated, permitted, or prohibited.

The function which implements it is in Code 16. In lines (2-3), we test whether the input agentive action is in the `ContractFlagState` with a utility function `testAgentive-ActionInContractFlagState`. If it is not, then we follow as in lines (20-36): the properties of the `SOA` are those after execution of the action (20-22), the `ContractFlagState` is inherited (23-25), the `History` records the execution of the action without markers for deontic specification (26-30), but with a marker for the world and time of execution (31-32), and the world and time of the `SOA` are updated (33-36). Notice that we *presume empty deontic specification* in order to have a uniform record type, though this is not strictly necessary. If the action *is* deontically specified, then we follow as in lines (4-19). The key difference from (20-36) is just in updating the input history with the record of the action executed along with the information relative to the deontic flag that is associated with execution of the action. This is achieved in lines (11-15) by finding the record which corresponds to the deontic specification which is associated with the input agentive action and extending it with world and time markers, then concatenating it with the list of the input history.
**Code 16** \( \text{doAgentiveActionSOAHistContract :: AgentiveAction} \rightarrow \text{SOAhistorical} \rightarrow [\text{PropList}] \rightarrow \text{SOAhistorical} \)

1. \( \text{doAgentiveActionSOAHistContract} \) in AgentiveAction in SOAhist in PropList
2.  
   | (testAgentiveActionInContractFlagState in AgentiveAction
3.   | (#contractFlagState in SOAhist)) =
4.   | (properties = (#properties (doAgentiveActionSOAHist
5.   | in AgentiveAction in SOAhist in PropList)),
6.   | contractFlagState = (#contractFlagState
7.   | (doAgentiveActionSOAHist
8.   | in AgentiveAction in SOAhist in PropList)),
9.   | history = ((#history (doAgentiveActionSOAHist
10.  | in AgentiveAction in SOAhist in PropList)) ++
11.  | ((world = (#world in SOAhist),
12.  | time = ((#time in SOAhist)+1) |
13.  | (!!!0)(giveAgentiveActionInContractFlagState
14.  | in AgentiveAction
15.  | (#contractFlagState in SOAhist)))),
16.  | time = (#time (doAgentiveActionSOAHist
17.  | in AgentiveAction in SOAhist in PropList)),
18.  | world = (#world (doAgentiveActionSOAHist
19.  | in AgentiveAction in SOAhist in PropList))
20.  | otherwise = (properties =
21.  | (#properties (doAgentiveActionSOAHist
22.  | in AgentiveAction in SOAhist in PropList)),
23.  | contractFlagState = (#contractFlagState
24.  | (doAgentiveActionSOAHist in AgentiveAction
25.  | in SOAhist in PropList)),
26.  | history = ((#history (doAgentiveActionSOAHist
27.  | in AgentiveAction in SOAhist in PropList)) ++
28.  | ([actionDone = (#label (#action in AgentiveAction)),
29.  | deonticSpec = NULLVOL, valueFlag = NULLVF,
30.  | onSpec= NULLAC, agent = (#agent in AgentiveAction),
31.  | world = (#world in SOAhist),
32.  | time = ((#time in SOAhist)+1))]),
33.  | time = (#time (doAgentiveActionSOAHist
34.  | in AgentiveAction in SOAhist in PropList)),
35.  | world = (#world (doAgentiveActionSOAHist
36.  | in AgentiveAction in SOAhist in PropList)))

225
We illustrate the above code. Suppose the following input `SOAHistorical` to `doAgentive-ActionSOAHist` before an action is applied. Notice that the history is empty, which means that there is no evidence that a previous action has been executed.

**Data 11**


```
(contractFlagState =
  [(actionDone = Action6, agent = Jill,
    deonticSpec = Obligated, onSpec = Action6,
    valueFlag = Fulfilled),
   (actionDone = Action7, agent = Jill,
    deonticSpec = Obligated, onSpec = Action6,
    valueFlag = Violated),
   (actionDone = Action9, agent = Bill,
    deonticSpec = Prohibited, onSpec = Action9,
    valueFlag = Violated)],
  history = [],
  properties = [prop1, prop7, prop5, neg-prop4,
    neg-prop6], time = 2, world = 7)
```

Suppose then that Jill does execute `Action7` with respect to this `SOAHistorical`. This means that we should indicate that Jill has violated her obligation. Thus, in the `History` of the subsequent `SOAHistorical`, we record that Jill executed `Action7`. We also record that this action violates Jill’s obligation to execute `Action6`, as well as the world and time stamp where the violation occurred. Note that the time of the violation (or fulfillment) is indicated in the `history`, not the deadline of the deontic specification. We also see that the time of the `SOAHistorical` is updated. The properties are updated as well. Had Jill executed `Action6` instead, then the subsequent `SOAHistorical` would have recorded that Jill had fulfilled her obligation to execute that action.

**Data 12**


```
(contractFlagState =
  [(actionDone = Action6, agent = Jill,
    deonticSpec = Obligated, onSpec = Action6,
    valueFlag = Fulfilled),
   (actionDone = Action7, agent = Jill,
    deonticSpec = Obligated, onSpec = Action6,
    valueFlag = Violated),
```

226
In the Data 11 and 12, we executed actions which either violated or fulfilled a deontic specification in the contract state. However, as we discussed in chapters 2 and 4 as well as in section 5.5.3, we have actions which, if executed, give rise to neither violation nor fulfillment of a given deontic specification. It is then possible that there are deontic specifications in a contract state, but an action is executed which is not then recorded for violation or fulfillment in the history. For example, we have the following, where Jill executes Action72 with respect to the previous soaHistorical, but no violation or fulfillment flags appear; this is because our execution function does not find this action among those in the contract state, so does not pass violation or fulfillment flags onto the history. The values NULLVOL, NULLAC, and NULLVF are fill-in values.

**Data 13**

\[
\text{(contractFlagState = }
\begin{align*}
\text{(actionDone = Action6, agent = Jill,} \\
\text{deonticSpec = Obligated, onSpec = Action6,} \\
\text{valueFlag = Fulfilled),} \\
\text{(actionDone = Action7, agent = Jill,} \\
\text{deonticSpec = Obligated, onSpec = Action6,} \\
\text{valueFlag = Violated),} \\
\text{(actionDone = Action9, agent = Bill,} \\
\text{deonticSpec = Prohibited, onSpec = Action9,} \\
\text{valueFlag = Violated)),} \\
\text{history = [(actionDone = Action72, agent = Jill,} \\
\text{deonticSpec = NULLVOL, onSpec = NULLAC,} \\
\text{time = 2, valueFlag = NULLVF, world = 7)],} \\
\text{properties = [prop1, prop7, prop5, prop3, neg-prop4,} \\
\text{prop6], time = 3, world = 7)}\end{align*}
\]

In this way, it is possible for some agentive actions to be deontically specified in the contract state, but for other agentive actions to be executable without giving rise to either
violation or fulfillment. In principle, it is possible that no agent ever violates or fulfills any
deohtic specification; however, as we have indicated earlier, issues relating to enforcement
of the deontic specifications or action selection are outside the scope of this thesis. One
strategy, as we point out below, is to introduce deadlines, but these are not necessary to
the system.

At this point, we have represented what we take to be the meaning of deontic specification
on an action; namely, that the deontic specification on an action associates markers for
fulfillment with that action and deontic operator as well as markers for violation for actions
which are considered as opposites. We can view these markers as secondary and non-
essential properties that are associated with the action. When the action is executed, these
secondary properties are recorded in the history. For us, the main purpose of recording
the deontic specifications which have been fulfilled or violated is to serve as triggers for
contract state update.

As we have discussed earlier, we distinguish between this meaning of deontic specification
on an action and what may follow from the flags. It is this latter aspect which we use in
our implementation of CTDs, which we discuss in the subsequent section.

5.5.6 Rule Functions and CTDs

In this subsection, we show how what holds in the History is used to trigger updates
of the ContractFlagState with respect to rule functions. We use such rule functions to
model the effect of the CTD structure. In contrast to the execution of actions, which not
only update the History but also the time of the SOAHistorical, updates of Contract-
FlagState do not themselves update the time. We use this to model the difference between
actions which induce temporal change of the SOAHistorical, i.e. where the time changes,
and those functions which induce atemporal change of the SOAHistorical, where the time
does not change. The latter, therefore, appear as semantic implications. In this section,
we also clarify the conception of contract, which is essentially comprised of the contract
state and the rule function. We start with a discussion of the rule function, then provide
some simple examples to illustrate how it works. We then discuss some alternatives.

As we see, the introduction of rules brings our implementation and analysis into the domain
of expert systems [78], which implement reasoning in a domain with production rules.
However, such rules are distinct from purely logical approaches, such as we discussed in
chapters 2-4. A range of issues arise concerning the consistency, ordering, and organisation
of the rules; the various solutions to such issues as found in the literature can, in principle,
be applied. However, as we have not cast this thesis in terms of expert systems, we focus on the presentation of the rules, leaving development in terms of a well-developed expert system for future work.

In the following, we have hard-coded the rule function into the execution function. An alternative would be to modularize the rules by abstracting the particulars of the rule function into a record so as to be able to manipulate it. However, the execution function, which updates components of the valued specified context, will still have to implement the rules, though this could then appear in a more general, but less immediately clear, form. Such a manipulation of a rule function could be understood as a legislative action since the consequences of fulfillment or violation of a law are manipulated.

5.5.6.1 Implementation

To implement the CTD structure, we allow contract state modification relative to actions which have been executed in the history. Recall from the discussion of the CTD structure that we want a secondary obligation to arise only where some other obligation has been violated (or fulfilled). In other words, if a particular violation of an obligation appears in the History, we want a secondary obligation to be introduced into (or subtracted from) the ContractFlagState of the SOAHistorical. For example, suppose Jill is obligated to move the left toggle up. If Jill violates this obligation (by moving the left toggle down), then she incurs a secondary obligation to move the right toggle right. On the other hand, if Jill fulfills her obligation, then she perhaps incurs a secondary permission such as moving the left toggle down. The secondary obligations or permissions only arise in cases where a primary obligation has been violated or fulfilled; as such, they are context dependent. We initially present this aspect of ContractFlagState update, then discuss additions.

We first have to examine whether a particular violation marker appears in the history. Second, we have to make that violation flag trigger ContractFlagState modification. For instance, suppose that it appear in the History that Jill has violated her obligation to do Action6 by doing Action7. As a consequence, we modify the current contract state by removing her previous obligation and introducing an obligation on Action11. In such an operation, only the ContractFlagState is modified. This gives the appearance of inference in a state-wise analysis, for there is no temporal updating.

In Code 17, we have an execution function doRDSMOD, which implements action execution for relativized deontic specifications; it is (essentially) a function from AgentiveActions.
and SOAHistorical to SOAHistorical. It incorporates modification of the Contract-FlagState. Let us first look at the triggers for update.

In this code sample, we have two expressions of type HistoryFlag: in line (2), historyFlag01 represents Jill’s having violated her obligation to execute Action6 by having executed Action7 instead at time 1; in line (22), historyFlag02 represents Jill’s having fulfilled her obligation to execute Action6 by having executed Action6 at time 1. In either case, we investigate whether the given trigger is an element of the current History. We can, then, change the contract state relative to whether Jill has violated or fulfilled her obligation to execute Action6. One can hard-code additional triggers by adding clauses of the similar forms. Note that this is part of the execution function, so some of the elements of the code reproduce earlier execution elements, such as in lines (5-7) and (14-21). Other times, agents, deontic operators, or actions which do not count as violations or fulfillment of the deontic specification will not trigger contract state update. Such highly specific triggers are useful for illustrating how the system works.

Now let us consider the consequences of the trigger. In lines (9-15), where we have an update of the output ContractFlagState by concatenating the input ContractFlagState with a new prohibition on Jill’s execution of Action3. In other words, where Jill has violated her obligation to execute Action6 at time 1, we update the contract state by adding this prohibition. In lines (29-39), as a consequence of Jill having fulfilled her obligation to execute Action6 at time 1, we have instead added one deontic specification and also removed another: in particular, Jill is permitted to execute Action3 and the obligation to execute Action6 is removed. Note that what is not explicitly changed by the action or the update of the ContractFlagState remains in the output SOAHistorical. Finally, in lines (46-47), if there are no relevant triggers for ContractFlagState update, then it is carried forward. In other words, as a consequence of violating or fulfilling the obligation, deontic specifications are introduced or removed from the contract state. This is the essential mechanism for contract state modification.

Following our discussion of the CTD structure in section 3.4, we need not update the ContractFlagState with an additional obligation, but rather have other deontic specifications. However, we could have, and this would directly fulfill one of our primary goals of showing sequences of obligations in that as one deontic specification is fulfilled or violated, another arises. As the output of the application of doRDSMOD on an action and a SOAHistorical is itself a SOAHistorical, we allow iterative application to the next execution of the action; that is, subsequent executions of actions are relative to this new context of deontic specifications. It is rather arbitrary how the rule function changes the contract state, which we see as an advantage to a system designed to experiment with
alternative definitions. Moreover, as we discuss in a subsequent subsection, within the
types and functions given to this point, what the triggers are and what follows from them
are entirely up to the designer of the system.

**Code 17**

```
doRDSMOD :: AgentiveAction -> SOAhistorical -> [PropList] ->
SOAhistorical
  doRDSMOD inAgentiveAction inSOAhist inPropList
  | (elem historyFlag01
     (#history (doAgentiveActionSOAHistContract
       inAgentiveAction inSOAhist inPropList))) =
     (properties =
      (#properties (doAgentiveActionSOAHistContract
                        inAgentiveAction inSOAhist inPropList)),
       contractFlagState =
       (addToContractFlagState
        (prohibitedCompFlag
         (giveAgAcSimple Jill Action3
          testLexActions04) testLexActions04 []))
     (#contractFlagState
      (doAgentiveActionSOAHistContract inAgentiveAction
        inSOAhist inPropList)),
     history = (#history (doAgentiveActionSOAHistContract
                          inAgentiveAction inSOAhist inPropList)),
     time = (#time (doAgentiveActionSOAHistContract
                     inAgentiveAction inSOAhist inPropList)),
     world = (#world (doAgentiveActionSOAHistContract
                      inAgentiveAction inSOAhist inPropList)))
  | (elem historyFlag02
     (#history (doAgentiveActionSOAHistContract
       inAgentiveAction inSOAhist inPropList))) =
     (properties =
      (#properties (doAgentiveActionSOAHistContract
                        inAgentiveAction inSOAhist inPropList)),
       contractFlagState =
       (addToContractFlagState (permittedCompFlag
                                  (giveAgAcSimple Jill
                                   Action3 testLexActions04)
                                   testLexActions04 []))
```

231
5.5 Overview of the Implementation

Let us give two examples, one where Jill has fulfilled her obligation on Action6 by executing Action6 and another where she has violated this obligation by executing Action7. We assume the execution function doRDSMOD is executed with respect to these actions and with respect to the following soaHistorical:

**Data 14**  \((\text{contractFlagState} = \)
\[
((\text{actionDone} = \text{Action6}, \text{agent} = \text{Jill}, \\
\quad \text{deonticSpec} = \text{Obligated}, \text{onSpec} = \text{Action6}, \\
\quad \text{valueFlag} = \text{Fulfilled}), \\
((\text{actionDone} = \text{Action7}, \text{agent} = \text{Jill}, \\
\quad \text{deonticSpec} = \text{Obligated}, \text{onSpec} = \text{Action6}, \\
\quad \text{valueFlag} = \text{Violated}), \\
((\text{actionDone} = \text{Action6}, \text{agent} = \text{Bill}, \\
\quad \text{deonticSpec} = \text{Prohibited}, \text{onSpec} = \text{Action6}, \\
\quad \text{valueFlag} = \text{Violated}), \\
\text{history} = [], \\
\text{properties} = [\text{prop1, prop7, prop5}], \\
\text{time} = 2, \text{world} = 7)\)
In the first example, where Jill executes Action6, which she is obligated to do, the resulting contract state is as follows, where Jill is permitted to execute Action3 and her prior obligation is removed. This follows from lines (29-39) of the execution function doRDSMOD. Bill’s prohibition is unaffected.

Data 15  \( (\text{contractFlagState} = \)
\[
[(\text{actionDone} = \text{Action3}, \text{agent} = \text{Jill}, \text{deonticSpec} = \text{Permitted}, \text{onSpec} = \text{Action3}, \text{valueFlag} = \text{NULL}),
(\text{actionDone} = \text{Action6}, \text{agent} = \text{Bill}, \text{deonticSpec} = \text{Prohibited}, \text{onSpec} = \text{Action6}, \text{valueFlag} = \text{Violated})],
\]
\[
\text{history} = [(\text{actionDone} = \text{Action6}, \text{agent} = \text{Jill}, \text{deonticSpec} = \text{obligated}, \text{onSpec} = \text{Action6}, \text{time} = 2, \text{valueFlag} = \text{Fulfilled}, \text{world} = 7)],
\]
\[
\text{properties} = \{\text{prop1}, \text{prop7}, \text{prop5}, \text{prop3}, \text{neg-prop4}, \text{neg-prop6}\},
\]
\[
\text{time} = 3, \text{world} = 7)
\]

Alternatively, where Jill executes Action7 relative to Data 14, which flags a violation of her obligation to execute Action6, we get the resulting contract state, where Jill accrues a prohibition (in addition to maintaining her prior obligation). This follows from lines (8-12) of the execution function doRDSMOD. Again, Bill’s prohibition is unchanged. It is important to emphasise that we take no stance on the introduction or elimination of deontic specifications, which is a matter up to the rule designer.

Data 16  \( (\text{contractFlagState} = \)
\[
[(\text{actionDone} = \text{Action6}, \text{agent} = \text{Jill}, \text{deonticSpec} = \text{Obligated}, \text{onSpec} = \text{Action6}, \text{valueFlag} = \text{Fulfilled}),
(\text{actionDone} = \text{Action7}, \text{agent} = \text{Jill}, \text{deonticSpec} = \text{Obligated}, \text{onSpec} = \text{Action6}, \text{valueFlag} = \text{Violated}),
(\text{actionDone} = \text{Action6}, \text{agent} = \text{Bill}, \text{deonticSpec} = \text{Prohibited}, \text{onSpec} = \text{Action6}, \text{valueFlag} = \text{Violated})],
\]
\[
\text{history} = [(\text{actionDone} = \text{Action7}, \text{agent} = \text{Jill}, \text{time} = 2, \text{valueFlag} = \text{Fulfilled}, \text{world} = 7)],
\]
\[
\text{properties} = \{\text{prop1}, \text{prop7}, \text{prop5}, \text{prop3}, \text{neg-prop4}, \text{neg-prop6}\},
\]
\[
\text{time} = 3, \text{world} = 7)
\]
5.5 Overview of the Implementation

In our view, the implementation captures the essence of the fundamental CTD problem. It models how the execution of an action relative to a ContractFlagState induces a modification of the ContractFlagState. As discussed earlier, we are not claiming to have implemented with this all issues or examples argued to be associated with CTDs; rather, we have claimed that the central issue is relativized introduction of deontic specification into a ContractFlagState and that additional issues have to be analyzed independently.

For us, a contract is comprised of the contract state, expressed in the ContractFlagState, along with the ways in which contract states can be modified according to the rule function, expressed in the execution function. Within this conception, we may be more specific. For example, we call the initial contract that contract which is the contract state as it is initialized and prior to the execution of any agentive actions along with the rule relation (which does not change). Alternatively, we have the local contract, which is the contract state subsequent to every execution of every agentive action along with the rule function; clearly, the initial contract is also a local contract, but not vice versa. The logical contract is that set of local contracts which are the result of every possible execution of an agentive action. In our previous example, Data 14 is our initial contract; Data 14, Data 15, and Data 16 are local contracts; and the union of the local contracts is the logical contract since no other local contracts are possible given an execution of an agentive action.

5.5.6.3 Additional Rules

In Code 17, we only used elements of the History to trigger contract state update. However, in principle, any boolean expression (i.e. any Haskell expression which has a truth value) can be used to trigger an update of any part of the soaHistorical. We restrict ourselves to relevant cases, although alternatives are possible.

Suppose we want rules which are triggered by a literal, by a particular deontic specification, by a time along with some violation or fulfillment flag, or by an agentive action. The first two represent alternative conditional obligations as discussed in chapter 2. Thus, while we have not required our system to represent and reason with them, neither have we ruled them out. In particular, in Examples (129a) and (129b), we have logical implications. A
rule in the form of Example (129a) could be used to represent *stative obligations* such as *It is obligatory that the yard is clean*. In Example (129c), we have the notion of *deadline* as in *Bill is obligated to return the book*, where the deadline is time 2. If Bill returns the book, so fulfilling his obligation to return the book, and the time is less than or equal to 2, then we remove the obligation from the contract state. Alternatively, if the time is greater than 2 and Bill has not yet fulfilled his obligation to return the book, then we introduce an additional obligation to the contract state as well as introduce a violation flag to the history relative to Bill’s obligation. Note, however, we have not implemented violation and fulfillment flags to mark specifically temporal aspects, which would have to be an extension to the system of flags. Finally, in Example (129d), with respect to an agent having executed a particular action, the contract state is updated. This is *not* the same thing as saying that the action *itself* updates the contract state, which actions cannot do. This last example is particularly relevant to our discussion of interruptable obligations in section 5.5.7.2.3. In these ways, our implementation can be used to express a range of implications. P represents an arbitrary literal, while Q, ..., W represent arbitrary actions.

**Example 129**

a. If \( P \), then \( OB(Q) \)

b. If \( OB(R) \), then \( OB(S) \)

c. If the current time is less than or equal to 2 and \( OB(T) \) has been fulfilled, then remove \( OB(T) \) from the contract state.

d. If the current time is greater than 2 and \( OB(T) \) has not yet been fulfilled, then add \( OB(U) \) to the contract state and add a violation flag of \( OB(T) \) to the history.

e. As a consequence of an agentive action having been executed, the contract state is updated, e.g. the execution of agentive action \( A \) triggers the introduction of \( OB(W) \).

These inferences have the following forms, respectively, which we could add to a rule relation. Note that expressions of the form \((\text{giveAgAcSimple Jill Action6 testLex-Actions04})\) are functions which provide an agentive action relative to the lexicon; \(<=\) and \(>\) are *less than or equal to* and *strictly greater than*, respectively. These examples are provided to clearly illustrate *how* the system would implement implications along the lines of the Examples (129a)-(129e). In the following, we abbreviate that section of the code in Code 17 which retrieves the *contractFlagState* from the *soaHistorical* with *RetrieveContractFlagState*; this is the *contractFlagState* which is modified.
Data 17  \[\text{elem prop1\ (#properties\ (doRDSMOD2\ (giveAgAcSimple Jill Action6\ testLexActions04)\ soaHistorical02 \[\])) =}
\]
\[\text{(properties =}
\]
\[\text{(#properties\ (doAgentiveActionSOAHistContract}
\]
\[\text{inAgentiveAction\ inSOAhist\ inPropList)),}
\]
\[\text{contractFlagState =}
\]
\[\text{(addToContractFlagState}
\]
\[\text{(obligatedCompFlag}
\]
\[\text{(giveAgAcSimple Jill Action7}
\]
\[\text{testLexActions04\ testLexActions04 \[\])}
\]
\[\text{RetrieveContractFlagState)}\]

Data 18  \[\text{subSet\ (Set\ (obligatedCompFlag\ (action = (label = Action6, xcond = [prop1,prop7,prop5], ycond = [prop3,neg-prop4,neg-prop6], agent = Jill)
}\]
\[\text{testLexActions04 \[\]))\ (Set\ (#contractFlagState}
\]
\[\text{(doRDSMOD2\ (giveAgAcSimple Jill Action6\ testLexActions04)
}\]
\[\text{soaHistorical02 \[\]))\) =}
\[\text{(properties =}
\]
\[\text{(#properties\ (doAgentiveActionSOAHistContract}
\]
\[\text{inAgentiveAction\ inSOAhist\ inPropList)),}
\]
\[\text{contractFlagState =}
\]
\[\text{(addToContractFlagState}
\]
\[\text{(obligatedCompFlag}
\]
\[\text{(giveAgAcSimple Jill Action10}
\]
\[\text{testLexActions04\ testLexActions04 \[\))}
\]
\[\text{RetrieveContractFlagState)}\]

In the following, recall that historyFlag02 reflects that Jill has fulfilled her obligation to execute Action6.

Data 19  \[\text{(#time\ soaHistorical02 < = 2) \&\&\ (elem\ historyFlag02\ (#history\ soaHistorical02)) =}
\]
\[\text{(properties =}
\]
\[\text{(#properties\ (doAgentiveActionSOAHistContract}
\]
\[\text{inAgentiveAction\ inSOAhist\ inPropList)),}
\]
\[\text{contractFlagState =}
\]

236
In the following, the obligation for Jill to execute Action\textsubscript{6} is removed, a violation is noted in the history, an additional obligation is incurred, and a violation flag is introduced. Note that the particular flag is introduced for clarity; in a more general expression, several functions would provide the relevant flag.

**Data 20** \((\#\text{time} \text{soaHistorical02} > 2) \&\& \neg (\text{not} (\text{elem} \text{historyFlag02} \text{#history soaHistorical02}))) =\)

\[
\begin{align*}
\text{(properties =} \&\& \\ \text{(doAgentiveActionSOAHistContract} \\
\text{inAgentiveAction inSOAhist inPropList)),} \\
\text{contractFlagState =} \\
\text{(addToContractFlagState (prohibitedCompFlag}} \\
\text{(giveAgAcSimple Jill} \\
\text{Action3 testLexActions04} \\
\text{testLexActions04 []})} \\
\text{(subFromContractFlagState (obligatedCompFlag}} \\
\text{(giveAgAcSimple Jill} \\
\text{Action6 testLexActions04} \\
\text{testLexActions04 []})} \\
\text{RetrieveContractFlagState}
\end{align*}
\]

In the final example, the trigger tests whether the agentive action which has Jill as the agent and Action\textsubscript{6} as the action appears in the history. If it does (i.e. returns \text{true}), then the contract state is updated with a prohibition on Jill’s executing Action\textsubscript{72}.

**Data 21** \(((!!0)\{((\text{Jill} =\ (#\text{agent } x)) \&\& (\text{Action6} =\ (#\text{actionDone } x)) \\
| x <\ (- (\text{history} \text{doAgentiveActionSOAHistContract} \\
\text{inAgentiveAction inSOAhist inPropList})) \} = \text{true}))
\]

\[
\begin{align*}
\text{(properties =} \&\& \\ \text{(doAgentiveActionSOAHistContract} \\
\text{inAgentiveAction inSOAhist inPropList)),} \\
\end{align*}
\]
Chapter 5

5.5 Overview of the Implementation

\[
\text{contractFlagState} = \\
(\text{addToContractFlagState} \\
(\text{prohibitedCompFlag} \\
(\text{giveAgAcSimple Jill Action72} \\
\text{testLexActions04}) \text{testLexActions04} ))) \\
\text{RetrieveContractFlagState}
\]

While we have not explored or presented the full range of implications one might want, we have shown how the implementation allows one to express a relevant range as well as provides the means to express others of interest.

The rules may be realised in other ways. For instance, to represent deadlines, we might have incorporated temporal indices into the records which express the \text{ContractFlagStates}, but have not, essentially to keep the overall design issues and the implementation as fundamental and clear as possible, leaving other issues as extensions to the basic framework. Instead, we have used more complex rule expressions to represent deadlines. Asides from introducing further information into the \text{ContractFlagState}, we could alternatively introduce functions to match elements of the historical record in other ways to create less specific triggers; we have seen such functions where we match the executed agentive action against an agent and action in the \text{ContractFlagState} to output a \text{ContractFlag}. For instance, we could have a trigger associated with just the fields for Jill as the agent or a trigger associated with Jill and a violation. Such triggers would mean that some \text{ContractFlagState} update occurs whenever Jill appears in the history or whenever we find a record with Jill associated with a violation. Such alternatives represent different abstractions over what triggers the update. For simplicity, we do not develop this discussion further here.

The discussion so far has focussed entirely on \text{simple agentive actions}, which are agentive actions which have no action combinators other than opposition. However, we have made it a fundamental goal to show that deontic specifications and the consequences which follow from them can be given for \text{complex} expressions as well. In one sense, we have \text{just made this point}, for \text{any} way of enriching the records that comprise the \text{ContractFlagState}, including them in the \text{History} and using the records (or abstractions on them) would allow us to represent deontic specifications on more complex expressions as well as more specific inferences which follow relative to fulfillment or violation.

The point, then, of the following exercise, is essentially to \text{extend} the basic framework we have already provided to cover different approaches to deontic specification on \text{sequences of actions}. We will, then, not consider \text{choice} or \text{simultaneous} combinators. Our discussion

\[238\]
also connects to our earlier analysis of the difference between sequences of obligations and obligations on sequences.

5.5.7 Deontic Specifications on Complex Actions

We have argued throughout the thesis that deontic specification of complex actions must have violation or fulfillment markers that are as rich as the structure of the complex action; that is, deontic specification on a complex action cannot be reduced to a representation which only makes use of one primitive violation or fulfillment marker. We show elements of how we have implemented this claim. Note that, in the interests of getting to our point, we present the results schematically, not the full code or the issues.

In section 4.3, we discussed different interpretations of obligations on sequences of actions – OB\textsubscript{dist} distributive, OB\textsubscript{col} collective, and OB\textsubscript{int} interruptable. These are different ways that one could introduce deontic specification on a sequence.

In some ScDLs ([130]), a sequence of actions implies that the sequence forms a one-step action that cannot be interrupted, and there is no complex violation/fulfillment marker which represents the deontic specification on the complex action. Here, we take a different tack, which was discussed in chapter 4. This has two parts. First, we maintain the structure of the complex action – the actions which are input, the operator which combines them, and their output. The deontic operator applies to the complex action relative to this structure (which we do not present) such that the complex violation or fulfillment marker appears in the contract state. Second, the update of the contract state is handled by the rule function component. With this, we have attained our goal of demonstrating and implementing a crucial point, which is the introduction, justification, analysis, and implementation of complex violation and fulfillment markers relative to complex actions. Furthermore, our discussion here motivates the use of our lexical semantic functions in cases where we want the collective interpretation of an obligation on a sequence.

5.5.7.1 Complex Action Combinators

In order to show our implementation, we construct richer markers for deontic specification on complex actions. In order to construct these richer markers, we represent a sequence such as $\alpha;\beta$ with a richer structure which distinguishes the input actions $\alpha$ and $\beta$, the resultant action (if any) $\gamma$, and the mode of formation, such as the sequence operator. The deontic specifiers can then access different component parts of the complex action representation. This allows us to define a range of deontic specifications, as discussed
below. In other words, we define the different obligation operators to be sensitive to the
syntactic structure of the complex action.

We implement complex actions as records. We want our simple or complex actions to
have a uniform type for two reasons. First, we want our action execution function to apply
uniformly to simple or complex actions. Second, we want our deontic operators to apply to actions of either type. Therefore, we provide the means to shift the type of actions to this higher type ([136] for similar arguments on intensional types). This requires null and opposite combinators to apply to our basic actions. Otherwise, we have standard action combinators for action sequence, choice, and simultaneity. For brevity and relevance, we give null, opposition, and sequence combinators, but only discuss sequence. Complex actions have fields for the input actions, the complex action operator, and the result of the application of the operator to the input actions. The particular functional result of applying the operator to the actions provides the output.

We use the following three types.

Code 18

\[
\begin{align*}
\text{type } & \text{ActionOperator} = \text{String} \\
\text{type } & \text{ActionOperators} = [\text{ActionOperator}] \\
\text{type } & \text{ComplexActionCons} = \text{Rec (inActionA :: AgentiveAction,} \\
& \quad \text{inActionB :: AgentiveAction, inActionOperator :: ActionOperator,} \\
& \quad \text{outAction :: AgentiveAction)}
\end{align*}
\]

Below, we have the ActionOperators and ComplexActionCons for the null, opposite, and sequence action combinators.

Code 19

\[
\begin{align*}
\text{actionOperators1} & = [\text{Null, NegAct, Seq, Chc, Sim}] \\
\text{nullComplexAction} & :: \text{AgentiveAction} \rightarrow \text{AgentiveAction} \rightarrow \text{ComplexActionCons} \\
\text{nullComplexAction inAgentiveAction1 inAgentiveAction2} & = \\
\quad (\text{inActionA} = \text{inAgentiveAction1}, \\
\quad \text{inActionB} = \text{inAgentiveAction2}, \text{inActionOperator} = \text{NULLAC}, \\
\quad \text{outAction} = \text{inAgentiveAction1}) \\
\text{negComplexAction} & :: \text{AgentiveAction} \rightarrow \text{AgentiveAction} \rightarrow \text{ComplexActionCons} \\
\text{negActComplexAction inAgentiveAction1 inAgentiveAction2} & = \\
\quad (\text{inActionA} = \text{inAgentiveAction1}, \\
\quad \text{inActionB} = \text{inAgentiveAction2}, \text{inActionOperator} = \text{NegAct},
\end{align*}
\]
It is important to note several issues with these types and functions, which are simplifications of the operative functions in several respects. First, the nullComplexAction function may appear redundant, and there are several alternatives, for instance, providing only a designated null action as input; however, as we noted, for our purposes, it is easier to have uniform types for the actions. The output agentive action found in the outAction field is simply the first input agentive action, so the second input agentive action has no effect. Second, the negComplexAction function is, fundamentally, just our findOpposites function, albeit with a vacuous input of an agentive action and missing utility inputs; here the output agentive action is the result of applying this function to the first input agentive action. Finally, the seqComplexAction function is also missing several utility inputs, in particular, sequenceActions. Informally, the outAction is, in this case, function composition of the input actions (pace several restrictions on well-formedness and alternative formulations): the preconditions of outAction are the preconditions of inActionA; the postconditions of outAction are those of inActionB together with those of inActionA which remain by inertia; the preconditions of inActionB must be a subset of the postcondition properties of inActionA; and the postcondition properties of outAction must otherwise be consistent. One can say that the constraints are the same as one might find where the first action is executed followed immediately by the second action, meeting all the constraints on action executions in the inferred intermediate SOA. However, as in state-changing logics, the result of the application of the sequence operator is to create a one-step action. It should also be pointed out that the assumption here is that complex action operators, as functions which create actions from actions, are sensitive to the lexical constraints placed on the rest of the domain of actions. For instance, if we have a transition constraint on actions, say that no action allows a transition from a SOA where prop1 holds to a SOA where a prop5 holds, nor can any complex action derived from two actions by an action combinator generate such an action. Such restrictions may be arbitrary or principled, but they restrict what appears in the lexicon by filtering the lexicon. In contrast, there need be no corresponding restriction on one action applying after another without using a complex action combinator such that the precondition of
first action has prop1 and the postcondition of the second action has prop5. Finally, we assume we construct labels for complex actions; for example, where ActionA, ActionB, and ActionQ are actions that can be sequenced, if we have ActionA followed by ActionB in sequence, then we produce the label of an action in the lexicon which has the resultant preconditions and postconditions, say ActionQ, which the action in the output.

5.5.7.2 Obligations on Sequences

In ([130]), obligations on sequences are reduced to sequences of obligations on the component actions. In ([106]), obligations on sequences are irreducible to sequences of obligations, but rather are obligations on the sequence per se. Yet, in neither case, do we find distinct markers which represent fulfillment or violation of the obligation with respect to the sequence. Our analysis and implementation provides ways to articulate these differences. We use action labels found in a sample lexicon. The sequence is Action175 followed by Action2158. We suppose Jill is the agent of the sequence. Furthermore, Action137 is the opposite of Action175 and Action2134 is the opposite of Action2158. Let us consider OB_dist, OB_coll, and OB_int in order.

5.5.7.2.1 OB_dist The distributive interpretation of obligation as in ([130]), \( OB_{\text{dist}}(\text{Action175};\text{Action2158}) \), implies \( OB_{\text{atomic}}(\text{Action175}) \land [\text{Action175}] (OB_{\text{atomic}}(\text{Action2158})) \). In this case, the OB operator applies to each part of the complex action, but also introduces a modification of the rule function. The \( OB_{\text{atomic}} \) is just our obligation operator from earlier. We need two components. First, we have an initial contract state

\[
\text{Data 22} \quad [(\text{actionDone} = \text{Action175}, \text{agent} = \text{Jill}, \text{deonticSpec} = \text{Obligated}, \text{onSpec} = \text{Action175}, \text{valueFlag} = \text{Fulfilled}),
\quad (\text{actionDone} = \text{Action137}, \text{agent} = \text{Jill}, \text{deonticSpec} = \text{Obligated}, \text{onSpec} = \text{Action175}, \text{valueFlag} = \text{Violated})]
\]

In addition, we modify the rule function such that where the first action has been executed (checked in the history), then the obligation on the second action of the sequence is introduced. Such a trigger and rule is along the lines as presented in Data 21 of section 5.5.6.3. Note that automating such modifications of the rule function over the course of contract state change would require the rule function to be separated out from the execution function. We have not done so, and the modifications of the rule function here are made manually.
This results in the following contract state, which specifies the fulfillment and violation cases for each of the component actions after the execution of the first action of the sequence. Notice that in this example, we have not removed the obligation \( \text{OB}_\text{atomic} \) (Action175) from the contract state, though we have introduced an obligation \( \text{OB}_\text{atomic} \) (Action2158). As we have noted earlier, the designer can specify just how contract states are modified. For example, one could design a rule function to have removed \( \text{OB}_\text{atomic} \) (Action175) if it is fulfilled.

\[
\text{Data 23} \quad [(\text{actionDone} = \text{Action175}, \text{agent} = \text{Jill}, \text{deonticSpec} = \text{Obligated}, \text{onSpec} = \text{Action175}, \text{valueFlag} = \text{Fulfilled}),
\text{(actionDone} = \text{Action137}, \text{agent} = \text{Jill}, \text{deonticSpec} = \text{Obligated}, \text{onSpec} = \text{Action175}, \text{valueFlag} = \text{Violated}),
\text{(actionDone} = \text{Action2158}, \text{agent} = \text{Jill}, \text{deonticSpec} = \text{Obligated}, \text{onSpec} = \text{Action2158}, \text{valueFlag} = \text{Fulfilled}),
\text{(actionDone} = \text{Action2134}, \text{agent} = \text{Jill}, \text{deonticSpec} = \text{Obligated}, \text{onSpec} = \text{Action2158}, \text{valueFlag} = \text{Violated})]
\]

We say that the obligated sequence has been fulfilled where the obligations on each action have been fulfilled and in the right order. As we have previously noted, even in such cases, there may be other actions executed between the execution of the first and second actions of the sequence, but which do not give rise to violation or fulfillment.

5.5.7.2.2 \( \text{OB}_{\text{coll}} \) In contrast, we could represent the interpretation of [106] by applying the operator to \text{Action179} with a collective interpretation of obligation, \( \text{OB}_{\text{coll}} \); in this case, the deontic operator applies directly to the result of the sequence operator. It does not modify the rule function. We suppose that \text{Action141} is the opposite of \text{Action179}:

\[
\text{Data 24} \quad [(\text{actionDone} = \text{Action179}, \text{agent} = \text{Jill}, \text{deonticSpec} = \text{Obligated}, \text{onSpec} = \text{Action179}, \text{valueFlag} = \text{Fulfilled}),
\text{(actionDone} = \text{Action141}, \text{agent} = \text{Jill}, \text{deonticSpec} = \text{Obligated}, \text{onSpec} = \text{Action179}, \text{valueFlag} = \text{Violated})]
\]

5.5.7.2.3 \( \text{OB}_{\text{int}} \) The most interesting case is the interruptable notion of obligation on a sequence, \( \text{OB}_{\text{int}} \) (Action175:Action2158). Recall that \text{Action137} is the opposite of \text{Action175}, \text{Action2134} is the opposite of \text{Action2158}. Furthermore, \text{Action179} is the result of the sequence operation on (Action175:Action2158). In this case, depending
on the execution of Action175, Action2158, Action137, and Action2134 violation or and fulfillment flags arise relative to the whole sequence labelled with Action179. Which flag is given depends on which action is executed and in which order. In a sense, it is a combination of the techniques used for OB\textsubscript{dist} and OB\textsubscript{coll}. Like OB\textsubscript{coll}, OB\textsubscript{int} introduces a violation and fulfillment flag with respect to the complex action; like OB\textsubscript{dist}, it modifies the rule function. In particular, where the first action of the sequence is not executed, then the History is modified to reflect a violation of the obligation on the sequence. If the first action is executed, then as in Data 21, we update the contract state with an obligation on the execution of the second action. If the second action is executed, then we mark the history with a fulfillment of the obligation on the sequence Action179; if the opposite of the second action is executed, then we mark the history with a violation of the obligation on the sequence Action179. We say that this obligation is interruptable since between the execution of the first and second actions, other actions can be executed which do not give rise to either a violation or fulfillment of the obligation on the sequence or any other deontic specification.

OB\textsubscript{int} \((\text{Action175};\text{Action2158})\) has two parts. First, it provides an initial contract state which we add to the contract state of the soaHistorical; this contract state indicates a violation of the sequence where the opposite of the first action of the sequence is executed. Second, in the rule function, it updates the contract state of the soaHistorical so that where the second action of the sequence is executed, then fulfillment of the sequence is flagged, while where the opposite of action of the second action of the sequence is executed, then violation of the sequence is flagged.

In the initial contract state, Action137 is the opposite of the first action of the sequence Action175. In this case, the flag for violation is relative to the complex action Action179 per se:

Data 25 \([(\text{actionDone} = \text{Action137}, \text{agent} = \text{Jill}, \text{deonticSpec} = \text{Obligated}, \text{onSpec} = \text{Action179}, \text{valueFlag} = \text{Violated})]\),

Next, we provide the elements which are added to the contract state subsequent to an execution of the first action of the sequence. At this point in the development of the implementation, we have to manually provide the records in the contract state that are added since we have not implemented functions to pass the label of the complex action into the relevant contract state. Nonetheless, this provides a straightforward realisation of interruptable obligations on sequences of actions in the expressions of the implementation.
Chapter 5 5.6 Inputs to and Outputs of the Program

We see that we can define deontic specifications on complex actions in different ways, which may be designed to suit particular purposes and interpretations. The language is thus expressive and can be used to implement different notions of values applied to actions for the purposes of simulation in a multi-agent system.

5.6 Inputs to and Outputs of the Program

At this point, let us quickly review the components that the user inputs into the system.

First, it is important to recall that the program applies to abstract actions. If a user wishes to model a “real” contract expressed in natural language such as Bill is obligated to move the left toggle up and Jill is prohibited from moving the right toggle left, then one must decide some mapping from the natural language expressions to the abstract actions.

Second, the program we have provided is a tool in which alternative definitions of the deontic operators and their relations one to another can be given. It is thus up to the designer of the system to determine just which definitions to use.

Given abstract actions, the user must then specify an initial value specified context as well as a lexicon. Recall that a value specified context is comprised of:
Chapter 5

5.7 Some Comparisons

- A *state of affairs*, which represents the properties of the current state and indices for world and time.

- A *valued specification list*, which is the list of deontic specifications on actions.

- A *history*, which records what has occurred.

- A *rule function*, which is a function from what has occurred in the history to an updating of the *valued specification list*.

The purpose of the *lexicon* is to constrain the search space for actions which are used to trigger violation or fulfillment of a deontic specification. In addition, the user might also define the deontic operators and their relations.

If a designer has already defined the deontic operators and their relations, provides a lexicon, and a *value specified context*, then all the user need input is the action which is executed relative to the lexicon and *value specified context*. The output is the resultant *value specified context*. In the output, we record in the history all the actions which have been executed to that point. In the VSL at output, we have a representation of what is obligated, prohibited, and permitted. The processes involved are described in section 5.3.

A variety of tests could also be made available. For example, conflicting obligations, where an agent is obligated to do an action and obligated not to do that action can be expressed in the program. This is simply an obligation on an action and an obligation on the negation of that action. A designer can allow such conflicting obligations in a contract or bar them. We have barred vacuous obligations, namely those which are obligations on actions which cannot be violated.

### 5.7 Some Comparisons

There have been several recent efforts to formalise and operationalize norms, what we call deontic specifications, in the context of e-contracting and Multi-agent Systems ([123], [62], [20], [70], [164], [166], [10], among others). In the Multi-agent Systems literature, norms are discussed with respect to how they are used to organise the behaviour of individuals and collectives within the society or institution. A range of components are presented which do not bear directly on our analysis or implementation such as the internal structure of agents (i.e., their beliefs or motivations), notions of global utility of actions, the different roles agents may play in an organisation, the spread of norms in an organisation, or communication languages. While interesting and relevant, we are not concerned in this
thesis with the presentation of such high level elements of institutions or social organisations, so focus on those components which bear on norms. Moreover, other proposals have strengths in comparison to our implementation such as user interfaces, querying, or planning, none of which we discuss here. Finally, we have not presented an abstract formalism, we do not model system behaviour from an abstract point of view and so cannot verify properties of the system.

[122] and [123] provide a high level abstraction of open societies, covering a range of issues which are not directly relevant to our central concerns. They claim there is a lack of a consensus around the concept of norm and do not consider the range of issues specifically tied to deontic logic as in [128]. Rather, they claim to provide a high level abstraction of norms and open societies. Some of the general characteristics of norms are that they prescribe behaviour in social contexts and are related to goals. Norms are said to have beneficiaries, exceptions, and rewards or punishments. It is claimed that norms are either fulfilled or unfulfilled, which are introduced as literals; for example, [123, p. 234]“....the fulfillment of a norm is assessed through its normative goals....” We may interpret this as follows: suppose $\text{OB}(P)$, where $P$ is a “goal”; where $P$ and $\text{OB}(P)$ hold, then $\text{OB}(P)$ is fulfilled; where $\neg P$ and $\text{OB}(P)$ hold, then $\text{OB}(P)$ is violated. However, it does not seem to us that in either of these cases $P$ or $\neg P$ ought to be considered a “goal” in its own right. The CTD structure is given as interlocking norms, where norms are introduced or eliminated by fulfillment or violation of other norms. A spectrum of distinct categories of norms are suggested beyond the deontic concepts such as social commitments, social codes for behaviour, enforcement and reward norms, legislative norms. Norms can be introduced, maintained, or eliminated; agents can adopt or deliberate over them.

In contrast to our discussion of the deontic notions, norms need not specify violation conditions. Thus, what we called epistemic norms or generics may well be considered norms for the purposes of [123]. Moreover, the level of abstraction leaves a range of issues that are relevant to our central concerns unclear such as the definition of action or action negation, notions of complex actions, notions of deontic specification of an action, and articulated violation and fulfillment markers. Indeed, while violation and fulfillment flags appear, it is unclear whether norms are or are not reduced to them. Norms are defined using a notion of goal, which we do not use; it is unclear that the conception is appropriate to represent violation of deontic specifications, which may arise for reasons other than the agent’s goal directness with respect to that particular action. Moreover, little consideration is given a comparison and contrast of the analysis of norms with SwDL or ScDL, particularly concerning the CTD structure.
Chapter 5

5.7 Some Comparisons

[70] represent contracts using an XML formalisation of the Event Calculus which keeps track of the normative state of a contract as it is executed over time. However, the Event Calculus does not have an explicit semantics in terms of labeled transitions between states. The normative state is comprised of normative relations which cover concepts such as obligation, prohibition, the Hohfeldian concepts of power and right, among others. However, normative relations are not discussed in relation to the literature on deontic logic. Over the course of the execution of the contract, contract variables are assigned values in relation to the normative relations, marking fulfillment or violation of the normative relation. The Event Calculus [108] (and closely related to [55] and [142]) expresses how fluents (propositional properties of a domain, which are literals in our system) change over time with respect to the occurrence of events. Events initiate or terminate periods of time during which fluents have a particular boolean value which holds given a condition, where the condition refers to the values of other fluents and the occurrence of other events. Temporal extents can be specified.

Norms are evaluated with respect to state norms, which are a subclass of fluents. Events can be specified with respect to having violated or fulfilled a particular normative relation, and in this way, the system keeps track of violations and fulfillments. Conceptually, in our view, this is similar to deontically specified actions. An example contract is represented in this language.

[70] do not discuss the CTD problem or other problems related to deontic logic; they do not provide a means to update contract states as a consequence of the execution of deontically specified action and with respect to a rule relation. This is consistent with [69], which explicitly argues against using deontic logic for service-level agreements. While events are conceptually related to our definition of actions, [70] do not represent complex actions such as action negation or sequence. There is no representation of complex violation and fulfillment markers. Thus, it is unclear how they would represent deontic specification of complex actions. Finally, the notion and use of a history is unclear, though it is essential in our implementation.

[164] and [166] develop nC+, which is an extension of C+ that incorporates a notion of permission. C+ is a formalism with semantics in terms of Kripke models with transitions where one can specify and reason about the effects of actions as well as inertia. An action description is given as a set of rules which define a labeled transition. nC+ partitions states and transitions into red/green, or legal/illegal, or ideal/subideal. While these classifications can be associated with a concept of permission, the concept is not explicit. Broadly, our implementation is compatible with this approach. For example, actions are labeled transitions between SOAs which have sets of literals. In addition,
the alternative classifications are compatible with our tool, as the value operator labels and value flags may also have different interpretations, though we have focussed on their deontic interpretation. However, in our implementation, we have no abstract expressions of causal laws, though the static laws and fluent dynamic laws of nC+ can be expressed in the implementation as in section 5.5.6.3. [166] discuss constraints on transitions, which we have not discussed.

Our analysis has focussed on the CTD structure and contract state modification, which is not discussed in [166]. There is no notion of lexical semantics of actions so as to reason about what actions give rise to fulfillment or violation of an obligation. Finally, [166] do not represent complex actions such as sequence and the variety of interpretations it has.

[10] presents a framework to provide executable specifications of open agent societies which are instances of normative systems. It is formalised in both the Event Calculus and C+. Thus, the framework brings together various strands discussed above. In this framework, it is essential to specify normative relations such as obligation. In addition, an emphasis is placed on the representation of institutionalised power, where an agent is empowered to create relations or states of affairs. We have not addressed such issues of power, and it is unclear to us the necessity of this for the representation of the deontic concepts, however important it may be in general. Four levels of social constraints are presented: physical capabilities; institutionalised powers; permissions, prohibitions, and obligations; and sanctions and enforcement. We have only addressed the latter two.

In the C+ version of the presentation, social states are an interpretation of fluent constants, which include permissions, obligations, and sanctions. We can understand these as elements of what we represent in our contract states; thus, actions in the framework can change the values of these fluents.

One crucial difference between [10] and our proposal is that [10] relate deontic concepts to power. For instance, an agent is permitted to perform an action if an only if that agent has the institutional power to perform the action. While we understand how institutional powers may be a key precondition for the assignment of a deontic specification, once assigned, it is unclear what role institutional power subsequently plays.

We have a case where an obligation arises as a consequence of some other action, thus appearing as one of our cases of contract state modification. Obligations are said to be discharged if the obligated action is executed, but this is given informally. Sanctions arise relative to times and actions executed, not relative to a deontic specification. In other words, it appears that in contrast to our proposal, obligations are not necessarily reduced to violation and fulfillment flags.
The Event Calculus version of the presentation is very similar in terms of the deontic notions as the C+ version: permission is also understood in terms of power; obligations are \textit{initiated} contingent on actions and fluents. However, in contrast, sanctions arise dependent on obligations and actions.

In both approaches, deontic specification on complex actions are not taken into consideration. Action negation in terms of lexical semantics is not provided. Violations of obligations arise with respect to actions that are not executed with respect to deadlines, which we have argued is not central to the notion of obligation. \cite{10} discuss disadvantages of a labeled transition system such as C+: there is no representation of a history nor a direct representation of a deadline. In our implementation, a history is a component of a \textit{Value Specified Context} and is modified by the execution function relative to an action. For us, deadlines for deontic specifications are represented in the rule functions, which are similar to the constraints on termination and initiation of C+. Finally, neither the Event Calculus or C+ frameworks make the sharp distinctions we do between the results of actions and the secondary effects that arise relative to the contract state and rules relation.

In other work, \cite{76}, \cite{3}, and \cite{4} use deontic specifications to filter out or sort actions, which does not represent the essence of the deontic notions since violations are ruled out rather than reasoned with. \cite{23} present an architecture for normative systems which is similar to ours in that deontic specifications \textit{add} information to basic structures. However, it is unclear how they implement their design, integrate complex actions, or account for the CTD problem.

\section*{5.8 Summary}

We have sketched the issues and implementation of the \textit{Abstract Contract Calculator}. We have pointed out the key role of calculating \textit{action opposition} and \textit{violation and fulfillment} markers for complex actions. We have shown the importance of violation and fulfillment markers for reasoning with the CTD case. We sketched the implementation in a Haskell program. The language allows the expression of alternative concepts of the deontic notions. One can input an agentive action, whether simple or complex, and determine, relative to a contract, whether that action violates or fulfills an obligation. However, we have not presented key notions such as \textit{consistency} constraints on contract states and \textit{implicational} relations between deontic specifications. For this, we need to provide the \textit{negation} of a deontic specification and functional meaning postulates.
Chapter 5

The main strength of the approach is that it allows alternative definitions of actions and deontic notions to be represented and animated. One can select which, out of those alternatives, most accurately represent one's intuitions. However, the program is not a logic, even if logic-based. It needs to be expressed in an explicit logic. As the implementation uses abstract actions and has no temporal operators, it has limited application.

5.9 Next Chapter: Conclusion

In the next chapter, we conclude the thesis with an overview of how the thesis has addressed the main aims set out in chapter 1 as well as points for future work.
Chapter 6

Conclusion

6.1 Introduction

In this final chapter, we review what has been covered in the thesis, how the aims have been satisfied, and some suggestions for future work.

6.2 Summary of the Results of the Thesis

The thesis reports our research on the language, logic, and automated processing of the deontic concepts represented by the operators obligation, prohibition, and permission on complex, abstract actions, focusing largely on obligation. We recall our general objective and the outline of our contributions. We then recall the contributions by chapter.

6.2.1 General Objective and Contributions

General Objective:

to provide a flexible, open framework and implemented tool in which we can express and exercise alternative definitions of the deontic concepts as applied to complex actions.

To meet this general objective, the thesis contributed the following:
i. *linguistically* well-motivated semantic interpretations of the deontic concepts as well as *linguistically* well-motivated logical forms for expressions which contain the deontic concepts. These interpretations and logical forms are suitable for the formal representation of contracts.

   a. distinction between epistemic and non-epistemic interpretations of the deontic operators;
   b. restriction of the available forms of conditional obligations;
   c. solution to the *Gentle Murderer Paradox* using *focus*;
   d. solution to the *Good Samaritan Paradox* using *Discourse Representation Theory* and introduction of the related *restrictive relative clause* problem;
   e. introduction of the *Antonym Restriction* on obligations.

ii. a reanalysis of the CTD Structure in State-wise Deontic Logics;

   a. abstraction and generalisation of the CTD structure;
   b. introduction of the violation and fulfillment markers as intermediate concepts in State-wise Deontic Logics;

iii. a reductionist, fine-grained analysis of deontic operators in Dynamic Deontic Logic.

   a. introduction of complex violation and fulfillment markers for State-changing Deontic Logics;
   b. reanalysis of the CTD structure in State-changing Deontic Logics;
   c. distinction between obligations on sequences and sequences of obligations;
   c. interpretation of action negation as the lexical semantic relation of *antonym*;

iv. a generic prototyping tool for exercising the deontic operators relative to some particular definition of them;

   a. generation of *trails of executions of actions by agents* which *indicate the agents’ violation or fulfillment of the deontic specifications*;
   b. introduction of *contracts* as deontic specifications on actions and a rule function;
   c. violation and fulfillment markers recorded as *secondary* effects of the execution of an action and relative to a contract;
   d. lexical semantic functions on actions;
   e. alternative definitions of deontic specifications on complex actions;
6.2.2 Contributions by Chapter

Chapter 2 made the contributions in (i). We presented two versions of SwDL – *Standard Deontic Logic* and the *Kanger-Andersonian Reduction*. We discussed a range of issues that arise with these approaches, clarifying the relationship between *ideality* and *violation*, pointing out that the reduction facilitates explicit reasoning with respect to violation (and fulfillment). However, we pointed out that the approach problematically implies that all obligations are synonymous. We pointed out that the problem can be solved with articulated violation and fulfilment markers.

It is well-known that SwDL suffers from a range of paradoxes and divergent opinions on the interpretation of the operators and logical forms of expressions with the operators. To clarify these issues, we examined the interpretations and logical forms linguistically. We distinguished between *epistemic* and *non-epistemic* interpretations of *ought*, and asserted that only the *non-epistemic* interpretation is relevant to legal contracting. A range of constructions were examined with the aim of simplifying the discussion of the CTD paradox, which appears in chapters 3 and 4 as well as other issues related to deontic operators that are relevant to the implementation in chapter 5. We showed that linguistic tools can motivate logical forms which restrict the logical forms or obviate paradoxes. For example, we argued for a restricted form of *conditional obligation*, and we provided solutions to the *Gentle Murderer Paradox* and the *Good Samaritan Paradox*. We also took well-motivated positions on issues such as deadlines, conflicting obligations, the relationship between *permission* and *obligation*, and introduced the *Antonym Restriction* on obligations. Thus, we removed a range of conflating and obscuring factors from consideration.

In chapters 3 and 4, we provided the contributions in (ii) and (iii). Based on our discussions in chapter 2, we focussed on a particular CTD structure. In chapter 3, we reanalysed the CTD structure, abstracting and generalising it. We drew particular attention to the various roles of components of the structure, and we showed that the problems which arose could be clarified with the introduction of violation and fulfillment markers. We also outlined how our proposal met the criteria for an adequate theory of the CTD structure as presented in [30]. The proposal of [30] was analysed and compared to our proposal, concluding that it was not appropriate for legal contracting.

In chapter 4, we continued our analysis of the CTD structure with reference to dynamic theories of the deontic operators. In this chapter, we showed that, contrary to claims, dynamic theories did not have an adequate analysis of the CTD structure. Rather the proposed analysis gave rise to counterintuitive results as well as conflated sequences of obligations with obligations on sequences. We provided our semi-formal analysis in terms
of fine-grained markers for violation and fulfillment which took into account different structures of complex actions. In addition, we argued for an interpretation of action negation in terms of the lexical semantic notion of antonym. We used this analysis to show a superior analysis of the CTD structure.

Finally, in chapter 5, we fulfilled our general objective. We outlined our generic, prototype tool in which different definitions of deontic specification can be cast and animated. As we discussed, the fundamental architecture of the tool can be used to incorporate additional aspects of deontic reasoning such as needed for contract state modification relative to literals in the SOA or temporal sequences. The implementation has states-of-affairs, which represent literals at a time, and basic types for actions and agentive actions, which are state-transitions. We provide a version of the Frame Axiom and incorporate it into our execution function, restricting the effect of the execution of an action. One novel aspect of the analysis is the representation of the lexical space of actions as well as the lexical semantic functions. This provides a way to calculate the antonym of a simple or complex action in the lexical space, if there is one. Deontic specifications are implemented as reductions to fine-grained violation and fulfillment markers. Contract states, which are sets of deontically specified actions, are manipulated with a rules function which is executed relative to a history, accounting for basic properties of the CTD structure. In addition, we gave examples of how rules could be introduced which express other ways to motivate the modification of the contract state. Finally, we showed how complex actions are formed and deontically specified in such a way as to allow deontic specification of sequences with collective, distributive, and interruptable interpretations. Thus, we show how the framework is open and flexible. In general, we have implemented a system which can represent and animate deontic specifications on complex actions in a reductionist analysis.

6.3 Limitations

In this section, we discuss several of the limitations of this study. Broadly speaking, the limitations are those of omission since we had to pick and choose among a plethora of topics in the extensive body of literature, each of which is worthy of a thesis itself.

In chapter 1, we related our study to legal and electronic contracting. However, a significant limitation of the research is the degree of abstraction. Indeed, we have not translated an existing legal contract, written in natural language, into a format which could be processed by the implementation. Nor did we consider in depth research on the law or
electronic contracting which is more closely related to processing legal contracts in natural language; we have discussed some of our reasons for this. Along these lines, there is much research yet to be done on the development of a corpus of contracts which is then subjected to computational linguistic techniques of text-mining to determine just what structures appear in them.

An additional limitation is that to deploy our implementation of the deontic concepts with respect to natural language would require a more substantial analysis of natural language semantics and in particular lexical semantics. In our analysis and implementation, the actions are highly abstract state-changing functions. To relate natural language to such abstract actions would require that every natural language action be \textit{decomposed} into a state-changing function in which literals specify the preconditions and postconditions. To our knowledge, such a decomposition into literals has not been carried out, though there are efforts in that direction [150]. On the other hand, there is a substantive body of research in lexical semantics in which lexical semantic relations could provide the crucial notions of \textit{antonym} and \textit{hyponym} which the analysis and implementation require.

Along the lines of natural language analysis, chapter 2 reflects only a beginning of research into the interactions of the deontic operators and a range of linguistic phenomena. For instance, we did not discuss in depth the interactions between the temporal operators and the deontic operators, which is itself a significant research topic. Even more central to the concerns of logicians are the interactions between the deontic operators and some notion of implication, which we barely touched on. As indicated by the literature ([1] and [67]), there are many very important and as yet unresolved issues relating to the natural language semantics of implication; the interactions with modals such as the deontic operators adds yet another layer of complexity. As we have pointed out in chapter 2, a formal analysis of contracts in natural language will have to address and resolve these issues which otherwise do not arise in logic or mathematics.

In chapter 3, while we have set out our view of the CTD paradox, we have been highly selective in our comparison and contrast to other approaches and the course of development of the issues in the literature. Rather, we narrowly focussed on key pieces of current literature on the topic. Our justification is that a range of issues were, in our view, conflated in the literature, so some simplification is in order. Nonetheless, given our point of view, a thorough re-examination of the literature should be carried out to check our analysis against extant works and see if it holds up to scrutiny.

Similar remarks can be made concerning chapter 4, where we consider just some of the approaches to actions in the literature. Moreover, and central to the claims of the thesis,
the fine-grained violation and fulfillment markers should be cross-checked to the extant approaches to see just to what extent it subsumes them or must be further extended to accommodate them.

Finally, the implementation in chapter 5 is but a proof of concept, showing how actions and deontic specifications can be represented and animated along with key notions of contract state change and deontic specification on complex actions. Natural language examples motivate some of the elements of the implementation, but cannot themselves appear in the implementation. While it was not our objective, the implementation cannot process existing contracts. On the other hand, a system in which actions are represented in terms of explicit preconditions and postconditions of literals could, in principle, be processed. So, for example, anything expressed in terms of the actions of a programming language might allow deontic specification and processing with the implementation.

However, the implementation is highly abstract. Its chief achievement is to allow processing of complex deontic concepts in a rather simple, yet extensible, fundamental language. Yet, without examples of exactly how this could be carried out, it remains an exercise in analysis and implementation.

6.4 Future Work

Our discussion of the limitations of the thesis research provides a basis to further explore and develop a range of closely related issues and proposals. Here we touch on some of them.

- A detailed comparison and contrast of the implementation reported here with the implementations discussed under related work. The comparison and contrast ought to consider differences in terms of action formalisms, approaches to negation and action negation, representations of inertia, constraints/rules on actions, relationships between deontic specifications, expressivity, and other aspects.

- Development of a corpus of contracts in natural language to which one applies computational linguistic analysis to determine linguistic patterns found in contracts. To do this, one would gather publically accessible legal contracts and apply a range of textual data-mining and machine learning techniques. For example, the analysis ought to be able to determine the relationship between a deontic specification on an action, the violation it gives rise to, and any related secondary obligations (the promise-breach-remedy structure found in the law).
• Further research on semantic issues such as the interactions of the deontic operators with temporal operators or, more importantly, with implication. The key challenges in this area are to tease apart issues related to alternative contexts, *aktionsart* structure, and quantificational issues. In chapter 2, we discussed *aktionsart*, showing that deadlines are not particular to deontic concepts, but that the *internal temporal* structure must be considered as well. In addition, we discussed the conditional. As [67] points out, the conditional has quantificational aspects: *if* *x*, *then* *y* is interpretable as *in every circumstance where* *x* *holds, y holds in that circumstance*; such an interpretation may interact with the interpretations of the deontic concept. These problems must be very systematically analysed out, but it is unclear just what evidence would firmly establish the distinct semantic interpretations.

• Provide an automated way to generate trails of executions of actions. At this point in the development of the implementation, each action is manually selected and executed with respect to a *Value Specified Context* (VSC). In order to automatically generate trails of executions, we must first identify all the actions which could be executed in the VSC, which are all actions which have as preconditions a subset of the literals given in the *State-of-Affairs* (SOA) of the VSC. From among these available actions, some method must be introduced to choose which of the actions to execute, and each method would examine different issues. For example, a random selection would allow one to investigate the degree to which a given contract can be violated. An alternative method would be to enrich the structure of agents to endow them with a capacity to *reason* about their actions with respect to their goals, preferences, and relationships to other agents. In this approach, the agents would *select* among the available actions those which are more likely to give rise to desirable results. For example, should an agent have as a goal to avoid all violation and to fulfill deontic specifications ascribed to it immediately, then the function to do so would select from among the available actions those which satisfy this goal. Other approaches to selection of actions could be designed.

• Experiment with interactions among agents so as to express the notion that were one agent to violate one obligation, another agent would have an obligation imposed on her. This may also incorporate *jural relations* such as *rights* and *duties*. To implement this, we could allow agents to reason with respect to the deontic specifications of *other* agents as well as the actions those agents have or could execute. In particular, an agent could be enabled with a capacity to introduce or eliminate deontic specifications in order to discourage or encourage certain behaviours on the part of other agents. A simple example would be if one agent wants to block another agent from executing some action with a literal *p* among the preconditions, the
interfering agent executes some action such that wherever \( p \) would have otherwise held, it does not. To address jural relations such as a right, we would introduce into the contract specification that if an agent \( A \) has a right to execute action \( \alpha \), then \( A \) has a permission with respect to \( \alpha \) while all other agents have a prohibition to do any action which prevents the agent from executing \( \alpha \), say by interfering with the preconditions.

- Higher level structures such as roles and organisations as sets of agents with related contract states. The challenges are to define sets of agents relative to deontic specifications, in effect, understanding the role of an agent in terms of what it is permitted, prohibited, and obligated to do within the context of an organisation. Different roles are defined by different sets of deontic specifications. To capture organisational relationships such as power, we would introduce asymmetrical relationships among the agents in roles such that if one agent has power over another agent, this implies that the powerful agent has an asymmetrical capacity to modify the subordinate’s deontic specification; that is, the powerful agent could, for example, add an obligation to the subordinate’s deontic specifications, but the subordinate agent could not change any deontic specification of the powerful agent.

- Enrich the relationships among the deontic specifications. In the implementation, we only suggested how the deontic specifications are related. One line of research would be to model a variety of theories of relationships among the deontic specifications in our framework. Additional functionality would also be introduced to add or remove deontic specifications relative to other deontic specifications, thus modelling implicational relationships.

- To make a real system which would be usable by people, user interfaces would have to be built, for at this current stage of development, the user interface incorporates code and abbreviations. This would require some easier input method, perhaps in natural language, as well as graphical interfaces to represent the results of the computations over time. One substantial challenge is to relate the abstraction of the implementation to “natural” concepts, for the current system cannot represent simple deontic specifications such as \( \text{Bill is obligated to be in his office during his office hours} \); this would require some means to systematically map natural language expressions onto our abstract actions. Another challenge that is easier to meet is to provide a graphical representation of the results of the output. This could be accomplished by having a baseline of time (middle line) wherein executed actions neither violate nor fulfill an obligation. As actions which violate some deontic specification increase, the line of the graph goes up and is coloured red, while the more actions
which fulfill some deontic specification increase, the line of the graph goes down and is coloured blue.

These are but a few of the ways in which the tool could be further developed.
Bibliography


