Regulatory Conformance Checking: Logic and Logical Form

Nikhil Dinesh
University of Pennsylvania, nikhil.dinesh@gmail.com
Regulatory Conformance Checking: Logic and Logical Form

Abstract
We consider the problem of checking whether an organization conforms to a body of regulation. Conformance is studied in a runtime verification setting. The regulation is translated to a logic, from which we synthesize monitors. The monitors are evaluated as the state of an organization evolves over time, raising an alarm if a violation is detected. An important challenge to this approach comes from the fact that regulations are commonly expressed in natural language. The translation to logic is difficult. Our goal is to assist in this translation by: (a) the design of logics that let us formalize regulation one sentence at a time, and (b) the use of natural language processing as an aid in the sentential translation.

There are many features that are needed in a logic, to accommodate a sentential translation of regulation. We study two features, motivated by a case study. First, statements in regulation refer to others for conditions or exceptions. Second, sentences in regulation convey legal concepts, e.g., obligation and permission. Obligations and permissions can be nested to convey concepts, such as, rights. We motivate and design a logic to accomodate these two features of regulatory texts. The common theme is the importance of the notion of saying in such constructs.

We begin by extending linear temporal logic to allow statements to refer to others. Inter-sentential references are expressed via the use of a predicate, called "says", whose interpretation is determined by inferences from laws. The "says" predicate offers a unified analysis of various kinds of inter-sentential references, e.g., priorities of exceptions over rules, and references to definitions or list items.

We then augment the logic with obligation and permission, by considering problems in access control and conformance. Saying and permission are combined using an axiom that permits a principal to speak on behalf of another. The combination yields benefits to both applications. For access control, we overcome the problematic interactions between hand-off and classical reasoning. For conformance, we obtain a characterization of legal power by nesting saying with obligation and permission. A useful fragment of the logic has a polynomial time decision procedure.

Finally, we turn to the use of natural language processing to translate a sentence to logic. We study one component of the translation in a supervised learning setting. Linguistic theories have argued for a level of logical form as a prelude to translating a sentence into logic. Logical form encodes a resolution of scope ambiguities. We define a restricted kind of logical form, called abstract syntax trees (ASTs), based on the logic developed. Guidelines for annotating ASTs are formulated, using a case study of the Food and Drug Administration's Code of Federal Regulations.

We describe experiments on a modest-sized corpus, of about 200 sentences, annotated with ASTs. The main step in computing ASTs is the ordering or ranking of operators. We adapt a learning model for ranking to order operators. Features are designed by studying subproblems, such as, disambiguating between de re and de dicto interpretations. We obtain an F-score of 90.6% on the set of pairwise ordering decisions.

Degree Type
Dissertation

This dissertation is available at ScholarlyCommons: http://repository.upenn.edu/edissertations/295
Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Aravind K. Joshi

Second Advisor
Insup Lee

Keywords
modal logic, access control, conformance, deontic logic, logical form, scope

Subject Categories
Logic and foundations of mathematics | Other Computer Engineering | Semantics and Pragmatics

This dissertation is available at ScholarlyCommons: http://repository.upenn.edu/edissertations/295
REGULATORY CONFORMANCE CHECKING: LOGIC AND LOGICAL FORM

Nikhil Dinesh

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2010

Supervisor of Dissertation

Aravind K. Joshi
Professor, Computer and Information Science

Co-Supervisor

Insup Lee
Professor, Computer and Information Science

Graduate Group Chairperson

Jianbo Shi, Associate Professor, Computer and Information Science

Dissertation Committee:

Claire Cardie, Professor, Computer Science, Cornell University

Steven O. Kimbrough, Professor, Operations and Information Management

Fernando C. N. Pereira, Professor, Computer and Information Science

Oleg Sokolsky, Research Associate Professor, Computer and Information Science
Acknowledgements

My first thanks must go to my two advisors Aravind Joshi and Insup Lee, and committee chair Oleg Sokolsky. They have contributed significantly to all sections of this thesis. Initially, my research progress was slow to non-existent, and I thank them for their patience with all my ideas – the good, the bad, and the ugly. Aravind has helped shaped my thinking about problems at a high level, Insup with attention to detail, and Oleg with that gray area in the middle.

This thesis was shaped by a great committee, in addition to Oleg. Fernando Pereira has been a great source of ideas throughout grad school. A quick conversation with him usually reveals about ten things that I haven’t thought about. I thank Steve Kimbrough for several illuminating discussions about the logical aspects of this thesis, and Claire Cardie for getting me thinking about different levels of evaluation in the computation of logical form.

I was fortunate to collaborate with researchers in different areas at Penn. Being a part of the team that built the Penn Discourse Treebank was a lot of fun. The adjudication sessions, with Eleni Miltsakaki, Alan Lee, Rashmi Prasad, and Bonnie Webber, helped foster my interest in annotation. I thank Jana Beck for helping with the inter-annotator agreement experiments in this thesis, and Mitch Marcus for suggesting self-agreement. And, the members of the real-time systems group for various discussions, especially David Arney, Arvind Easwaran, and Michael May.

This work would not have been possible without the support of David Hislop from the ARO and Paul Jones from the FDA. They introduced us to the problem of
conformance checking, and encouraged us to pursue it over the years.

Frustration in research can only be overcome through sport! Thanks to Michael Kearns for several years of squash, Adam Aviv and Lukasz Abramowicz for tennis sessions, and Alex Kulesza and Jennifer Gillenwater for running lessons.

Despite my need to spend quality time with my computer, I’ve enjoyed hanging out with several fellow students. Barbecues and poker sessions will not be the same without Ameesh Makadia, Ryan McDonald, and Mirko Visontai. John Blitzer and I explored a curious combination of coffee and Mediterranean history, with a liberal dash of linguistics and machine learning. Thanks to Yuan Ding for lectures on the musculature of saber-toothed cats and other animals. And, to Liang Huang and Bei Xiao for their company during dinners. Axel Bernal and I were the last (of our era) left behind, and I’m grateful to him, with some reservations, for making me give up all the food I like. I also thank Anne Bracy, Andrew McGregor, Nick Montfort, Arun Raghavan, Hana Wallach, and Kilian Weinberger, for their friendship.

My friends from the early years of grad school – Abhijeet, Narayanan, Pranav, and Prasanna, and my undergraduate years – Girish, Subhash, and Roopesh, made those periods significantly more enjoyable. And, finishing up would not have been so much fun, if I hadn’t shared it with Mythili. Some sentiments are best expressed with two negations and a counterfactual.

I’m thankful to have a wonderful family in India and the USA. Thanks to the Menons – Govind, Eva, David, Sunitha, and Kavitha, for their hospitality at Thanksgiving. And, to the Sanjays – Nisha, Kamya, Nino, and the man himself, for wonderful Christmases. Space prohibits acknowledging the entire family. But, I would like to thank my maternal and paternal grandparents for all their love and support, and my cousins – Aditya, Ashwin, Menaka, Mridula, Niranjan, and Rajiv.

Finally, I thank my parents, Dinesh and Lata, and my sister, Nandita. A proper acknowledgement of their contributions would be as long as the thesis. So, in the interests of saving half a rain forest, I conclude by dedicating this thesis to them.
We consider the problem of checking whether an organization conforms to a body of regulation. Conformance is studied in a runtime verification setting. The regulation is translated to a logic, from which we synthesize monitors. The monitors are evaluated as the state of an organization evolves over time, raising an alarm if a violation is detected. An important challenge to this approach comes from the fact that regulations are commonly expressed in natural language. The translation to logic is difficult. Our goal is to assist in this translation by: (a) the design of logics that let us formalize regulation one sentence at a time, and (b) the use of natural language processing as an aid in the sentential translation.

There are many features that are needed in a logic, to accommodate a sentential translation of regulation. We study two features, motivated by a case study. First, statements in regulation refer to others for conditions or exceptions. Second, sentences in regulation convey legal concepts, e.g., obligation and permission. Obligations and permissions can be nested to convey concepts, such as, rights. We motivate and design a logic to accommodate these two features of regulatory texts. The common theme is the importance of the notion of *saying* in such constructs.

We begin by extending linear temporal logic to allow statements to refer to others. Inter-sentential references are expressed via the use of a predicate, called *says*, whose interpretation is determined by inferences from laws. The *says* predicate offers a unified analysis of various kinds of inter-sentential references, e.g., priorities of exceptions over rules, and references to definitions or list items.

We then augment the logic with *obligation* and *permission*, by considering problems in access control and conformance. *Saying* and *permission* are combined using an axiom that permits a principal to speak on behalf of another. The combination
yields benefits to both applications. For access control, we overcome the problematic interactions between hand-off and classical reasoning. For conformance, we obtain a characterization of legal power by nesting saying with obligation and permission. A useful fragment of the logic has a polynomial time decision procedure.

Finally, we turn to the use of natural language processing to translate a sentence to logic. We study one component of the translation in a supervised learning setting. Linguistic theories have argued for a level of logical form as a prelude to translating a sentence into logic. Logical form encodes a resolution of scope ambiguities. We define a restricted kind of logical form, called abstract syntax trees (ASTs), based on the logic developed. Guidelines for annotating ASTs are formulated, using a case study of the Food and Drug Administration’s Code of Federal Regulations.

We describe experiments on a modest-sized corpus, of about 200 sentences, annotated with ASTs. The main step in computing ASTs is the ordering or ranking of operators. We adapt a learning model for ranking to order operators. Features are designed by studying subproblems, such as, disambiguating between de re and de dicto interpretations. We obtain an F-score of 90.6% on the set of pairwise ordering decisions.
# Contents

## Acknowledgements

<table>
<thead>
<tr>
<th>1 Introduction</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Kinds of Regulations Studied Here</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Conformance - A Definition</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Conformance at Runtime</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Translating Regulation to Logic</td>
<td>6</td>
</tr>
<tr>
<td>1.4.1 Translating a Sentence to Logic Using NLP</td>
<td>7</td>
</tr>
<tr>
<td>1.5 Contributions and Outline</td>
<td>9</td>
</tr>
<tr>
<td>1.5.1 Logic</td>
<td>10</td>
</tr>
<tr>
<td>1.5.2 Logical Form</td>
<td>12</td>
</tr>
</tbody>
</table>

## 2 References to Laws

<p>| 2.1 Introduction | 13 |
| 2.2 Motivation | 15 |
| 2.2.1 Examples | 15 |
| 2.2.2 Distribution of Lexical Categories | 17 |
| 2.3 A Logic for Referring to Laws | 19 |
| 2.3.1 Predicate-based Linear Temporal Logic (PredLTL) | 19 |
| 2.3.2 References to Other Laws – A Sketch | 24 |
| 2.3.3 Reference Logic (Refl) | 31 |
| 2.4 Axiomatization | 36 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.1</td>
<td>The Acyclic Fragment</td>
<td>36</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Towards a Modal Analysis of Says</td>
<td>38</td>
</tr>
<tr>
<td>2.5</td>
<td>Related Work</td>
<td>40</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Default Rules in REFL</td>
<td>41</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Lists</td>
<td>41</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Priorities</td>
<td>43</td>
</tr>
<tr>
<td>2.6</td>
<td>Conclusions</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>Conformance Checking at Runtime</td>
<td>46</td>
</tr>
<tr>
<td>3.1</td>
<td>Runtime Checking for LTL</td>
<td>46</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Monitor</td>
<td>47</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Three-Valued Monitors</td>
<td>48</td>
</tr>
<tr>
<td>3.1.3</td>
<td>The EAGLE Formalism</td>
<td>49</td>
</tr>
<tr>
<td>3.2</td>
<td>Adapting the EAGLE Calculus for REFL</td>
<td>51</td>
</tr>
<tr>
<td>3.3</td>
<td>Precomputing Satisfiability</td>
<td>55</td>
</tr>
<tr>
<td>3.3.1</td>
<td>The Single Copy Property</td>
<td>55</td>
</tr>
<tr>
<td>3.3.2</td>
<td>The Pre-computation Procedure</td>
<td>58</td>
</tr>
<tr>
<td>3.4</td>
<td>Case Study</td>
<td>63</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Schemas, Predicates, and the SAT Solver</td>
<td>63</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Formalizing the Regulation</td>
<td>65</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Precomputing Satisfiability and Checking Traces</td>
<td>72</td>
</tr>
<tr>
<td>3.5</td>
<td>Conclusions</td>
<td>73</td>
</tr>
<tr>
<td>4</td>
<td>Permission to Speak</td>
<td>75</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>75</td>
</tr>
<tr>
<td>4.2</td>
<td>Permission to Speak</td>
<td>79</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Preliminaries</td>
<td>80</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Representation in Access Control</td>
<td>82</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Power and Nested Permissions</td>
<td>86</td>
</tr>
</tbody>
</table>
5 Annotating and Computing Logical Form

5.1 Introduction ............................................. 123
  5.1.1 Approach ........................................... 124
  5.1.2 Contributions and Outline ....................... 126

5.2 Abstract Syntax Trees ................................. 127
  5.2.1 Terminals .......................................... 128
  5.2.2 Preterminals ....................................... 129
  5.2.3 Internal Nodes ..................................... 129
  5.2.4 Implicit Tokens Revisited ....................... 130
  5.2.5 Notation for Scope Ordering ..................... 130

5.3 Operators ................................................ 131
  5.3.1 Auxiliary Verbs and The Postcondition Marker .... 131
  5.3.2 Determiners – De Re and De Dicto ................. 132
  5.3.3 Clause and Verb Phrase Modifiers ................. 144
  5.3.4 Co-ordinating Conjunctions ...................... 148
5.4 Corpus and Distribution of Operators .................................. 151
5.5 Computing ASTs – Overview ............................................. 154
  5.5.1 Ordering Operators .................................................. 154
  5.5.2 Finding the Scope Preterminal ...................................... 158
5.6 Feature Design ........................................................... 161
  5.6.1 Binary Classifier ..................................................... 162
  5.6.2 De Re vs De Dicto ................................................... 163
  5.6.3 Ordering Sibling and Embedded Operators ...................... 167
5.7 Search ................................................................. 170
  5.7.1 Algorithms .......................................................... 170
  5.7.2 Metrics ............................................................. 174
  5.7.3 Results ............................................................ 175
5.8 Agreement ............................................................. 177
5.9 Related Work .......................................................... 182
  5.9.1 Annotating ASTs .................................................... 182
  5.9.2 Alternatives to ASTs ................................................. 183
  5.9.3 Computing ASTs .................................................... 184
5.10 Conclusions .......................................................... 185

6 Conclusions .............................................................. 186
  6.1 Logic ................................................................. 187
    6.1.1 Other Kinds of Intersentential References ...................... 187
    6.1.2 Excessively Personal Obligations ............................... 187
    6.1.3 Permission to Choose a Value .................................. 188
    6.1.4 Contracts ....................................................... 189
  6.2 Logical Form ........................................................ 190
    6.2.1 Annotation and Computation ................................... 190
    6.2.2 Connecting ASTs and Logic .................................... 190
## A Refl: Proofs

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1 Fixed Point</td>
<td>192</td>
</tr>
<tr>
<td>A.2 Complexity</td>
<td>196</td>
</tr>
</tbody>
</table>

## B Permission to Speak: Proofs

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1 Semantics</td>
<td>199</td>
</tr>
<tr>
<td>B.2 Soundness and Completeness</td>
<td>200</td>
</tr>
<tr>
<td>B.3 Decidability</td>
<td>205</td>
</tr>
<tr>
<td>B.4 Complexity</td>
<td>209</td>
</tr>
<tr>
<td>B.5 Non-interference</td>
<td>216</td>
</tr>
<tr>
<td>B.6 Conformance</td>
<td>219</td>
</tr>
<tr>
<td>B.7 Completeness of the Decision Procedure for Chain Formulas</td>
<td>224</td>
</tr>
</tbody>
</table>

## C Computing ASTs: Proof

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>233</td>
</tr>
</tbody>
</table>
List of Tables

1.1 Hohfeld’s Fundamental Legal Conceptions .................................. 3

2.1 Distribution of anaphoric items in the WSJ and CFR. ..................... 18
2.2 A run and the associated utterances. ........................................... 28

3.1 Types of rules and formulas used in the case study. ....................... 66
3.2 Subtypes of clause and verb phrase modifiers. ............................... 71

4.1 A state and fixed point utterances for the HIPAA example. ................ 106

5.1 Types of Operators. ............................................................. 151
5.2 Subtypes of determiners. ....................................................... 152
5.3 Subtypes of modal auxiliaries. ................................................ 153
5.4 Subtypes of clause and verb phrase modifiers. ............................... 153
5.5 Types of syntactic categories annotated. ..................................... 153
5.6 De Re scope distribution. ....................................................... 164
5.7 De Re scope distribution for determiners. ..................................... 165
5.8 De Re scope distribution for S & VP modifiers. ............................... 165
5.9 De Re vs De Dicto classification. .............................................. 167
5.10 Pairwise ordering of sibling operators. ....................................... 168
5.11 Scope of embedded operators w.r.t. embedding operators................. 169
5.12 Performance of the algorithms in computing the ASTs. ................... 176
5.13 Evaluation on preterminals with two or more operators. ................ 176
# List of Figures

1.1 Conformance checking as runtime verification .......................... 6  
1.2 Procedure for translating a sentence to logic ......................... 8  
1.3 Parse tree and Logical form ........................................... 9  
2.1 Schema definition for donations ...................................... 64  
2.2 Examples of predicate definitions ..................................... 65  
2.3 CFR 610.40 Paragraph (a) and its translation in logic ............... 67  
2.4 CFR 610.40 Paragraph (d) and its translation in logic ............... 68  
2.5 CFR 610.40 Paragraph (a), after the *says* predicate is compiled out. 69  
3.1 Interaction between the components of the logic ..................... 89  
3.2 Axiomatization of saying and permission .............................. 94  
4.1 Example of an abstract syntax tree (AST) ............................. 128  
4.2 Position of auxiliary verbs in an AST ................................ 132  
5.1 AST for (34) ..................................................................... 135  
5.2 Parse tree of an NP with nested PPs ...................................... 138  
5.3 Default AST for NPs with nested PPs .................................. 139  
5.4 Fragment of the AST for (37) ............................................. 140  
5.5 AST for (38) ..................................................................... 142  
5.6 AST for (39) ..................................................................... 143  
5.7 AST for (41) ..................................................................... 146  
5.8 AST for (44) ..................................................................... 149
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.11</td>
<td>Processed parse tree (PPT) for (30).</td>
<td>156</td>
</tr>
<tr>
<td>5.12</td>
<td>AST for (30).</td>
<td>157</td>
</tr>
<tr>
<td>5.13</td>
<td>PPT for (37).</td>
<td>159</td>
</tr>
<tr>
<td>5.14</td>
<td>AST for (37).</td>
<td>159</td>
</tr>
<tr>
<td>5.15</td>
<td>Second PPT for (37).</td>
<td>160</td>
</tr>
<tr>
<td>5.16</td>
<td>Portion of the state space encountered during beam search.</td>
<td>173</td>
</tr>
<tr>
<td>B.1</td>
<td>Partial model satisfying a formula.</td>
<td>212</td>
</tr>
<tr>
<td>B.2</td>
<td>Partial model demonstrating satisfiability.</td>
<td>227</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Regulations, laws, and policies that affect many aspects of our lives are represented predominantly as documents in natural language. For example, the Food and Drug Administration’s Code of Federal Regulations [144] (FDA CFR) governs the operations of American bloodbanks. Bloodbanks are organizations that collect, store, test, and ultimately, ship donations of blood to their intended recipients. The CFR is framed by experts in the field of medicine, and regulates the tests that need to be performed on donations of blood before they are used. Bloodbanks, in turn, maintain records of the donations, donors, and the tests performed, to demonstrate conformance to the CFR. In such safety-critical scenarios, it is desirable to assess formally whether an organization (bloodbank) conforms to the regulation (CFR).

There is a growing interest in using formal methods to assist organizations in complying with regulation in a variety of contexts. Examples include privacy policy [6, 10, 12, 25, 74, 101] and the broader problem of access control [1–3, 33, 50, 51, 93], and business contracts [4, 54, 56, 60, 62, 88]. Assisting an organization in compliance involves a number of tasks related to the notion of a violation. For example, it is of interest to detect or prevent violations, assign blame, and if possible, recover from violations. In this thesis, we focus on conformance checking which involves detecting the presence of violations and assigning blame in the case of a violation.
We begin, in Section 1.1, by describing the kinds of regulations that we examine in this work. Section 1.2 gives an informal definition of conformance in legal terms. We then cast conformance as a runtime verification question (Section 1.3). In Section 1.4, we narrow our focus to the problem of translating regulation to logic. We conclude, in Section 1.5, with a discussion of the contributions and organization of this thesis.

## 1.1 Kinds of Regulations Studied Here

Regulations are used in a wide variety of contexts. Different contexts give rise to differing salient notions, which need to be accommodated by a conformance checking solution. For example, in privacy regulation, we need to reason about delegations of trust, where an individual permits another to access her private information. Business contracts have complex interactions with time, e.g., the delivery of goods within a time frame under different contracts.

While we attempt to identify general problems and solutions, our methods are undoubtedly influenced by the specific regulations that we have examined. We briefly describe the regulations studied here. Broader applicability is an empirical question for further research.

The main regulatory corpus studied here is the Food and Drug Administration’s Code of Federal Regulations [144] (FDA CFR). The CFR is divided into sections that apply to different organizations. Our focus has been on section 610 of the CFR, which applies to bloodbanks.

While the CFR provides many challenges, each section of the CFR usually applies to a single organization. New problems arise when multiple parties are involved. To study these problems, we consider examples from privacy regulation and the broader problem of access control. An example of privacy regulation is the Health Insurance Portability and Accountability Act [143] (HIPAA), which regulates the collection and disclosure of patient health information by health-care providers. HIPAA has a rich vocabulary of rights, which pose many interesting challenges in assessing conformance.
1.2 Conformance - A Definition

Conformance is a legal notion, and hence, it needs to be defined in legal terms. Hohfeld [69], in his seminal work in 1913, identified a set of fundamental legal conceptions, which remain well-accepted to this day. Hohfeld starts by relating the notions in a scheme of “opposites” and “correlatives”, as shown in Table 1.1.\(^1\) We will now define each of the terms informally, starting with duty:

**Definition 1.1** (Duty [69, Page 32]). *A duty or legal obligation is that which one ought or ought not to do.*

Duties are easily understood, e.g., “the duty of a bloodbank to test a donation of blood”. Henceforth, we use the term *obligation* instead of duty, because the former is used in logics for regulation.

Claims are defined in terms of obligations, i.e., when a claim is invaded, an obligation is violated. As an example, a patient has a claim that a hospital notify her of disclosures of her health information. And, the claim is equivalent to an obligation of the hospital to notify her. Next, we define the notion of privilege:

\(^1\)Hohfeld uses the word “right” instead of “claim”, but suggests the latter may be a better term [69, Page 32]. In particular, he observes that the term “right” is used indiscriminately to cover what in a given case may be a privilege, a power or an immunity [69, Page 30]. We believe that this ambiguity persists, as evidenced by the continuing debate in jurisprudence. We will discuss example regulations where “right” is used in different senses.

<table>
<thead>
<tr>
<th>Jural Opposites</th>
<th>Privilege Correlatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duty</td>
<td>No-Claim</td>
</tr>
<tr>
<td>Claim</td>
<td>Power</td>
</tr>
<tr>
<td>Power</td>
<td>Immunity</td>
</tr>
<tr>
<td>Jural</td>
<td>Claim</td>
</tr>
<tr>
<td>Duty</td>
<td>Privilege</td>
</tr>
<tr>
<td>No-Claim</td>
<td>Power</td>
</tr>
<tr>
<td>Liability</td>
<td>Disability</td>
</tr>
</tbody>
</table>

Table 1.1: Hohfeld’s Fundamental Legal Conceptions
Definition 1.2 (Privilege [69, Page 32]). A privilege is the negation of an obligation, i.e., the negation of an obligation having content or tenor precisely opposite to that of the privilege in question.

For example, a hospital has the privilege to disclose patient health information in emergency situations. Henceforth, we use the term permission instead of privilege, again due to its adoption by logics for regulation. It has been observed that this definition of permission is incomplete from a logical perspective (cf. [23]), and we will have reason to revisit it. The concept of “no-claim” is the correlative of permission, e.g., a patient has no claim against disclosures in an emergency. We now define power:

Definition 1.3 (Power [69, Page 44]). A person (or persons) may be said to have the power to effect a particular change in legal relations, if the change in legal relations results from some superadded facts which are under his volitional control.

For example, a company may have the power to create contractual obligations. Liability is the correlative, i.e., a duty that is caused by the exercise of power. Disability and immunity are easily understood using the negation of power and liability, resply.

Which of these terms should factor into a definition of conformance? In this work, we explore the idea that obligation is the only fundamental legal conception. All other conceptions are derived. Conformance is defined as follows:

Definition 1.4 (Conformance). An individual or organization conforms to a body of regulation, if and only if, she satisfies all the obligations that are imposed on her.

This definition of conformance is relatively uncontroversial, if powers, liabilities, and their negations are excluded. Following Ross [130], we will analyse permissions as exceptions to obligations. Claim and no-claim are easily defined using obligation and permission. We analyse powers and disabilities using nested obligations and permissions, e.g., “permitted to require”, “permitted to permit”. While this analysis has some precedent in logic (see Lindahl [95, Part II], and Jones and Sergot [74]),
there are several open problems in theory and practice. We defer a discussion of the advantages and limitations of our approach to Chapter 4. A formalization of liability and immunity is left to future work, because these concepts are not prevalent in the kinds of regulations that we examine.

1.3 Conformance at Runtime

Given our definition of conformance, it is natural to cast it as a problem of verification, i.e., to show that a system (organization) satisfies its specification (obligations). There are a variety of techniques for verification, e.g., theorem proving [20], model checking [34], testing [115], and runtime verification [92].

Our choice of runtime verification is motivated by the kinds of organizations that we examine. Many aspects of a bloodbank’s or hospital’s operations do not involve computers. A complete description of operations (as needed for theorem proving or model checking) has to include a model of human users, which is a research problem in its own right that is well beyond the scope of this work. Given the need to demonstrate conformance, such organizations maintain a record of the operations that they perform. The evolution of records are easily described as a run. We note that if a finite-state model of an organization can be created, the propositional version of the logics developed here can be adapted to work with available model-checkers.

Figure 1.3 gives an overview of the checking process. From the regulation, we obtain a set of atomic symbols, which are used to translate the regulation to logic. Monitors are synthesized from the logic. An organization, such as a bloodbank, generates events, which are changes in the valuations of the atomic symbols. The monitors are evaluated using the events, and output violations, if any. The shaded circles are those which require manual intervention. When a violation is detected, the problem could be in one of three places – (a) the organization’s operations, (b) the logical representation of regulation, or (c) the regulation itself.

There are several challenging aspects with the process in Figure 1.3. A complete
solution is well-beyond the scope of this thesis. In the following section, we narrow our focus to the problem of translating regulation to logic.

### 1.4 Translating Regulation to Logic

An important challenge with the process in Figure 1.3 is the translation of regulation to logic, and it is the central focus of this work. We take the approach of translating regulation to logic *one sentence at a time*. A practical motivation for a sentential translation is to aid in error-diagnosis. When a violation is detected, it is helpful to associate it with a sentence in regulation, and its translation to logic. In Chapter 2, we argue, using examples and lexical statistics, that a sentential translation eases the difficulty in translating regulation to logic.
There are two difficulties in sententially translating regulation to logic. First, the logic needs to be expressive enough to accommodate a sentential translation. One of the goals of this thesis is to design logics to handle frequently occurring constructs in regulation. The second difficulty is independent of the sentential aspect of our approach. Regulatory bodies are large and complex. Manually translating each sentence to logic is a difficult task. A long term goal of this work is to use natural language processing (NLP) techniques to assist in the translation. In the short term, we set ourselves the more modest goal of identifying subproblems on which NLP techniques can be brought to bear. We now describe the common architecture (in linguistics and NLP) for translating sentences to logic, and narrow our focus to a particular component in the translation.

1.4.1 Translating a Sentence to Logic Using NLP

The translation of sentences in natural language to logic (with a correspondence at the phrase or word-level) has been of interest for several years in linguistics (cf. [67]), and more recently in NLP [24, 149, 150]. While there are different (proposed and implemented) architectures for such systems, they typically consist of the components shown in Figure 1.4.1.

The various steps in Figure 1.4.1 are best understood using an example. Let us consider the procedure for translating the sentence (1) to the first-order logic statement in (2):

(1) Everyone ate an apple.

(2) \( \forall x : \text{person}(x) \Rightarrow (\exists y : \text{apple}(y) \land \text{ate}(x, y)) \)

We start with a grammar or ontology which associates words or phrases (in a lexicon) with syntactic information and its translation to logic. Examples of such grammars include the Tree Adjoining Grammar [76], and the Combinatory Categorial Grammar [141]. Given a sentence and the grammar, the first step consists of producing a parse tree which captures notions of constituency. For (1), the parse tree is shown on
Figure 1.2: Procedure for translating a sentence to logic. The computation of parse trees has been explored. We focus on the definition and computation of logical form. The conversion of logical form to logic is left to future work.

Using the grammar, the parse tree is rearranged to form another tree-like structure, which more closely resembles the structure of the logic than natural language. Following May [103], we call this tree the logical form of a sentence, and for (1), the logical form is shown on the right in Figure 1.3. Finally, given the logical form and the translation of each word (in the grammar) are used to compute the translation in logic. The details of this step are not considered in this work, and we refer the reader to [67] for examples.

In this work, we focus on the logical form component of translation. While logical form has been studied extensively in linguistics (cf. [67]), it has not received attention in the corpus-driven NLP approach. Our goal is to study the challenges in defining logical form, in the context of regulatory texts, and the CFR in particular. We discuss our contributions in the following section.
1.5 Contributions and Outline

There are two main directions for research that we explore in this thesis:

- **Logic** – Our goal is to design logics that accommodate a sentential translation of regulation, while preserving the decidability (and practicability) of conformance checking. As with all applied logic design, the engineering problem is to find the appropriate blend of expressive power and tractability.

- **Logical Form** – A challenge in NLP is to identify tasks beyond parsing, on which we can make reasonable progress at the state-of-the-art. Various efforts have been undertaken to annotate and compute aspects of predicate-argument structure [110, 119]. Logical form offers a new direction in which progress can be made, i.e., in the annotation and computation of scope.

In Section 1.5.1, we describe our work on the logical aspects. And, in Section 1.5.2, we describe the logical form aspects.
1.5.1 Logic

We design logics to accommodate two aspects of regulatory texts – (1) references between laws, and (2) the concepts of obligation and permission. The two aspects are related. Conformance (Definition 1.4) is dependent on the satisfaction of obligations. However, obligations refer to permissions for exceptions, indirectly affecting conformance.

At a conceptual level, an underlying theme is the importance of the notion of saying in the formalization of regulatory texts. Laws are introduced via communicative acts, either spoken or written by a regulator. There are several lines of work which suggest that an explicit representation of communication (saying) is useful in logics for regulation:

- Abadi et al. [3] introduced the notion of saying in logics of access control, in order to distinguish between policies introduced by different principals, and express notions of delegation. Subsequent works, for example [2, 16, 50, 51], have expanded on these ideas.

- Jones and Sergot [75], in their formalization of legal powers, use a modality to characterize the facts that are operative according to an institution. Gelati et al. [53] augmented this approach with proclamations, to reason about individuals representing others on statements.

- Kimbrough and colleagues [81–84] initiated the use of logics to reason about speech acts, in the context of business contracts and transactions. For example, it is of interest to tie events to speech acts, such as, the delivery of goods to the promise of delivery.

In this work, we explore a particular sense of saying – speaking via laws. Informally, a regulator introduces a set of (conditional) laws or statements. The regulator says $\varphi$ via a set of laws if we can prove $\varphi$ from the laws. We use this sense of saying to
describe references to laws, and in combination with obligation and permission, to analyse legal powers.

**Referring to Laws:** Sentences in a body of regulation refer to others in a variety of ways. Inter-sentential references are used to accommodate, for example, exceptions to laws, requirements in other regulatory bodies, terms defined in different sections, and even to refer to items in lists and tables. In Chapter 2 (see also Dinesh et al. [44]), we motivate and develop a logic to accommodate inter-sentential references, by focussing on the problem of exceptions to laws.

We relativize the exceptive reasoning to a predicate, called *says*, whose interpretation is determined by inferences from laws. Conditions and exceptions are expressed by laws that reason about what other laws *say* and *do not say*. We build on ideas from Reiter’s default logic [127] and Kripke’s theory of truth [85]. The *says* predicate offers a unified analysis of various kinds of inter-sentential references, e.g., priorities of exceptions over rules, and references to definitions or list items. In prior formalisms, exceptions are not related to other kinds of inter-sentential references. The predicative analysis is shown to have a modal flavour, i.e., it satisfies the properties of a normal modal operator (c.f. [63]). We identify basic axioms for *saying* which we will use in Chapter 4 to formalize concepts of legal power.

**Checking Algorithms:** Chapter 3 (see also Dinesh et al. [42]) presents algorithms for checking conformance at runtime. Runtime verification has been extensively studied in the context of temporal logic (cf. [92]). We extend the rule-based formalism EAGLE [11] to handle inter-sentential references.

Allowing references between sentences aids in the ease of translation of regulation, and in error-diagnosis. However, the evaluation involves (large) satisfiability tests, which can be expensive at runtime. We identify a condition, motivated by a case study of the CFR, under which satisfiability can be pre-computed, and replaced by tests of lower complexity at runtime. We evaluate the algorithms and the convenience of the logic using a prototype checking tool, applied to the CFR Section 610.40.
Permission to Speak: In Chapter 4 (see also Dinesh et al. [45]), we augment the logic with obligation and permission, by considering problems in access control and conformance. Saying and permission are combined using an axiom that permits a principal to speak on behalf of another. The combination yields benefits to both access control and conformance. For access control, we overcome the problematic interactions between hand-off and classical reasoning. For conformance, we obtain a characterization of legal power by nesting saying with obligation and permission.

The axioms result in a decidable logic, and we characterize its complexity. We integrate the axioms with the logic for reasoning about exceptions, giving says a modal aspect and a non-monotonic aspect. Conformance checking, in the presence of nested obligations and permissions, is shown to be decidable. We prove a non-interference property of the logic, to demonstrate that the axioms do not introduce unwarranted dependencies between statements. We also show that a useful fragment of the logic has a polynomial time decision procedure.

1.5.2 Logical Form

In Chapter 5, we turn to the annotation and computation of logical form. We define a restricted kind of logical form, called abstract syntax trees (ASTs), based on the logic developed. ASTs divide a sentence into preconditions, which correspond to facts, and postconditions, which correspond to obligations and permissions. We present guidelines for annotating ASTs, based on a case study of the Food and Drug Administration’s Code of Federal Regulations (FDA CFR).

We describe experiments on a modest-sized corpus, of about 200 sentences, annotated with ASTs. The main step in the computation of ASTs is the ordering or ranking of operators. We adapt a learning model for ranking to order operators. Features are designed by studying subproblems, such as, disambiguating between de re and de dicto interpretations. We obtain an F-score of 90.6% on the set of pairwise ordering decisions.
Chapter 2

References to Laws

2.1 Introduction

Sentences in a body of regulation refer to others in a variety of ways. Inter-sentential references are used to accommodate, for example, exceptions to laws, requirements in other regulatory bodies, terms defined in different sections, and even to refer to items in lists and tables. We motivate and develop a logic to accommodate inter-sentential references, by focussing on the problem of exceptions to laws.

The design of logics for exceptions has been of interest for several years [5, 106, 134], and is related to the broader area of non-monotonic reasoning [97, 113, 117, 126]. Formalisms that have been applied to regulation are commonly based on default logic [26, 134], logic programming [105, 136], and defeasible logic [56, 60]. Such logics use two mechanisms to analyse exceptions – (a) consistency checks on rules, and (b) priorities to break ties in the case of conflicting rules. The priorities are commonly described by specifying a partial order over the rules in the meta-logic. The use of a meta-language makes it difficult to describe the interaction between reasoning about exceptions, and the other operators in the logic. In addition, due to the separation between consistency checks and priorities, exceptions are not related to other kinds of inter-sentential references.
We generalize the mechanisms for exceptions via the use of a location sensitive provability predicate, which we call \textit{says}. Informally, the idea is that a regulator communicates via laws, and to apply the laws in a given situation, we need to reason about what the regulator \textit{says via her laws}. References between laws are expressed by laws that reason about what the regulator \textit{says} via other laws. The motivation for this analysis is as follows:

1. The \textit{says} predicate offers a unified analysis of various kinds of inter-sentential references, e.g., priorities of exceptions over rules, and references to definitions or list items. In prior formalisms, due to the separation between consistency checks and priorities, exceptions are not related to other kinds of references.

2. At a conceptual level, we believe that the notion of \textit{saying via laws} has broad applicability. In this chapter, to formalize references between laws, we use tests of the form “Does the regulator \textit{say} \phi via a particular set of laws”. In Chapter 4, we combine this notion of \textit{saying} with \textit{obligation} and \textit{permission}, in order to formalize concepts of power that arise in privacy regulation.

This chapter is organized as follows. In Section 2.2, we argue that a logic to represent regulation should provide a mechanism for statements to refer to others. We discuss how sentences from the FDA CFR can be represented in a logic without references, and conclude that this would make the translation to logic difficult. We then compare and contrast the distribution of some lexical categories in the CFR with the newspaper text, which lets us conclude that intersentential references are the predominant way of relating sentences in the CFR.

We then turn to the design of the logic. In Section 2.3.1, we begin by defining a trace or run-based representation for the operations of an organization, and a predicate-based linear temporal logic (PredLTL) to make assertions about runs. PredLTL is extended to express two kinds of normative statements (obligations and permissions), but does not accommodate references between laws. Conformance is de-
fined as the satisfaction of obligations, and at this stage, permissions are not relevant to conformance.

In Sections 2.3.2 and 2.3.3, we extend PredLTL to allow references between laws thereby making permissions relevant to conformance. Specifically, we introduce a predicate, called \textit{says}, whose interpretation is determined by inferences from laws. The justifications in default logic [127] can be cast as an instance of this predicate. Statements are evaluated using the fixed points of an appropriate function, based on a technique used in Kripke’s theory of truth [85]. The complexity of conformance checking is established. Section 2.4 presents an axiomatization of the \textit{says} predicate. The predicative analysis is shown to have a modal flavour, i.e., it satisfies the properties of a normal modal operator (c.f. [63]). We identify basic axioms for \textit{saying} which we will use in Chapter 4 to formalize concepts of legal power.

Section 2.6 concludes with a discussion of constructs that we have not yet formalized.

2.2 Motivation

In this section, we argue that a logic to represent regulation should provide a mechanism for sentences to refer to others. The discussion is divided into two parts. In Section 2.2.1, we discuss examples of the phenomena that we are interested in and how they may be represented in a logic with no mechanism for sentences to refer to others. We then contrast the distribution of some lexical categories in the CFR with newspaper text, which suggest that references to sentences are an important way of expressing relationships between sentences in regulation (Section 2.2.2).

2.2.1 Examples

The examples in this section are shortened versions of sentences from the CFR Section 610.40, which we will use through the course of this chapter. Consider the following
(3) Except as specified in (4), every donation of blood or blood component must be tested for evidence of infection due to Hepatitis B.

(4) You are not required to test donations of source plasma for evidence of infection due to Hepatitis B.

Statement (3) conveys an obligation to test donations of blood or blood component for Hepatitis B, and (4) conveys a permission not to test a donation of source plasma (a blood component) for Hepatitis B. To assess an organization’s conformance to (3) and (4), it suffices to check whether “All non-source plasma donations are tested for Hepatitis B”. In other words, (3) and (4) imply the following obligation:

(5) Every non-source plasma donation must be tested for evidence of infection due to Hepatitis B.

There are a variety of logics in which one can capture the interpretation of (5), as needed for conformance. Now suppose we have a sentence that refers to (3):

(6) To test for Hepatitis B, you must use a screening test kit.

The reference is more indirect here, but the interpretation is: “If (3) requires a test, then the test must be performed using a screening test kit”. A bloodbank is not prevented from using a different kind of test for source plasma donations. (6) can be represented by first producing (5), and then inferring that (5) and (6) imply the following:

(7) Every non-source plasma donation must be tested for evidence of infection due to Hepatitis B using a screening test kit.

It is easy to represent the interpretation of (7) directly in a logic. However, (7) has a complex relationship to the sentences from which it was derived, i.e., (3), (4) and (6). The derivation takes the form of a tree:
To summarize, if one wishes to use a logic with no support for referring to other sentences, derived obligations must be created manually. We argue that the manual creation of derived obligations is impractical in terms of the effort involved. The full version of statement (3) in the CFR contains six exceptions, and these exceptions in turn have statements that qualify them further. It is difficult to inspect a derived obligation, and determine if it captures the intended interpretation of the sentences from which it came. We discuss a realistic example in the context of a case study in Chapter 3 (see Figure 3.5).

In the following section, we provide a more quantitative motivation for handling intersentential references.

### 2.2.2 Distribution of Lexical Categories

In the previous section, we saw several examples of how sentences in regulation refer to others. Natural language offers a variety of devices to relate sentences to others. A large class of such devices fall under the rubric of *anaphora*, which is a means of linking a sentence to the prior discourse. Common examples of such anaphoric items are pronouns and adverbial connectives, e.g., however, instead, furthermore, etc.\(^1\)

Table 2.1 contrasts the distribution of potentially anaphoric items in the Wall Street Journal (WSJ) corpus, with the CFR. The first three rows show counts of pronouns, and the CFR has a markedly lower number of pronouns than the WSJ. The next two rows show counts of adverbial connectives. ADV1 comprises of the connectives *also, however, in addition, otherwise, for example, therefore, previously, later, earlier, until* and *still*. These connectives have specialized uses in the CFR and

\(^1\)Not all uses of pronouns are anaphoric. Some pronouns are bound by quantifiers, e.g., *every one loves their mother*. We report counts based on occurrence of strings and do not distinguish between different uses.
Table 2.1: Differences in the distribution of some anaphoric lexical items in the Wall Street Journal (WSJ) corpus and the CFR. Both the WSJ and the CFR have approximately 1M words.

tend to be quite frequent, with otherwise being the most frequent in the CFR (517 cases). ADV2 is a set of 48 adverbial connectives annotated by the Penn Discourse Treebank [110] excluding those in ADV1, e.g., instead, as a result, nevertheless. The connectives in ADV2 are significantly more frequent in the WSJ than in the CFR.

The last two rows in Table 2.1 show two common ways of establishing relationships between sentences in the CFR. The adjective such is a common way of referring to a set discussed in an immediately preceding law, e.g., such tests. The last row counts explicit references to other law, by searching for phrases like this section, or references to section and paragraph identifiers. Of the categories we considered this is by far the most frequent in the CFR. The frequency of intersentential references has also been observed in other bodies of regulation [18, 25].

We conclude that there are compelling qualitative and quantitative reasons to handle intersentential references in a logic for regulation. In the following sections, we develop a logic with a syntax and semantics for references.
2.3 A Logic for Referring to Laws

In this section, we extend linear temporal logic (LTL) to distinguish between obligations and permissions, and allow references between statements. We begin, in Section 2.3.1, by representing a bloodbank as a run or trace. LTL is extended to distinguish between obligations and permissions, leading to definitions of conformance. We then extend the logic to allow sentences to refer to others. Section 2.3.2 presents a simplified version of the semantics which can accommodate some but not all kinds of regulations. We identify the difficult cases and generalize the definitions in Section 2.3.3. We also discuss the complexity of conformance checking with references.

Section 2.3.1 is intended as background, in which we discuss several underlying assumptions. Our goal is to focus on the problem of references, and to treat the representation of obligations and permissions as an important but orthogonal issue (which we will return to in Chapter 4).

2.3.1 Predicate-based Linear Temporal Logic (PredLTL)

Representing regulated operations: Given the need to demonstrate conformance to the regulation in case of an audit, regulated organizations such as bloodbanks keep track of their operations in a database, for example, donor information and the tests they perform. Such a system can be thought of abstractly as a relational structure evolving over time. At each point in time (state), there are a set of objects (such as donations and donors) and relations between the objects (such as an association between a donor and her donations). The state changes by the creation, removal or modification of objects. We represent this as a run.

Definition 2.1 (A Run of a System). Given countable sets \( \Phi_1, ..., \Phi_n \) (where \( \Phi_j \) is a set of predicate names of arity \( j \)) and object names \( O \), a run of a system \( R( \Phi_1, ..., \Phi_n, O) \), abbreviated as \( R \), is a tuple \( (r, \pi_1, ..., \pi_n) \) where:

- \( r : \mathbb{N} \rightarrow S \) is a sequence of states. \( \mathbb{N} \) is the set of natural numbers, and \( S \) is a
set of states.

- \( \pi_j : \Phi_j \times S \to 2^O \) is a truth assignment to predicates of arity \( j \). Given \( p \in \Phi_j \), we will say that \( p(o_1, \ldots, o_j) \) is true at state \( s \) iff \( (o_1, \ldots, o_j) \in \pi_j(p, s) \).

Given a run \( R \) and a time \( i \in \mathbb{N} \), the pair \( (R, i) \) is called a point (statements in linear temporal logic are evaluated at points). Given the predicate names \( (\Phi_1, \ldots, \Phi_n) \) and object names \( O \), the corresponding space of runs is denoted by \( \mathcal{R}(\Phi_1, \ldots, \Phi_n, O) \), abbreviated as \( \mathcal{R} \).

**Representing the regulation:** The logic that we define in this section is a restricted fragment of first-order modal logic. The restriction is that we allow formulas with free variables, but no quantification over objects. Formulas will be interpreted using the universal generalization rule, i.e., over all assignments to free variables. Quantifiers are omitted because we will need to use provability tests in a fragment of the language here, in order to formalize references. Unrestricted quantification would make these tests undecidable. At the same time, the lack of quantification here can be too restrictive, if we wish to perform a sentential translation of the regulation. We will show, in Section 2.3.3, that when references are added, the logic becomes more expressive than first order logic, and quantifiers can be added in certain places without affecting decidability.

We begin by defining the syntax:

**Definition 2.2 (Syntax).** Given countable sets \( \Phi_1, \ldots, \Phi_n \) (of predicate names), object names \( O \), and a set of variables \( X \), the language \( L(\Phi_1, \ldots, \Phi_n, O, X) \), abbreviated as \( L \), is the smallest set such that:

- \( p(y_1, \ldots, y_j) \in L \) where \( p \in \Phi_j \) and \( (y_1, \ldots, y_j) \in (X \cup O)^j \).

- If \( \varphi \in L \), then \( \neg \varphi \in L \) and \( \Box \varphi \in L \). If \( \varphi, \psi \in L \), then \( \varphi \land \psi \in L \).

Disjunction \( \varphi \lor \psi = \neg (\neg \varphi \land \neg \psi) \) and implication \( \varphi \Rightarrow \psi = \neg \varphi \lor \psi \) are derived connectives. The temporal operator is understood in the usual way: \( \Box \varphi \) (\( \varphi \) holds and will always hold (globally)). \( \Diamond \varphi \) (\( \varphi \) will eventually hold) is defined as \( \neg \Box \neg \varphi \).
The syntax is extended to express three kinds of statements in a body of regulation:

**Definition 2.3** (Syntax of Regulation). Given a finite set of identifiers \( ID \), a body of regulation \( \text{Reg} \) is a set of statements such that for each \( id \in ID \), there exist \( \varphi, \psi \in L \) such that either: (id) \( \varphi \mapsto \psi \in \text{Reg} \), (id).o: \( \varphi \mapsto \psi \in \text{Reg} \), or (id).p: \( \varphi \mapsto \psi \in \text{Reg} \).

A body of regulation is a finite set of rules, which are understood as follows. (id) \( \varphi \mapsto \psi \) is read as “If \( \varphi \), then the regulatory authority says \( \psi \) via the law labeled (id)”. Such rules are used to represent, for example, institutional facts [73], e.g., If a priest performs a particular ceremony for a couple, then the regulatory authority says that the couple is married (where married is an institutional fact). The normative statement (id).o: \( \varphi \mapsto \psi \) is read as: “If \( \varphi \), then the regulatory authority says \( \psi \) is obligated (permitted) via the law labeled id”. \( \varphi \) is called the precondition of the law, and \( \psi \) is called the postcondition. We use the notation (id).x: \( \varphi \mapsto \psi \) to stand for a generic rule corresponding to institutional facts, obligations, or permissions.

We note that the formal notation doesn’t quite capture the informal reading of normative statements. In particular, the informal interpretation suggests that obligation and permission need to be operators within the postcondition. We make some simplifying assumptions in this chapter (discussed below), in order build up intuition for the analysis of obligation and permission in Chapter 4. The semantics is defined as follows:

**Definition 2.4** (Semantics). Given a run \( R = (r, \pi_1, ..., \pi_n) \), \( \varphi \in L \), and a variable assignment \( v: X \rightarrow O \), the relation \( (R, i, v) \models \varphi \) is defined inductively as follows:

- \( (R, i, v) \models p(y_1, ..., y_j) \) iff \( (o_1, ..., o_j) \in \pi_j(p, r(i)) \) where \( o_k = v(y_k) \) if \( y_k \in O \), and \( o_k = y_k \) otherwise.

- The semantics of conjunction and negation is defined in the usual way.
\( (R, i, v) \models \Box \varphi \) iff for all \( k \geq i : (R, k, v) \models \varphi \)

We extend the semantic relation to regulatory statements. We take \( \models \) to stand for “conforms to”:

\( (R, i, v) \models (id).o : \varphi \mapsto \psi \) iff \( (R, i, v) \models \varphi \Rightarrow \psi \) (\( \Rightarrow \) is implication)

\( (R, i, v) \models (id).p : \varphi \mapsto \psi \). Runs vacuously conform to permissions. Permissions will become relevant when references from obligations are present (Section 2.3.2).

\( (R, i, v) \models (id) \varphi \mapsto \psi \). Institutional facts are also not directly relevant to conformance, and will be used via references from other laws.

Consider again our example from Section 4.2. We use three predicates defined as follows. \( d(x) \) is true iff \( x \) is a donation. \( sp(x) \) is true iff \( x \) consists of source plasma. \( test(x) \) is true iff \( x \) is tested for Hepatitis B. Statement (5) is represented as:

\[(5).o : d(x) \land \neg sp(x) \mapsto \Diamond test(x)\]

Statement (4) is be represented as:

\[(4).p : d(y) \land sp(y) \mapsto \neg \Diamond test(y)\]

However, statement (3) cannot be represented directly.

We will now define conformance, and then discuss the various definitions in the context of related work. Conformance of a run \( R \) is defined using the notion of validity. A formula \( \varphi \) is valid at the point \((R, i), (R, i) \models \varphi \), iff for all variable assignments \( v \): \( (R, i, v) \models \varphi \). A formula \( \varphi \) is valid on \( R \) if it is valid in all points, that is, \( R \models \varphi \) iff for all \( i : (R, i) \models \varphi \).

**Definition 2.5** (Run Conformance). *Given a body of regulation \( \text{Reg} \) and a run \( R \) representing the operations of an organization, we say that \( R \) conforms to the regulation iff for all \((id).x : \varphi \mapsto \psi \in \text{Reg}\), we have \( R \models (id).x : \varphi \mapsto \psi \).*
**Discussion:** The deontic concepts of obligation and permission are treated as properties of sentences. Only obligations matter for conformance. If a non-source plasma donation is not tested, there is a problem. On the other hand, a bloodbank may choose to test a donation of source plasma or not. In assessing conformance, the function of a permission is to serve as an exception to an obligation, and in this indirect manner it becomes relevant. We will give a semantics to this function of permissions in Section 2.3.2. Such a treatment of permissions has its basis in the legal theory of Ross [130].

Ross' approach to permission is by no means the only one. Theories have distinguished between various kinds of permission (cf. [23]), the most common distinction being that of positive and negative permission. We discuss the analysis by Makinson and van der Torre [99]. \( \varphi \) is said to positively permitted iff it is explicitly permitted by the laws, and \( \varphi \) is negatively permitted iff it is not forbidden. The key issue is whether positive permissions can give rise to violations. In regulations phrased exclusively in terms of permissions, it is desirable to say that if \( \varphi \) denotes a “relevant” condition which is not explicitly permitted, then it should not hold in conforming implementations. While this has been analysed as a property of permission, following Ross, we take such violations as arising from an implicit obligation, i.e., the italicized clause. This implicit obligation can be represented using the techniques we discuss in Section 2.3.2, provided that the relevance of the condition is known.

In this chapter, we treat obligation and permission as top-level operators. Nested deontic constructs [100] cannot be expressed, i.e., sentences of the form “required to allow x” or “allowed to require x.”. Conformance is defined at the level of a run, and as a result, blame cannot be assigned to different individuals. The motivation for these simplifications is to focus on the problem of exceptions, which is taken to be orthogonal to the analysis of obligations and permissions. We will formalize obligation and permission in Chapter 4.
2.3.2 References to Other Laws – A Sketch

In this section, we develop some intuition for reference logic (Refl), which is used to handle references. We describe the syntax and present a simplified version of the semantics, which can accommodate some but not all kinds of regulations. We will then identify the difficult cases and generalize the definitions in Section 2.3.3.

The syntax of PredLTL is extended with an inference predicate says\textsubscript{Id}\(\varphi\), where Id is a set of identifiers. says\textsubscript{Id}\(\varphi\) is read as “the regulatory authority says \(\varphi\) via the laws labeled Id". There are two restrictions: (a) \(\varphi\) is a statement in PredLTL (Definition 2.2) and (b) the predicate says\textsubscript{Id}\(\varphi\) can appear only in preconditions of laws. These restrictions are similar to those that apply to justifications in default logic [127]. In Chapter 4, when we consider nested obligations and permissions, these restrictions will be lifted:

**Definition 2.6 (Syntax of Preconditions).** Given countable sets \(\Phi_1, ..., \Phi_n\) (of predicate names), object names \(O\), and a set of variables \(X\), the language \(L'(\Phi_1, ..., \Phi_n, O, X)\), abbreviated as \(L'\), is the smallest set such that:

- \(p(y_1, ..., y_j) \in L'\) where \(p \in \Phi_j\) and \((y_1, ..., y_j) \in (X \cup O)^j\).
- If \(\varphi \in L'\), then \(\neg \varphi \in L'\) and \(\Box \varphi \in L'\). If \(\varphi, \psi \in L'\), then \(\varphi \land \psi \in L'\)
- If \(Id \subseteq ID\) and \(\varphi \in L(\Phi_1, ..., \Phi_n, O, X)\) (Definition 2.2), then says\textsubscript{Id}\(\varphi \in L'\)

The syntax of regulatory statements (Definition 2.3) is modified so that the preconditions of laws are statements from \(L'\). We use \((id).x : \varphi \leftrightarrow \psi\) to stand for a generic law (either institutional fact, obligation, or permission), where \(\varphi \in L'\) and \(\psi \in L\).

Let us consider how we might evaluate formulas of the form says\textsubscript{Id}\(\varphi\). Informally, we wish to capture the idea that says\textsubscript{Id}\(\varphi\) is true if the regulator says \(\varphi\) via the laws Id. Thus, we need some representation of what the regulator says via her laws. Recall that our informal interpretation of a rule \((id).x : \varphi \leftrightarrow \psi\) is “If \(\varphi\), then the
regulator says $\psi$ via the law labeled (id)”. The postconditions of laws provide a suitable representation of what the regulator says. We now make this idea precise.

We begin by defining the propositionalization of formulas, which is used in subsequent definitions:

**Definition 2.7 (Propositionalization).** Given $\phi \in L$ and an assignment $v \in V$, the propositionalization of $\phi$ w.r.t. $v$, denoted $v(\phi)$, is defined inductively as follows:

- $v(p(y_1,\ldots,y_n)) = p(o_1,\ldots,o_n)$, where $o_i = v(y_i)$ if $y_i \in X$ and $o_i = y_i$ otherwise ($y_i \in O$).
- $v(\phi \land \psi) = v(\phi) \land v(\psi)$, and $v(\neg \phi) = \neg v(\phi)$
- $v(\square \phi) = \Box(v(\phi))$

Given $\phi \in L$ and $v \in V$, we say that $v(\phi)$ is true at a point $(R,i)$, denoted $(R,i) \models v(\phi)$, iff $(R,i,v) \models \phi$. The notation lets us disregard the variable assignment in evaluating propositionalized formulas.

We now define *utterances*, which represent what the regulator can say, i.e., possible statements from the regulator:

**Definition 2.8 (Utterance).** Given a body of regulation $\text{Reg}$ and a variable assignment $v \in V(X,O)$, an utterance is a statement $(id,v(\psi))$ such that $id \in \text{ID}$ and $(id).x : \varphi \leftrightarrow \psi \in \text{Reg}$. The set of utterances is denoted by $U(\text{Reg},V)$, abbreviated $U$.

Given a set of utterances $U \subseteq U$ and $\text{Id} \subseteq \text{ID}$, $U_{\text{Id}}$ is the set such that $\psi \in U_{\text{Id}}$ iff $(id,\psi) \in U$ and $id \in \text{Id}$.

An utterance $(id,v(\psi))$ is read as “the regulator says $\psi$ via the law $id$”. Given a set of utterances $U$, the set $U_{\text{Id}}$ is understood as what the regulator says via her the labeled $\text{Id}$. As we mentioned, a set of utterances represent possible statements from a regulator. How do we determine what the regulator actually says? Intuitively, we need to obtain utterances from laws with true preconditions. However, we need an appropriate set of utterances to evaluate $\text{says}_{\text{Id}} \phi$ in the preconditions of laws. As
a result, there is a circular dependency between the appropriate set of utterances and the evaluation of preconditions. To address this circularity, let us assume the existence of a relation $|=1$, such that $(R, i, \text{Reg}, v) |=_1 \varphi$, read as “$\varphi$ is true at the point $(R, i)$ w.r.t. the regulation Reg and assignment $v$”. We can now define properties of utterances to determine what a regulator actually says:

**Definition 2.9**: (Sound and Complete Utterances). Given a body of regulation $\text{Reg}$, a set of utterances $U \subseteq \mathcal{U}$ is **sound** w.r.t. a point $(R, i)$ if:

**US** If $(id, \phi) \in U$, then there exists $(id) \varphi \mapsto \psi \in \text{Reg}$ and $v \in V$ such that $\phi = v(\psi)$ and $(R, i, \text{Reg}, v) |=_1 \varphi$

Similarly, a set of utterances $U$ is said to be **complete** w.r.t. a point $(R, i)$ if:

**UC** If there exists $(id) \varphi \mapsto \psi \in \text{Reg}$ and $v \in V$ such that $(R, i, \text{Reg}, v) |=_1 \varphi$, then $(id, v(\psi)) \in U$

Informally, a set of utterances is sound if all the utterances in it come from laws with true preconditions. A set of utterances is complete if the converse holds, i.e., all the laws with true preconditions have a corresponding utterance. Given a sound set of utterances $U$ and an assignment $v$, we will say that $\text{says}_{\text{id}} \phi$ is true w.r.t. $U$ and $v$ iff $U_{\text{id}}$ entails $v(\phi)$. We now define this notion of entailment:

**Definition 2.10**: (Entailment). Given a set of propositional LTL formulas $\Delta$, and a propositional formula $\psi$, we say that $\Delta$ entails $\psi$, denoted $\Delta |= \psi$, iff for all runs $R \in \mathcal{R}$ and times $i \in N$, if $(R, i) |= \varphi$ for all $\varphi \in \Delta$, then $(R, i) |= \psi$.

Note that $\Delta |= \psi$ involves a validity test in propositional LTL, which can be decided in space polynomial in the size of $\Delta$ and $\psi$ [139]. We remind the reader that formulas of the form $\text{says}_{\text{id}} \phi$ appear only in preconditions of laws, and are restricted so that $\phi$ is a statement in $\text{PredLTL}$. Crucially, these restrictions allow us to use validity tests in LTL. If we allowed either nested occurrences of $\text{says}$ or $\text{says}$ to appear in postconditions of laws, we would need an underlying notion of validity for $\text{saying}$.
In this chapter, we use the syntactic restriction to identify basic properties of \textit{says}, and we will use these properties to design a logic of \textit{saying} in Chapter 4.

We are now left with the task of defining the relation \( \models \). We discuss a candidate definition, which can accommodate some but not all kinds of regulations. We will then identify the difficult cases and generalize the definitions. Let us assume as given a run \( R \), a time \( i \in N \), a body of regulation \( \text{Reg} \) and an assignment \( v \in V \). We wish to determine whether the precondition of a law \( \varphi \in L' \) is “true” w.r.t. \( (R, i), \text{Reg} \) and \( v \). The relation \( (R, i, \text{Reg}, v) \models \varphi \) defined inductively as follows:

\begin{align*}
\text{P1} & \quad (R, i, \text{Reg}, v) \models_1 p(y_1, \ldots, y_j) \iff (o_1, \ldots, o_j) \in I_{\Phi_j}(p), \text{ where } o_i = v(y_i) \text{ if } y_i \in X \text{ and } o_i = y_i \text{ otherwise}. \\
\text{P2} & \quad \text{Conjunction, negation, and the temporal operator are handled as usual.} \\
\text{P3} & \quad (R, i, \text{Reg}, v) \models_1 \text{says}_{id} \phi \iff \text{there exists set of utterances } U \subseteq U \text{ such that } U \text{ is sound and } U_{id} \models v(\phi) \\
\text{P4} & \quad \models_1 \text{ is extended to statements in } \text{Reg} \text{ (again with the interpretation that } \models_1 \text{ stands for conforms to):} \\
& \quad (R, i, \text{Reg}, v) \models_1 (id).o : \varphi \rightarrow \psi \iff (R, i, \text{Reg}, v) \models_1 \varphi \Rightarrow \psi \\
& \quad (R, i, \text{Reg}, v) \models_1 (id).p : \varphi \rightarrow \psi \text{ and } (R, i, \text{Reg}, v) \models_1 (id) \varphi \rightarrow \psi - \text{Runs vacuously conform to permissions and institutional facts.}
\end{align*}

We now discuss two examples to illustrate the various definitions. The first example describes a case where the definitions suffice. We then consider an example where the definitions fail, and we will use it to generalize the definitions in Section 2.3.3.

**Example 1:** Consider again our example from the CFR:

(3) Except as specified in (4), every donation of blood or blood component must be tested for evidence of infection due to Hepatitis B.

(4) You are not required to test donations of source plasma for evidence of infection due to Hepatitis B.
(3) and (4) are represented in REFL as follows:

- (3). ∙: \( d(x) \land \neg \text{says}_4(\neg \Diamond test(x)) \leftrightarrow \Diamond test(x) \), and

- (4). ∙: \( d(y) \land sp(y) \leftrightarrow \neg \Diamond test(y) \)

In the obligation above, the subformula \( \text{says}_4(\neg \Diamond test(x)) \) is read as “The FDA says, via law (4), that \( x \) is not tested eventually”. The obligation is read as: “If \( x \) is a donation \( (d(x)) \) and the FDA does not say, via law (4), that \( x \) is not tested eventually, then the FDA says, via law (3), that \( x \) is tested eventually”.

<table>
<thead>
<tr>
<th>Time</th>
<th>Objects</th>
<th>Predicates</th>
<th>Utterances</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( o_1 )</td>
<td>( d(o_1), sp(o_1), \neg test(o_1) )</td>
<td>( (4, \neg \Diamond test(o_1)) )</td>
</tr>
<tr>
<td>2</td>
<td>( o_1 )</td>
<td>( d(o_1), sp(o_1), \neg test(o_1) )</td>
<td>( (4, \neg \Diamond test(o_1)) )</td>
</tr>
<tr>
<td></td>
<td>( o_2 )</td>
<td>( d(o_2), \neg sp(o_2), \neg test(o_2) )</td>
<td>( (3, \Diamond test(o_2)) )</td>
</tr>
<tr>
<td>3</td>
<td>( o_1 )</td>
<td>( d(o_1), sp(o_1), test(o_1) )</td>
<td>( (4, \neg \Diamond test(o_1)) )</td>
</tr>
<tr>
<td></td>
<td>( o_2 )</td>
<td>( d(o_2), \neg sp(o_2), \neg test(o_2) )</td>
<td>( (3, \Diamond test(o_2)) )</td>
</tr>
</tbody>
</table>

Table 2.2: A run and the associated utterances.

Table 2.2 shows a run of a bloodbank, where each point is associated with a set of utterances. First, an object \( o_1 \) is enters the system. \( o_1 \) is a donation of source plasma \( (d(o_1) \) and \( sp(o_1) \) are true). When a donation is added, its test predicate is initially false. Then, an object \( o_2 \) is added, which is a donation but not of source plasma. In the third step, the object \( o_1 \) is tested. At this point, unless the run is extended to test \( o_2 \) as well, it does not conform with the regulation.

The set of utterances associated with each point in Table 2.2 is the unique set of sound and complete utterances for the point. We will prove this below. At the times 2 and 3, there is an utterance from law (3) which requires a test for the non-source plasma donation \( o_2 \) \( ((3, \Diamond test(o_2))) \). And, at the times 1 to 3, there is an utterance from law (4) which permits the source plasma donations \( o_1 \) not to be tested \( ((4, \neg \Diamond test(o_1))) \). The bloodbank conforms if the utterances obtained from (3) are
true, and in this case, the bloodbank does not conform, since the donation \( o_1 \) is not tested. We now establish a property of the example sentences, to illustrate why the definition of \( \models_1 \) works for this example:

**Proposition 2.1.** Given a body of regulation \( \text{Reg} \) that consists of the following two statements:

\[
\begin{align*}
(3).o : d(x) \land \neg \text{say}_{\{4\}}(\neg \Diamond test(x)) & \rightarrow \Diamond test(x) \\
(4).p : d(y) \land sp(y) & \rightarrow \neg \Diamond test(y)
\end{align*}
\]

For all runs \( R \in \mathcal{R} \) and times \( i \in N \), there is a unique set of utterances \( U^{(R,i)} \) such that \( U^{(R,i)} \) is sound and complete w.r.t. \((R, i)\).

**Proof sketch.** Given a point \((R, i)\), let \( U^{(R,i)} \) be the set such that for all \( v \in V \):

- \((3), v(\Diamond test(x))) \in U^{(R,i)}\) iff \((R, i, v) \models_1 d(x) \land \neg sp(x)\).
- \((4), v(\neg \Diamond test(y))) \in U^{(R,i)}\) iff \((R, i, v) \models_1 d(y) \land sp(y)\).

Informally, \( U^{(R,i)} \) contains utterances from law (3) requiring a test for non-source plasma donations, and from law (4) permitting source plasma donations not to be tested. The utterances associated with each point in Table 2.2 corresponds to \( U^{(R,i)} \).

It is easy to show, using \( \text{P1-P4} \), that \( U^{(R,i)} \) is the unique set of sound and complete utterances. We leave the proof as an exercise to the reader.

A question arises with our formalization of (3). We only require that a test be performed eventually, i.e., the postcondition is \( \Diamond test(x) \). However, common sense tells us that donations must be tested before they are shipped to or used by their ultimate recipients. This intuition can be formalized by a separate law:

\[
(8) \text{Donations that are required to be tested must not be released or shipped prior to the completion of testing}
\]

- \((8).o : d(x) \land \text{say}_{\{3\}}(\Diamond test(x)) \land \neg test(x) \rightarrow \neg ship(x)\)
The formalized version of (8) is read as follows. If \( x \) is a donations \((d(x))\) that is required to be tested \((\text{says}_{(3)}(\diamond test(x)))\), and it has not been tested \((\neg test(x))\), then the regulator says that it must not be shipped \((\neg ship(x))\).

Although there is no explicit law equivalent to (8) in CFR Section 610.40, there are laws about the shipping of donations:

(9) Human blood or blood components that are required to be tested...may be released or shipped prior to completion of testing in the following circumstances...

Sentence (9) conveys permissions to ship donations prior to testing, in some exceptional circumstances. And, the formal version of (8) needs to be modified to take these into account.

In general, there may be implicit obligations that can be inferred based on common sense or domain knowledge. Such obligations need to be explicitly formalized. We believe that ReFL provides a convenient way to separately specify such implicit obligations, by referring to the explicit ones. For example, the formal representation of the implicit obligation (8) refers to the explicit obligation (3).

**Example 2:** As we mentioned, the definition of \( \models_1 \) is not appropriate for all kinds of regulations. The key feature of the example above is that the references between the laws in *acyclic*, i.e., law (3) refers to (4) via the use of \( \text{says}_{(4)} \neg \diamond test(x) \) in its precondition. When there are circular references, we can potentially obtain multiple sets of sound and complete utterances and a choice needs to be made. However, a more serious problem arises when negation is involved with circular references:

**Proposition 2.2.** There is no relation \( \models_1 \) that satisfies the properties P1-P4

**Proof.** Consider a regulation Reg that consists of the statement:

\[
(id).o : \neg \text{says}_{(id)} p(x) \leftrightarrow p(x)
\]

In other words, “If the regulator does not say \( p(x) \) via law (id), then the regulator says \( p(x) \) via law (id)”’. The self-referential nature of this sentence coupled with negation
leads to the problem. Kripke [85] discusses several similar examples.

Suppose \((R, i, \text{Reg}, v) \models_1 \text{says}_{\text{id}} p(x)\). By **P3**, there exists a set \(U\) which is sound w.r.t. \((R, i)\) and \(U_{\text{id}} \models v(p(x))\). However, since \(U\) is sound, \((R, i, \text{Reg}, v) \models_1 \neg \text{says}_{\text{id}} p(x)\), giving us a contradiction.

Suppose \((R, i, \text{Reg}, v) \not\models_1 \text{says}_{\text{id}} p(x)\). Then, \((R, i, \text{Reg}, v) \models \neg \text{says}_{\text{id}} p(x)\). The set \(U = \{(\text{id}, v(p(x)))\}\) is sound w.r.t. \((R, i)\) and \(U_{\text{id}} \models v(p(x))\). So, by **P3**, \((R, i, \text{Reg}, v) \models_1 \text{says}_{\text{id}} p(x)\), giving us a contradiction.

\[\square\]

### 2.3.3 Reference Logic (Refl)

The semantic evaluation outlined in Section 2.3.2 works only when the references are acyclic, as we saw in the proof of Proposition 2.2. To handle circular statements, we use a technique from Kripke’s theory of truth [85], which also forms the basis for the Kripke-Kleene-Fitting semantics of logic programs [49]. There are two pieces of machinery needed. First, we move to a three-valued logic, where the third (middle) value stands for ungrounded. The values are denoted by \(B^3 = \{\top, ?, \bot\}\). Second, we will evaluate sentences w.r.t. a pair of sets of utterances associated with each point. Let \((U, U')\) be a pair associated with the point \((R, i)\). Informally, \(U\) will be the set of utterances obtained from laws with true preconditions, while \(U'\) will be the set of utterances from laws with true or ungrounded preconditions \((U \subseteq U')\). The truth of \(\text{says}_{\text{Id}} \varphi\) will determined using \(U\) as before \((U_{\text{Id}} \models v(\varphi))\), but falsity is determined using \(U'\) \((U'_{\text{Id}} \not\models v(\varphi))\). We begin by defining utterance sequences:

**Definition 2.11 (Utterance Sequence).** Given a body of regulation \(\text{Reg}\) and a set of variable assignments \(V\), an utterance sequence is a function \(u : N \rightarrow 2^{U(\text{Reg}, V)}\), i.e., \(u(i) \subseteq U(\text{Reg}, V)\) is a set of utterances. We use \(u(i)_{\text{Id}}\) to denote the set of formulas \(\psi\) such that \((\text{id}, \psi) \in u(i)\) and \(\text{id} \in \text{Id}\).

The space of utterance sequences is denoted by \(\Upsilon(\text{Reg}, V)\), abbreviated \(\Upsilon\).

We begin by defining an assignment of truth values to formulas w.r.t. a point \((R, i)\), a variable assignment \(v\) and an pair of utterance sequences \((u, u')\):
Definition 2.12 (Truth Value Assignment). Given a body of regulation \( \text{Reg} \), a set of assignments \( V \), and a pair of utterance sequences \((u, u') \in \Upsilon \times \Upsilon\), the function \( \eta_{(u, u')}: L^+ \times R \times N \times V \rightarrow B^3 \) is defined as follows:

- \( \eta_{(u, u')}(p(x_1, ..., x_j), R, i, v) = \top \) if \( (v(x_1), ..., v(x_j)) \in \pi_j(p, r(i)) \)
- \( \eta_{(u, u')}(p(x_1, ..., x_j), R, i, v) = \bot \) otherwise.
- Conjunction and negation are handled using the Kleene semantics.
- \( \eta_{(u, u')}([\varphi], R, i, v) = \top \) if for all \( j \geq i \), \( \eta_{(u, u')}(\varphi, R, i, v) = \top \)
- \( \eta_{(u, u')}([\varphi], R, i, v) = \bot \) if there exists \( j \geq i \), \( \eta_{(u, u')}(\varphi, R, i, v) = \bot \)
- \( \eta_{(u, u')}([\varphi], R, i, v) = ? \) otherwise.
- \( \eta_{(u, u')}((\text{says}_{\text{id}} \varphi, R, i, v) = \top \) if \( u(i)_{\text{id}} \models v(\varphi) \)
- \( \eta_{(u, u')}((\text{says}_{\text{id}} \varphi, R, i, v) = \bot \) if \( u'(i)_{\text{id}} \not\models v(\varphi) \)
- \( \eta_{(u, u')}((\text{id}) \circ \varphi \mapsto \psi, R, i, v) = \top \)
- \( \eta_{(u, u')}((\text{id}) \circ \varphi \mapsto \psi, R, i, v) = ? \) otherwise.
- \( \eta_{(u, u')}((\text{id}) \circ \varphi \mapsto \psi, R, i, v) = ? \) otherwise.
- \( \eta_{(u, u')}((\text{id}) \circ \varphi \mapsto \psi, R, i, v) = ? \) otherwise.

Definition 2.12 places no restrictions on the utterance sequences \((u, u')\). However, as we sketched in Section 2.3.2, \( u \) and \( u' \) need to satisfy certain conditions, in order to give an appropriate interpretation to formulas. We now define what it means for a pair of utterance sequences to be sound and complete:

Definition 2.13 (Sound and Complete Utterances). Given a body of regulation \( \text{Reg} \) and a set of assignments \( V \):

A pair \((u, u') \in \Upsilon \times \Upsilon\) is sound w.r.t. a point \((R, i)\) if \( u(i) \subseteq u'(i) \) and:

US1 If \((\text{id}, \phi) \in u(i)\), then there exists \((\text{id}) . x : \varphi \mapsto \psi \in \text{Reg} \) and \( v \in V \) such that \( \phi = v(\psi) \) and \( \eta_{(u, u')}(\varphi, R, i, v) = \top \)
US2 If \((id, \phi) \notin u'(i)\), then for all \((id).x : \varphi \mapsto \psi \in \text{Reg} \) and \(v \in V\) such that 
\(\phi = v(\psi)\) and \(\eta_{(u,u')}(\varphi, R, i, v) = \bot\)

\((u, u')\) is sound w.r.t. \(R\) if it is sound w.r.t. all points \((R, i)\).

Similarly, \((u, u')\) is said to be complete w.r.t. a point \((R, i)\) if \(u(i) \subseteq u'(i)\) and:

UC1 If there exists \((id).x : \varphi \mapsto \psi \in \text{Reg} \) and \(v \in V\) such that 
\(\eta_{(u,u')}(\varphi, R, i, v) = \top\), then \((id, \phi) \in u(i)\).

UC2 If for all \((id).x : \varphi \mapsto \psi \in \text{Reg} \) and \(v \in V\) such that \(\phi = v(\psi)\), we have 
\(\eta_{(u,u')}(\varphi, R, i, v) = \bot\), then \((id, \phi) \notin u'(i)\).

\((u, u')\) is complete w.r.t. \(R\) if it is complete w.r.t. all points \((R, i)\).

A partial order is defined over the space of sound utterance pairs:

**Definition 2.14 (Partial Order).** Given a body of regulation \(\text{Reg}\), a set of assignments \(V\), and the pairs \(\{(u_1, u'_1), (u_2, u'_2)\} \subseteq \Upsilon \times \Upsilon\), we say that \((u_1, u'_1) \leq (u_2, u'_2)\) iff for all \(i \in N\), \(u_1(i) \subseteq u_2(i)\) and \(u'_1(i) \supseteq u'_2(i)\).

Let \(S^R_\Upsilon\) be the set such that \((u, u') \in S^R_\Upsilon\) iff \((u, u') \in \Upsilon \times \Upsilon\) and \((u, u')\) is sound w.r.t. \(R\). The pair \((S^R_\Upsilon, \leq)\) is a partially ordered set (poset).

Finally, we define the function whose fixed points we are interested in.

**Definition 2.15 (Inflationary function).** Given a poset \((S^R_\Upsilon, \leq)\), the function \(I^R_\Upsilon : S^R_\Upsilon \rightarrow S^R_\Upsilon\) is defined as follows. \(I^R_\Upsilon(u_1, u'_1)\) is the pair \((u_2, u'_2) \in S^R_\Upsilon\) such that for all \((id, \phi) \in \mathcal{U}\):

- \((id, \phi) \in u_2(i)\) iff there exists \((id).x : \varphi \mapsto \psi \in \text{Reg} \) and \(v \in V\) such that 
  \(v(\psi) = \phi\) and \(\eta_{(u,u')}(\varphi, R, i, v) = \top\).

- \((id, \phi) \notin u'_2(i)\) iff for all \((id).x : \varphi \mapsto \psi \in \text{Reg} \) and \(v \in V\) such that \(v(\psi) = \phi\) and \(\eta_{(u,u')}(\varphi, R, i, v) = \bot\).

We can show the following:
Theorem 2.1. Given the poset of sound utterance pairs \((S^R_Y, \leq)\) and a function \(\mathcal{I}^R_Y: S^R_Y \rightarrow S^R_Y\) which is inflationary and monotonic, \(\mathcal{I}^R_Y\) has a least fixed point and a maximal fixed point.

We refer the reader to Appendix A.1 for the proof.

Complexity: We mention the upper and lower bounds for the complexity of conformance checking w.r.t. the least fixed point. Given a run \(R\) and regulation \(\text{Reg}\), we say that \(R\) conforms to \(\text{Reg}\), denoted \(R \models \text{Reg}\), iff all obligations are valid in \(R\) at the least fixed point. \(R\) is assumed to be finite in two ways: (a) The set of objects \(O\) is finite, and (b) There exists \(n\), such that for all \(j \geq n\), \(r(n) = r(j)\), i.e., \(R\) eventually reaches a stable state. The following bounds can be established.

Lemma 2.1 (Upper Bound). Given a finite run \(R\) and regulation \(\text{Reg}\), \(R \models \text{Reg}\) can decided in \(\text{EXPSPACE}\) (space exponential in the size of \(\text{Reg}\)).

Lemma 2.2 (Lower Bound). Given a finite run \(R\) and regulation \(\text{Reg}\), \(R \models \text{Reg}\) is hard for \(\text{EXPTIME}\) (time exponential in the size of \(\text{Reg}\)).

The proofs are sketched in Appendix A.2.

Discussion: We now discuss some options in defining conformance, depending on the needs of the application. The sections of the FDA CFR that we have examined can be formalized so that there is a unique fixed point, and conformance is simply the satisfaction of obligations at this fixed point.

However, examples discussed in the literature suggest that it may not be desirable to always have a unique fixed point. A well-known example is that of contrary-to-duty (CTD) obligations [30]. CTD obligations are those that arise when other obligations have been violated. Prakken and Sergot [123] point out an inflexibility in casting CTD structures as an instance of non-monotonic reasoning. We outline how this inflexibility can be avoided, using alternate definitions of conformance. Consider the following example from [98]:

(10) Cottages must not have a fence or a dog.
If a cottage has a dog, it must have both a fence and a warning sign.

The question is what are the obligations when the cottage has a dog. We discuss two possible solutions.

The first solution is to treat the CTD norm (11) as an exception to (10):

\[(10) \cdot o : c(x) \land \neg \text{says}_{2} (f(x) \lor d(x)) \mapsto \neg (f(x) \lor d(x))\]

\[(11) \cdot o : c(y) \land d(y) \mapsto f(y) \land w(y)\]

The predicate \(c(x)\) is read as “x is a cottage”, and \(f(x), d(x)\) and \(w(x)\) correspond to the cottage \(x\) having a fence, dog and warning sign respectively. Let \(o\) be a cottage with a dog, i.e., \(c(o)\) and \(d(o)\) are true. The precondition of (11) is true for the assignment of \(y\) to \(o\), and we obtain the utterance \((11, f(o) \land w(o))\). Since \(f(o) \land w(o) \models f(o) \lor d(o)\), for the assignment of \(x\) to \(o\), \(\text{says}_{2} (f(x) \lor d(x))\) is true, and the precondition of (10) is false. So if \(f(o) \land w(o)\) holds (the cottage has a fence and a warning sign), there is no violation. However, as [123] points out, it may be useful to detect that the situation is worse than the one in which there is no dog.

In the second solution, we treat the laws as exceptions to each other:

\[(10) \cdot o : c(x) \land \neg \text{says}_{2} (f(x) \lor d(x)) \mapsto \neg (f(x) \lor d(x))\]

\[(11) \cdot o : c(y) \land d(y) \land \neg \text{says}_{1} (f(y) \land w(y)) \mapsto f(y) \land w(y)\]

At the least fixed point, both obligations are ungrounded, and we get two maximal fixed points – one with the utterance \((10, \neg (f(o) \lor d(o)))\), and one with the utterance \((11, f(o) \land w(o))\). Since \(d(o)\) holds, there is a violation w.r.t. the former fixed point. In a scenario where there is no dog, a unique fixed point is obtained.

Our analysis of CTD structures achieves the same effect as the analyses in [98, 123]. However, the solution is deficient in two respects. First, from an intuitive perspective, it is not clear why the laws should be exceptions to each other. (11) is an obligation to mitigate a violation of (10). The analysis as exceptions sheds no light on this issue.

Second, from a practical perspective, it is infeasible to enumerate all the fixed points...
as there may be exponentially many in the face of multiple violations. In [98, 123],
the CTD norm is characterized as presupposing the violation of the other, and then
revising the situation. In future work, we plan to investigate predicates that capture
this presuppositional analysis more directly.

2.4 Axiomatization

In Chapter 4, we will extend the predicative analysis of says to accomodate nested
obligations and permissions, by treating says as a modal operator. We provide an
axiomatization for a fragment of RefL in Section 2.4.1, and then, discuss how it can
be extended for a modal treatment of says (Section 2.4.2).

2.4.1 The Acyclic Fragment

As we discussed in the proof of Lemma 2.2, RefL contains first order logic enriched
with a least fixed point predicate. It can be shown that the validity problem is
$\Pi^1_1$-hard, and as a result, it cannot be recursively axiomatized. In this section, we
briefly discuss an axiomatization of the propositional fragment of $L'$ (the language of
preconditions).

We assume as given a fixed finite domain of quantification, and replace variables
by identifiers for domain elements. Given a set of identifiers $ID$, a propositionalized
body of regulation has one or more statements of the form $(id).x : \varphi \mapsto \psi$ for each
$id \in ID$. For example, the presence of $(id).x : \varphi_1 \mapsto \psi_1$ and $(id).x : \varphi_2 \mapsto \psi_2$
corresponds to different assignments to the variables.

To simplify presentation, we will assume that the references in the regulation are
acyclic. This lets us obtain a unique fixed point and restrict attention to a two-valued
logic. We refer the reader to [43] for a discussion of the general case.

We begin with axioms and rules for propositional LTL:

A1 All substitution instances of propositional tautologies
A2 $\Box(\varphi \Rightarrow \psi) \Rightarrow (\Box \varphi \Rightarrow \Box \psi)$

A3 $\Box \varphi \Rightarrow \varphi \land \Box \Box \varphi$

R1 From $\vdash \varphi \Rightarrow \psi$ and $\vdash \varphi$, infer $\vdash \psi$

R2 From $\vdash \varphi$ infer $\vdash \Box \varphi$

We say that $\varphi \in L$ is is provable (denoted $\vdash_L \varphi$) if it is an instance of the axioms A1-A3, or follows from the axioms using the rules R1 and R2. Given a finite set of formulas $\Delta \subseteq L$, we say that $\varphi$ is provable from $\Delta$, denoted $\Delta \vdash_L \varphi$, if $\vdash (\bigwedge \Delta) \Rightarrow \varphi$.

The says predicate is characterized by the laws it refers to. To axiomatize says$_{ld} \varphi$, we need to reason about provability in the language $L$ (propositional LTL). Crucially, we will use the negation of provability in the premise of a rule. Similar mechanisms have been used to axiomatize autoepistemic logics, e.g., Lakemeyer and Levesque [89] use satisfiability in the premise of a rule, and Halpern [63] augments a modal language with an operator for satisfiability.

We begin by developing some notation. Given a set of regulatory statements $F = \{(id_1).x : \varphi_1 \mapsto \psi_1, ..., (id_n).x : \varphi_n \mapsto \psi_n\}$, let $F_{pre} = \{\varphi_1, ..., \varphi_n\}$ be the set of preconditions, $F_{post} = \{\psi_1, ..., \psi_n\}$ be the set of postconditions, and $F_{id} = \{id_1, ..., id_n\}$ be the set of identifiers. Given a finite set of formulas $\Gamma$, we denote the conjunction by $\bigwedge \Gamma$. The conjunction of the empty set is identified with $\top$ (a tautology). We use two rules for the inference predicate:

R3 For all $F \subseteq Reg$ with $F_{id} \subseteq Id$, from $F_{post} \vdash_L \phi$, infer $\vdash \bigwedge F_{pre} \Rightarrow \text{says}_L \phi$

R4 For all $\psi \in L'$, if for all $F \subseteq Reg$ with $F_{id} \subseteq Id$, either $F_{post} \not\vdash_L \phi$, or $\vdash \psi \Rightarrow \neg \bigwedge F_{pre}$, then infer $\vdash \psi \Rightarrow \neg \text{says}_L \phi$.

Informally, R3 says that says$_{ld} \phi$ is true, if there exists a set of laws whose postconditions imply $\phi$, and whose preconditions are true. R4 says that says$_{ld} \phi$ is false, if one of the preconditions is false for all sets of laws whose postconditions imply $\phi$. 

37
In particular, if $F_{post} \not\vDash_L \phi$ for all appropriate subsets, then $\vdash \top \Rightarrow \neg \text{says}_{id} \phi$, and using R1, $\vdash \neg \text{says}_{id} \phi$.

Completeness can be shown by adapting the standard pre-model construction for LTL [94]. We now discuss the general case, i.e., when there are circular references and multiple fixed points. In the presence of multiple fixed points, we can define validity w.r.t. all fixed points, the least fixed point, or maximal fixed points. The axioms and rules discussed here can be adapted to characterize validity w.r.t. all fixed points. We refer the reader to [43] for the proof. Using the fact that $\varphi$ is true (or false) at the least fixed point iff it is true (resply., false) at all fixed points, we can use the axiomatization to reason about the least fixed point. We leave a characterization of validity w.r.t. maximal fixed points to future work.

### 2.4.2 Towards a Modal Analysis of Says

In this chapter, we have analysed says as a predicate, which applies to sentences in LTL. This restriction of the argument of says to LTL sentences is ad-hoc, and we cannot express nested statements, e.g., “The bloodbank says that the FDA says $\varphi$”. In Chapter 4, we will show that nested statements are useful to formalize, for example, notions of rights. A natural way to accomodate nested statements is to treat says as a modal operator. We explore how the predicative definition of says can be extended to a modal definition.

We evaluate the question “Does the FDA say $\varphi$ via its laws”, by checking if $\varphi$ is provable from the utterances obtained from the laws. More formally, $\text{says}_{id} \varphi$ is true w.r.t. the pair of utterance sequences $(u, u')$ at time $i$, if $u(i)_{id} \models \varphi$. The set $u(i)_{id}$ is associated with utterances $(id, \psi)$ with $id \in Id$. The intuitive interpretation of $(id, \psi)$ is “The FDA says $\psi$ in the law labeled id”. This suggests that an utterance $(id, \psi)$ is better represented as $\text{says}_{(id)} \psi$, and that we conduct our deduction over the entire set of utterances. We make this idea precise. Let $\Gamma_i$ be the set such that $\text{says}_{(id)} \psi \in \Gamma_i$ iff $(id, \psi) \in u(i)$. We will consider relations $\vdash'$ which extend the predicative definition
as follows:
\[ \Gamma_i \vdash' \text{says}_{Id} \varphi \iff u(i)_{Id} \models \varphi \quad (\forall \varphi \in L) \]

In other words, the relation \( \vdash' \) agrees with the predicative definition for propositional formulas, but additional items may be provable when there are nested constructions. Halpern [63] adopts a similar criterion for extending an autoepistemic logic to accommodate nesting. Note that there are (infinitely) many relations \( \vdash' \) satisfying the condition above. We will motivate the additional inferences that are needed for nested constructions in Chapter 4.

A relation \( \vdash' \) that extends the predicative definition (in the sense discussed above) has to support inferences of the form \( \{\text{says}_{Id_1} \varphi_1, \ldots, \text{says}_{Id_n} \varphi_n\} \vdash \text{says}_{Id} \varphi \). We now identify two axioms and a rule, which are necessary to characterize these inferences, and show that they are sufficient in Chapter 4 (see Theorem 4.4):

**Proposition 2.3.** The following are provable:

1. \( \vdash \text{says}_{Id}(\varphi \Rightarrow \psi) \Rightarrow (\text{says}_{Id} \varphi \Rightarrow \text{says}_{Id} \psi) \)

2. If \( \vdash_L \varphi \), then \( \vdash \text{says}_{Id} \varphi \).

3. \( \vdash \text{says}_{Id} \varphi \Rightarrow \text{says}_{Id'} \varphi \) for all \( Id \subseteq Id' \)

**Proof.** We begin with some notation to facilitate the proofs. Given \( \phi \in L \), let \( \mathcal{F}_{(Id, \phi)} \) be the set of subsets \((F \subseteq \text{Reg} \text{ with } F_{id} \subseteq Id)\) such that \( F \in \mathcal{F} \) iff \( F_{post} \vdash_L \phi \). Let \( \Delta_{(Id, \phi)} \) be the set such that \( \bigwedge F_{pre} \in \Delta_{(Id, \phi)} \) iff \( F \in \mathcal{F}_{(Id, \phi)} \). It can be show that:

\[ (\ast) \quad \vdash \text{says}_{Id} \phi \Leftrightarrow \bigvee \Delta_{(Id, \phi)} \]

Each direction follows easily from **R3** and **R4**.

For the first item, using (\( \ast \)):

\[ \vdash \text{says}_{Id}(\varphi \Rightarrow \psi) \Rightarrow \bigvee \Delta_{(Id, \varphi \Rightarrow \psi)} \]

\[ \vdash \text{says}_{Id} \varphi \Rightarrow \bigvee \Delta_{(Id, \varphi)} \]
Consider the set $\Delta = \{ \phi \land \phi' | \phi \in \Delta_{(Id,\varphi)} , \phi' \in \Delta_{(Id,\varphi)} \}$. By propositional reasoning, $\vdash \text{says}_{Id}(\varphi \Rightarrow \psi) \Rightarrow (\text{says}_{Id} \varphi \Rightarrow \bigvee \Delta)$ Furthermore, $\Delta \subseteq \Delta_{(Id,\varphi)}$, and so, $\vdash \bigvee \Delta \Rightarrow \bigvee \Delta_{(Id,\varphi)}$. The desired result follows using propositional reasoning.

The second item is immediate from $\textbf{R3}$. For the third item, using (*):

$$
\vdash \text{says}_{Id} \varphi \Rightarrow \bigvee \Delta_{(Id,\varphi)}
$$

Using propositional reasoning, $\vdash \bigvee \Delta_{(Id,\varphi)} \Rightarrow \bigvee \Delta_{(Id',\varphi)}$ for all $Id \subseteq Id'$, since $\Delta_{(Id,\varphi)} \subseteq \Delta_{(Id',\varphi)}$. And, using propositional reasoning, $\vdash \text{says}_{Id} \varphi \Rightarrow \text{says}_{Id'} \varphi$. 

The first two items are analogs of the distribution axiom ($\textbf{A2}$) and the generalization rule ($\textbf{R2}$), which hold for the global temporal modality ($\Box$). An operator satisfying these two properties is known as a normal modal operator (c.f. [63]). The third item gives us monotonicity of says, i.e., if we can deduce $\varphi$ from a set of utterances, then we can deduce it from a larger set of utterances. Note that inferences from regulations are not necessarily monotonic, i.e., if we deduce $\text{says}_{Id} \varphi$ from $\text{Reg}$, it is not necessary that we deduce $\text{says}_{Id} \varphi$ from all $\text{Reg}' \supseteq \text{Reg}$. The monotonicity is w.r.t. utterances.

2.5 Related Work

In this section, we compare ReFL to other non-monotonic formalisms that have been applied to regulation. Examples include those based on default logic [26, 134], logic programming [105, 136], and defeasible logic [56, 60]. We will use default logic to illustrate various issues, but the remarks apply to other systems as well. In Section 2.5.1, we describe how rules in default logic can be translated to ReFL, and consider the interaction of the consistency checks with connectives. Section 2.5.2 describes how references to list items can be expressed in ReFL via the same mechanism that is used to handle exceptions (the says predicate). Default logic needs separate mechanisms
to handle such references. We then compare the use of priorities in default logic to the *says* predicate (Section 2.5.3), and identify some directions for future work.

### 2.5.1 Default Rules in ReFL

The following is an example of a rule in default logic:

\[
\phi : \phi_1, \ldots, \phi_n \quad (id).o \quad \psi
\]

The rule is read as “If \( \varphi \), and \( \phi_i \) is consistent for all \( 1 \leq i \leq n \), then \( \psi \). Default rules can be directly translated to ReFL:

\[
(id).o : \varphi \land \bigwedge_{i=1}^{n} \neg says_{ID} \neg \phi_i \mapsto \psi
\]

\( \phi_i \) is consistent is translated as \( \neg says_{ID} \neg \phi_i \), i.e., the rules do not say \( \neg \phi_i \). In Reiter’s default logic [126], the consistency check is w.r.t. all the rules, and hence, in ReFL, we use the set of all identifiers \( ID \). A limitation of default logic is that it does not allow the mixing of consistency checks with the other connectives in the logic. For example, it is not possible to state the equivalent of \( \varphi \lor \neg says_{ID} \neg \phi \). This limitation is further magnified when we consider the addition of temporal operators. While we can describe exceptions involving an eventuality (\( \neg says_{ID} \neg \diamond \phi \)) in default logic, exceptions that hold eventually (\( \neg \diamond says_{ID} \neg \phi \)) cannot be described directly.

### 2.5.2 Lists

Lists are frequent in the CFR, and are used in a variety of ways. We discuss a simple example here to illustrate how the *says* predicate can be used to preserve the structure of lists. The following is an expanded version of the obligation from the CFR:

\[
(12) \text{Except as specified in (4) every donation of blood or blood component must be tested for evidence of infection due to the following disease agents:}
\]

a. HIV Type 1
(12) conveys an obligation to perform tests for the detection of a list of disease, and the exceptions can apply to tests for any of the diseases. Such lists can be easily represented in ReFL:

\[ (12\cdot o : d(x) \land \text{says}_{12a,12b,12c}(\text{list}(y)) \land \neg \text{says}_{4}(\neg \Diamond \text{test}(x,y)) \rightarrow \Diamond \text{test}(x,y) \]

\[ (12a) \ \text{HIV}\_\text{Type1}(z) \rightarrow \text{list}(z) \]

\[ (12b) \ \text{HIV}\_\text{Type2}(z) \rightarrow \text{list}(z) \]

\[ (12c) \ \text{HepB}(z) \rightarrow \text{list}(z) \]

... 

The predicate list(z) is read as “z is listed”. HIV\_Type1(z) is read as “z is the disease agent HIV Type1”. We assume that there is a unique object that satisfies HIV\_Type1(z), i.e., it denotes a constant symbol. The formula says_{12a,12b,12c}(\text{list}(y)) accesses the items that are listed by (12a)-(12c) and can be understood as the translation of the phrase *the following disease agents*. The predicate test(x,y) is read as “x is tested for the disease agent y”. We note that a richer representation is needed to accommodate reasoning about events, such as, tests and tests for a particular purpose. In Chapter 3, we describe a case study using a prototype implementation that extends ReFL with an object-oriented representation of events.

While the list in (12a)-(12c) is simple, it illustrates the applicability of says for constructions other than exceptions. In this example, the identifiers let us draw inferences from particular list items, thereby allowing us to preserve the structure of the list in logic. To our knowledge, this use of identifiers is novel to ReFL. Previous non-monotonic formalisms based on default logic [26, 134] and defeasible logic [56, 60] make use of identifiers to specify priorities over rules, and we will discuss this in the
following example. However, the priorities are commonly specified in the meta-logic, which limits its applicability to cases where rules are in conflict. To refer to list items or definitions in particular sections, we need to parametrize predicates, such as, list, with the identifier. This results in the need to maintain identifiers in the object language for lists, and in the meta-language for priorities. The *says* predicate offers a unified mechanism for these kinds of inter-sentential references.

## 2.5.3 Priorities

Let us consider an example of nested exceptions to illustrate the need for priorities in default logic:

(13) Except as specified in (14), every donation of blood or blood component must be tested for evidence of infection due to Hepatitis B.

(14) Except as specified in (15), you are not required to test donations of source plasma for evidence of infection due to Hepatitis B.

(15) If a source plasma donation is used for treatment, it must be tested for Hepatitis B.

Such nested exceptions arise, for example, when a regulatory document is modified over time. For example, the regulatory base may initially consist of only (13) and (14). At some point, the regulator may realize that source plasma donations have unforseen uses, and exceptions, such as (15), may be added. The sentences are represented in *Refl* as follows:

\[
(13).o : d(x) \land \neg\text{says}_{14} \Rightarrow \neg test(x) \iff test(x)
\]

\[
(14).p : d(x) \land sp(x) \land \neg\text{says}_{15} \Rightarrow test(x) \iff \neg test(x)
\]

\[
(15).o : d(z) \land sp(z) \land uft(z) \iff \neg test(x)
\]

The predicate *uft*(z) is read as “z is used for treatment”. And, the other predicates are read as before. Suppose we are given a donation *o*, which is a source plasma
donation \( sp(o) \) and used for treatment \( uft(o) \). At the least fixed point, we obtain utterances from (13) and (15) – (13, \( \diamond test(o) \)) and (15, \( \diamond test(o) \)). If a donation is not used for treatment, we obtain utterances from either (13) or (14), as we discussed in Section 2.3.2.

These statements are represented in default logic without priorities as follows:

\[
\begin{align*}
  d(x) : & \quad \diamond test(x) \quad \Rightarrow (13).o \\
  d(x) \land sp(x) : & \quad \neg \diamond test(x) \quad \Rightarrow (14).p \\
  d(z) \land sp(z) \land uft(z) : & \quad \diamond test(x) \quad \Rightarrow (15).o
\end{align*}
\]

Since the consistency checks are w.r.t. all rules, in the case of a non-source plasma donation, the preconditions of both (13) and (14) will be ungrounded at the least fixed point. To avoid this, an order is specified over the rules [26], e.g., \( (15) > (14) > (13) \). With the order over rules, the statements are interpreted in an analogous manner to the corresponding statements in REFL above. The use of identifiers, as a parameter to the \( says \) predicate, lets us avoid the need for a separate specification of priorities.

We conclude this section by discussing some limitations of REFL. While the \( says \) predicate generalizes default logic with priorities, rules have to refer to their exceptions. However, there are examples in the CFR where the exceptions refer to the rule. Consider the following paraphrase of (14) and (15):

(16) You are not required to test donations of source plasma for evidence of infection due to Hepatitis B.

(17) If a source plasma donation is used for treatment, exemption (16) no longer applies.

(16) and (17) would be expressed in REFL in a similar manner to (14) and (15), i.e., there would be a reference from (16) to (17) using \( says \). However, the phrasing of the sentences suggest that (17) refers to (16). Such references commonly arise in the context of abrogation and annulment of laws [58]. In the example above, (17) \( annuls \)
the permission provided by (16). Governatori and Rotolo [58, 59] analyse abrogation and annulment by using a temporal version of defeasible logic. The analysis involves the use of meta-rules which abrogate or annul other rules. However, the presence of variables in ReFL make matters more complicated. Only a subset of the utterances produced are annulled. For example, (17) annuls the utterances produced by (16) for source plasma donations that are used for treatment. We speculate that a solution in ReFL will involve the formalization of the word applies in (17) and its connection to says. We leave an investigation to future work.

2.6 Conclusions

We have motivated and described a logic (ReFL) that accommodates references between laws. Inter-sentential references are expressed using formulas of the form says_{ld} \varphi, read as “the regulator says \varphi via the laws Id”. In Section 2.3, we formalized the evaluation of says via a combination of techniques from Reiter’s default logic [127] and Kripke’s theory of truth [85]. The says predicate offers a unified analysis of various kinds of inter-sentential references, e.g., priorities of exceptions over rules (Section 2.3.2), and references to definitions or list items (Section 2.5.2).

In Section 2.4, we discussed an axiomatization of the says predicate, and showed that the predicative analysis satisfies the axioms of a normal modal operator. In Chapter 4, we will use these axioms in a modal logic that combines saying with obligation and permission, to formalize concepts of power that arise in privacy regulation.

While ReFL provides an expressive mechanism for references, assessing conformance is EXPTIME-hard (Section 2.3.3). It would be infeasible to use ReFL to check conformance w.r.t. traces with a large number of objects. The main obstacle is the provability tests that are used in evaluating the says predicate. In the following chapter, we develop algorithms for checking conformance at run time. We describe an assumption which lets us compile out the predicate (by replacing it with tests of lower complexity), leading to efficient checking in practice.
Chapter 3

Conformance Checking at Runtime

In Chapter 2, we motivated and developed the logic ReFL to accommodate interstential references. The mechanism for references is a predicate \((\text{says})\) which is evaluated using provability tests. In this chapter, we develop runtime checking algorithms for ReFL, and describe a case study conducted using a prototype implementation.

We begin, in Section 3.1, with some background on runtime checking for LTL, based on the rule-based formalism Eagle [11]. Section 3.2 adapts the Eagle calculus for ReFL. While ReFL allows for a direct representation of references, the provability tests at runtime are expensive. In Section 3.3, we describe an assumption (based on a case study of the CFR) which lets us compile out the provability tests, replacing it with tests of lower complexity. The case study is described in Section 3.4.

3.1 Runtime Checking for LTL

In this section, we give some background on runtime checking for LTL. Section 3.1.1 introduces the concept of a monitor. Following Bauer et al. [14], in Section 3.1.2, we interpret a monitor’s state using three values. In Section 3.1.3, we introduce the rule-based formalism Eagle [11], which we will extend in the following section.
3.1.1 Monitor

In runtime checking, we observe a run of a system, one state at a time. The evaluation of a property at a point in time is based on the states seen so far, i.e., *the prefix of the run* up to that point. We begin by developing some notation for prefixes of runs, and then, turn to the definition of a monitor.

**Prefixes of Runs:** Let $S$ be the set of system states, where each state contains the interpretation of atomic predicates, and $F \subseteq S$ is a set of final states. A prefix of a run is a finite sequence of states $(s_1, ..., s_n) \in S^n$. The set of all prefixes is denoted by $S^* = \bigcup_{n \in \mathbb{N}} S^n \cup \{\epsilon\}$. Given $\sigma \in S^*$, $|\sigma|$ denotes the length of the prefix and $\sigma_i$ denotes the $i^{th}$ element for $1 \leq i \leq |\sigma|$. A prefix is said to have terminated iff $|\sigma| \geq 1$ and $\sigma_{|\sigma|} \in F$.

A run $R = (r, \pi_1, ..., \pi_n)$ is said to have the prefix $\sigma \in S^*$ if for all $1 \leq i \leq |\sigma|$, we have $r(i) = \sigma_i$ and $(o_1, ..., o_j) \in \pi_j(p, \sigma_i)$ iff $p(o_1, ..., o_j) \in \sigma_i$. In addition, if $\sigma$ has terminated, then for all $j > |\sigma|$, we require that $r(j) = \sigma_{|\sigma|}$. Note that if $\sigma$ has terminated, there is a unique run $R$ which has the prefix $\sigma$, and we call $R$ the run generated by $\sigma$.

**Monitors:** A *monitor* for an LTL formula $\varphi$ receives as input a prefix of a run, one state at a time. On receiving a state, a monitor raises an alarm (denoted $a$) if a violation is detected. Otherwise, no alarm is raised (denoted $\bar{a}$). We now define a monitor formally:

**Definition 3.1 (Monitor).** A monitor for $\varphi$, denoted $m_\varphi : S^* \rightarrow \{a, \bar{a}\}$, is a function such that for all terminated prefixes $\sigma \in S^*$, if the run $R$ generated by $\sigma$ is such that $(R, 1) \not\models \varphi$, $m_\varphi(\sigma) = a$.

A monitor $m_\varphi$ is optimal if for all $\sigma \in S^*$, $m_\varphi(\sigma) = \bar{a}$ iff there is a run $R$ which has the prefix $\sigma$ and $(R, 1) \models \varphi$.

In other words, a monitor is guaranteed to raise an alarm if the run generated by a terminated prefix does not satisfy $\varphi$. And, a monitor is optimal if it raises an alarm
as early as possible, i.e., if no alarm is raised, then the prefix can be extended into a run satisfying $\varphi$.

### 3.1.2 Three-Valued Monitors

There is an important difference between safety and liveness properties from the perspective of runtime checking. For a global or safety property, e.g., $\Box p$, a monitor can raise an alarm as soon as it receives a state where $p$ is not true, but it needs to check every state for the truth of $p$. On the other hand, for an eventual or liveness property, e.g., $\Diamond p$, a monitor has to wait until the end of the run and raises an alarm if $p$ is false at all states, but it can stop checking as soon as it receives a state where $p$ is true. This difference suggests a three-valued interpretation of a monitor’s state [14]:

(a) $\top$ - True. No alarm will be raised, and no more states need to be checked.

(b) $\ ?$ - Undetermined. Depends on future states.

(c) $\bot$ - False. An alarm is raised.

Let $B^3 = \{\top, ?, \bot\}$. We can now define a three-valued monitor:

**Definition 3.2 (Three-valued Monitor).** Given a monitor $m_\varphi$, $m^3_\varphi : S^* \rightarrow B^3$ is a three-valued interpretation of $m_\varphi$ if for all $\sigma \in S^*$, $m^3_\varphi(\sigma) = \bot$ iff $m_\varphi(\sigma) = a$. In addition, for all terminated prefixes $\sigma \in S^*$, if the run $R$ generated by $\sigma$ is such that $(R, 1) \models \varphi$, $m^3_\varphi(\sigma) = \top$.

$m^3_\varphi$ is optimal if $m_\varphi$ is optimal and for all $\sigma \in S^*$, $m^3_\varphi(\sigma) = \top$ iff for all runs $R$ which have the prefix $\sigma$, $(R, 1) \models \varphi$.

In other words, a three-valued interpretation $m^3_\varphi$ agrees with the underlying monitor $m_\varphi$ on when an alarm is raised, and if $m^3_\varphi(\sigma) = \top$, no more states need to be checked. $m^3_\varphi$ is optimal if it raises an alarm or stops checking as early as possible.

We can show the following:

**Proposition 3.1.** Given an LTL formula $\varphi$:

(1) There is a unique optimal monitor $\hat{m}_\varphi$ and it has a unique optimal three-valued interpretation $\hat{m}^3_\varphi$.
Given a prefix $\sigma \in S^*$, deciding whether $m^3_\varphi(\sigma) = \bot$ is PSPACE-complete.

The first item follows easily from Definitions 3.1 and 3.2. For the second item, the decision problem $m^3_\varphi(\sigma) = \bot$ encodes the result of satisfiability testing, e.g., $m^3_\varphi(\varepsilon) = \bot$ iff $\varphi$ is not satisfiable. And, satisfiability testing for LTL is PSPACE-complete.

It is often of interest to use extensions of LTL in monitors, for which the satisfiability question is undecidable. For example, Barringer et al. [11] allow arbitrary Java expressions as atomic propositions, Basin et al. [13] use a variant of first-order LTL, and REFL is more powerful than first-order LTL. In these cases, synthesizing the optimal monitor is undecidable in general. However, there is a fragment of REFL, which suffices for the CFR, and for which optimal monitors can be synthesized. We will discuss this in Section 3.2.

### 3.1.3 The EAGLE Formalism

We now describe the EAGLE formalism [11] for monitor synthesis. The monitors obtained are not optimal, but the mechanism generalizes naturally to extensions of LTL. The core of the EAGLE formalism is a calculus for transforming formulas based on a state. The calculus (see [11]) provides a general treatment of past modalities and data dependencies. For simplicity, we will work directly with formulas in the logic. We begin by giving a brief sketch of the calculus, followed by a discussion of two examples. Then, we present the formal definitions.

**Sketch:** Let us assume as given a propositionalized PredLTL formula $\varphi$, i.e., with variables replaced by object names. When we receive a state $s_1$, we transform the formula $\varphi$ into another formula $\varphi_1 = \tau(\varphi, s_1)$. On receiving the next state $s_2$, we transform $\varphi_1$ into another formula $\varphi_2 = \tau(\varphi_1, s_2) = \tau(\tau(\varphi, s_1), s_2)$. And, so on. After receiving $n$ states, we will have a formula $\varphi_n$. We use a function $\eta$ to map formulas to truth values. For example, if $\eta(\varphi_n, s_n)$ is true, then we will conclude that the run satisfies $\varphi$. If $\eta(\varphi_n, s_n)$ is false, then we conclude that the run cannot be extended to
satisfy $\varphi$. Otherwise, $\eta(\varphi_n, s_n)$ is undetermined, and we continue to wait for states. However, if $s_n$ is a final state, we will ensure that $\eta$ returns either true or false.

**Example 1:** Suppose we are given $\varphi = \Box p(o)$. On receiving the state $s_1$ such that $p(o) \in s_1$, we transform $\varphi$ into $\varphi_1 = \tau(\varphi, s_1) = \tau(p(o), s_1) \land \Box p(o) = \top \land \Box p(o)$. The truth value assigned to $\varphi_1$ using the function $\eta$ is (the conjunction of $\top$ and) the truth value assigned to $\Box p(o)$. The truth value assigned to $\Box p(o)$ depends on whether or not $s_1$ is a final state. If $s_1$ is not a final state, then $\eta(\Box p(o), s_1)$ is undetermined. And, $\eta(\Box p(o), s_1)$ is true, if $s_1$ is a final state. Thus, if $s_1$ is non-final, $\eta(\varphi_1, s_1)$ is undetermined, and we will wait for the next state. If $s_1$ is final, $\eta(\varphi_1, s_1)$ is true, and we will conclude that the run satisfies the formula. However, if $p(o) \not\in s_1$, $\tau(p(o), s_1) = \bot$. As a result, $\eta(\varphi_1, s_1)$ would be false (regardless of the truth value assigned to $\Box p(o)$), and we raise an alarm.

**Example 2:** An eventual formula $\varphi = \Diamond p(o)$ is handled via the duality with global formulas, i.e., $\Diamond p(o) = \neg \Box \neg p(o)$. Here, we can conclude that the formula is true as soon as we see a state where $p(o)$ is true. However, we need to wait till a final state before we conclude that an eventuality has not been satisfied. To see that this procedure is not optimal, consider the formula $\Diamond \bot$. On receiving a state, we will transform it into $\bot \lor \Diamond \bot$. But, we cannot detect that this formula is unsatisfiable until a final state is received.

**Formal definitions:** We now turn to the formal definitions, starting with the transformation function:

**Definition 3.3** (Transformation function). The transformation function $\tau : L \times S \to L$ is defined as follows:

- $\tau(p(o_1, ..., o_j), s) = \top$ if $p(o_1, ..., o_j) \in s$ and $\bot$ otherwise
- $\tau(\varphi \land \psi, s) = \tau(\varphi, s) \land \tau(\psi, s)$, and $\tau(\neg \varphi, s) = \neg \tau(\varphi, s)$
- $\tau(\Box \varphi, s) = \tau(\varphi, s) \land \Box \varphi$
The transformation function is extended to prefixes. Given $\sigma = (s_1, ..., s_n) \in S^*$, let $\sigma' = (s_2, ..., s_n)$. Then, $\tau(\varphi, \sigma) = \tau(\tau(\varphi, s_1), \sigma')$.

Next, we define a function to map formulas to truth values:

**Definition 3.4 (Truth Value Mapping).** The function $\eta : L \times S \rightarrow B^3$ is defined as follows:

- Predicates, conjunction, and negation are handled in the usual way.
- $\eta(\Box \varphi, s) = \top$ if $s$ is a final state.
  $\eta(\Box \varphi, s) = ?$ otherwise.

Given a propositionalized PredLTL formula $\varphi \in L$, the Eagle monitor for $\varphi$, denoted by $m^E_{\varphi}$, is given by:

$$m^E_{\varphi}(\sigma) = \eta(\tau(\varphi, \sigma), \sigma|_{\sigma})$$

In other words, we transform the formula at each state in the prefix and then assign a truth value at the last state. It is easy to establish that $m^E_{\varphi}$ is a three valued monitor (Definition 3.2). We now extend this process of transforming and evaluating formulas to accommodate references between laws.

**3.2 Adapting the Eagle Calculus for ReFL**

The key idea is to treat the predicate $\text{says}_{id} \varphi$ as kind of eventuality. As we discussed in Section 2.3.2, to evaluate $\text{says}_{id} \varphi$ at time $i$, we need to check the utterances obtained from the laws in Id at time $i$. If the preconditions of the laws in Id are temporal, we need to wait until they are evaluated before the utterances are obtained. So, we need to keep utterances for a time $i$ until all subformulas $\text{says}_{id} \varphi$ for time $i$ have been evaluated. Given $\text{says}_{id} \varphi$ and a time $i$, we attempt to evaluate it using the current set of utterances. If we cannot determine the truth value, $\text{says}_{id} \varphi$ is
transformed into \( \text{says}_{Id}(\varphi, i) \) (read as “\( \text{says}_{Id} \varphi \) is true at time \( i \)”), and evaluated at subsequent times. We extend the syntax of preconditions to accommodate such formulas:

**Definition 3.5 (Syntax of Preconditions).** Given countable sets \( \Phi_1, \ldots, \Phi_n \) (of predicate names), object names \( O \), and a set of variables \( X \), the language \( L' (\Phi_1, \ldots, \Phi_n, O, X) \), abbreviated as \( L' \), is the smallest set such that:

- \( p(y_1, \ldots, y_j) \in L' \), where \( p \in \Phi_j \) and \( (y_1, \ldots, y_j) \in X^j \).
- If \( \varphi \in L' \), then \( \neg \varphi \in L' \) and \( \varphi \lor \psi \in L' \).
- If \( Id \subseteq \text{ID} \) and \( \varphi \in L(\Phi_1, \ldots, \Phi_n, X) \) (Definition 2.2), then \( \text{says}_{Id} \varphi \in L' \). In addition, for all natural numbers \( i \in \mathbb{N} \), \( \text{says}_{Id}(\varphi, i) \in L' \).

The syntax of regulatory statements (Definition 2.3) is modified so that the preconditions of laws are statements from \( L' \). The set \( L' \) together with a set of regulatory statements \( \text{Reg} \) is denoted by \( L^+ = L' \cup \text{Reg} \). Given a set of objects \( O \), \( V(X, O) \) denotes the set of all variable assignments, i.e., functions \( v : X \to O \).

We now extend the transformation function (Definition 3.3). As in Chapter 2, we use two utterance sequences \( u \) and \( u' \) such that for all \( i \), \( u(i) \subseteq u'(i) \). \( u(i) \) is the set of utterances obtained from laws with true preconditions, while \( u'(i) \) is set of utterances from laws with true or undetermined preconditions. The truth of \( \text{says}_{Id} \varphi \) is determined using \( u \), and falsity is determined using \( u' \).

**Definition 3.6 (Transformation function).** Given a pair of utterance sequences \( u \) and \( u' \) such that \( u(i) \subseteq u'(i) \) for all \( i \in \mathbb{N} \), the transformation function \( \tau_{(u, u')} : L^+ \times S \times \mathbb{N} \to L^+ \) is defined as follows:

\[
\begin{align*}
\tau_{(u, u')}(\text{says}_{Id}(\varphi), s, i) &= \tau_{(u, u')}(\text{says}_{Id}(\varphi, i), s, i) \\
\tau_{(u, u')}(\text{says}_{Id}(\varphi, j), s, i) &= \begin{cases} 
\top & \text{if } j \leq i \text{ and } u(j)_{Id} \models \varphi \\
\bot & \text{if } j \leq i \text{ and } u'(j)_{Id} \not\models \varphi \\
\text{says}_{Id}(\varphi, j) & \text{otherwise}
\end{cases}
\end{align*}
\]
For all other formulas, Definition 3.3 is used.

Note that the postcondition of permissions are not transformed, as their truth value is irrelevant. The only use of postconditions of permissions is to provide utterances. We now define the function to map formulas to truth values:

**Definition 3.7 (Truth Value Mapping).** Given a pair of utterance sequences $u$ and $u'$ such that $u(i) \subseteq u'(i)$ for all $i \in N$, the function $\eta_{(u,u')} : L^+ \times S \times N \rightarrow B^3$ is defined as follows:

- $\eta_{(u,u')} (\text{says}_{Id}(\varphi, s, i)) = \eta_{(u,u')} (\text{says}_{Id}(\varphi, i), s, i)$
- $\eta_{(u,u')} (\text{says}_{Id}(\varphi, j), s, i) = ?$
- $\eta_{(u,u')} ((\text{id}).o : \varphi \mapsto \psi, s, i) = \eta_{(u,u')} (\varphi \Rightarrow \psi, s, i)$.
- $\eta_{(u,u')} ((\text{id}).p : \varphi \mapsto \psi, s, i) = \top$
- $\eta_{(u,u')} ((\text{id}) \varphi \mapsto \psi, s, i) = \top$
- For all other formulas, Definition 3.4 is used.

At the end of the trace, subformulas $\square \varphi$ are replaced by $\top$, but subformulas $\text{says}_{Id}(\varphi, j)$ may still be undetermined. This is due to the fact that with circular references, we can create paradoxical statements $- (\text{id}).o : \neg \text{says}_{id} \varphi \mapsto \varphi$. This statement requires $\varphi$ to hold when it doesn’t require $\varphi$, and is always undetermined.

Algorithm 1 describes the procedure for computing the least fixed point in a runtime setting. In addition to $u$ and $u'$, we maintain a set of tuples $\Phi$, where each element is a transformed regulatory statement, the associated utterance, and time. Given $((\text{id}).x : \varphi \mapsto \psi, a, j) \in \Phi$, if $\varphi$ is determined to be true, the utterance $a$ is added to $u(j)$. On the other hand, if $\varphi$ is determined to be false $a$ is removed from
Update(Reg, Φ, u, u', s, i):

Input: The regulation Reg, the set of formulas to be updated Φ, the utterance sequences u and u', the state s and time i

Let u(i) = u'(i) = ∅;
Let Φ' = ∅;
for all ((id).x : ϕ ⇀ ψ, a, j) ∈ Φ do
Φ' = Φ' ∪ {τ(u,u')(((id).x : ϕ ⇀ ψ, s, i)};
end
Φ = Φ';
for all (id).x : ϕ ⇀ ψ ∈ Reg and assignments v do
Let φ = τ(u,u')(v(((id).x : ϕ ⇀ ψ), s, i));
Φ = Φ ∪ {(φ, (id, v(ψ)), i)}, and u'(i) = u'(i) ∪ {(id, v(ψ))}
end
repeat
for all e = ((id).x : ϕ ⇀ ψ, a, j) ∈ Φ do
Let φ = (id).x : ϕ ⇀ ψ;
If η(u,u')(ϕ, s, i, v) = ⊤, then u(j) = u(j) ∪ {a};
If η(u,u')(ϕ, s, i, v) = ⊥, then u'(j) = u'(j) − {a};
If η(u,u')(ϕ, s, i, v) ≠ ? and η(u,u')(ϕ, s, i, v) ≠ ?, Φ = Φ − {e};
If η(u,u')(ϕ, s, i, v) = ⊥, then raise alarm.
end
until u and u' do not change;

Algorithm 1: An algorithm for evaluating statements with references
For all \( j \in N \), \( u(j) \) increases monotonically, and \( u'(j) \) decreases monotonically with each execution of the repeat loop, until a fixed point is reached.

As we mentioned in Section 3.1, Algorithm 1 does not determine a formula to be true or false as early as possible. To decide if a formula is true as early as possible, we need to check whether all possible suffixes to the trace satisfy the formula [14]. In other words, we need to decide if the transformed formula is valid. In Chapter 2, we showed that with references one can encode formulas in first-order logic as regulations, and as a result, the validity problem is undecidable. The satisfiability tests used to evaluate the \( \text{says} \) predicate are in propositional LTL, and are decidable.

In the case where preconditions of laws are atemporal, optimal monitors can be synthesized. This is because the fixed point can be computed as soon as a state is received. The satisfaction of postconditions of obligations can then be handled using automata-based algorithms [14]. The sentences that we have examined in Section 610 of the CFR can be formalized with atemporal preconditions.

### 3.3 Precomputing Satisfiability

The main practical difficulty with Algorithm 1 is the size of the satisfiability tests that are used to evaluate the \( \text{says} \) predicate. In this section, we describe an empirically motivated assumption that lets us compile out the \( \text{says} \) predicate, i.e., by replacing it with tests of lower complexity during checking. Section 3.3.1 describes the assumption, called the single copy property, by contrasting two examples. In Section 3.3.2, we show how the assumption can be used to compile out the \( \text{says} \) predicate.

#### 3.3.1 The Single Copy Property

The complexity of Algorithm 1 in each state of a run depends on two factors – the number of steps necessary to reach a fixed point, and the size of the satisfiability instances that need to be handled in the evaluation of the predicate \( \text{says}_{Id} \varphi \). We
discuss examples that illustrate these two aspects, by encoding the graph reachability problem in different ways. In the first example, the number of steps taken to reach the fixed point grows with the number of objects. In the second example, the size of the satisfiability instances grows with the number of objects.

Both examples operate on the same model, where a state in the run contains a description of a graph. Objects $o_1$ and $o_2$ represent nodes, and the predicate $\delta(o_1, o_2)$ is true iff there is an edge between $o_1$ and $o_2$. In addition, $\delta^+(o_1, o_2)$ is true iff there is a path from $o_1$ to $o_2$. Suppose we wish to check whether $\delta^+$ has been computed correctly.

**Example 1.** Consider a self-referential sentence:

$$(id).o : \delta(x, z) \lor (\delta(x, y) \land \text{says}_{\{id\}} \delta^+(y, z)) \mapsto \delta^+(x, z)$$

The precondition of this sentence corresponds to the definition of a path. In other words, there is a path between $x$ and $z$ ($\delta^+(x, z)$), if there is an edge between $x$ and $y$ ($\delta(x, y)$), and a path between $y$ and $z$ ($\text{says}_{\{id\}} \delta^+(y, z)$). Let $u_0, ..., u_f$ be the utterance sequences obtained in the least fixed point computation. It is easy to see that $(id, \delta^+(o, o')) \in u_j(i)$ iff there is a path of length at most $j$ from $o$ to $o'$. Given a graph with $|O|$ nodes, there is a path from $o$ to $o'$ iff there is a path of length at most $|O|$ from $o$ to $o'$. As a result, the fixed point will be reached in at most $|O|$ steps. The worst-case number of steps needed to reach the fixed point is $O(m \times |O|^k)$, where $m$ is the size of the regulation, and $k$ is the maximum number of variables appearing in a sentence.

**Example 2.** Consider now the following statements:

$$(A).o : \text{says}_{\{B, C\}} \delta^+(x, y) \mapsto \delta^+(x, y)$$

$$(B).o : \delta(x, y) \mapsto \delta^+(x, y)$$

$$(C).o : \top \mapsto (\delta^+(x, y) \land \delta^+(y, z)) \Rightarrow \delta^+(x, z)$$

Note that $A$ refers to $C$. The presence of implication in the postcondition of $C$ is an important feature of this example. Let, for simplicity, the graph in the state
be a chain. Since the precondition of C is always true, the first step of the fixed point computation yields an utterance that contains \((C, (\delta^+(o, o') \land \delta^+(o', o'')) \Rightarrow \delta^+(o, o'')) \in u_1(i)\) for all \(o, o', o''\) in the graph. The next step of the evaluation will yield the fixed point, but the size of the validity test performed in this step is \(O(|O|^3)\), as Algorithm 1 uses all the available utterances. The worst-case size of the validity instances is in \(O(m \times |O|^k)\), and the time complexity of a step in computing the fixed point is \(O(2^{m \times |O|^k})\).

**Discussion.** In both examples above, Algorithm 1 checks validity instances of size polynomial in \(|O|\). However, there is a crucial difference in the maximum size of tests that are needed. In Example 1, \(\text{says}_{\{id\}} \delta^+(o, o')\) is true iff \((id, \delta^+(o, o')) \in u(i)\). In other words, at most one utterance is need to evaluate \(\text{says}_{\{id\}} \delta^+(o, o')\). In Example 2, we do need validity tests of size \(|O|\) to evaluate \(\text{says}_{\{B,C\}} \delta^+(o, o')\). A case study of the CFR revealed that the references behaved like Example 1 in that a single utterance or copy of the referenced statement suffices to evaluate formulas \(\text{says}_{Id} \phi\). We call this the single copy property.

**Definition 3.8 (Single Copy Property).** Given a body of regulation \(Reg\), \(\text{says}_{Id}(\phi, j)\) has the single copy property iff for all points \((R, i)\), and utterance sequences \((u, u')\):

\[
\eta(u, u')(\text{says}_{Id}(\phi, j), R, i, v) = \begin{cases} 
\top & \text{if } \psi \models v(\phi) \text{ for some } \psi \in u(j)_{Id} \\
\bot & \text{if } \psi \not\models v(\phi) \text{ for all } \psi \in u'(j)_{Id} \\
? & \text{otherwise}
\end{cases}
\]

Verifying whether the single copy property holds is undecidable. We have not been able to identify useful sufficient conditions for which the property holds. For example, if the postconditions of laws are atomic predicates, then \(\text{says}_{Id}(p(x), j)\) has the single copy property. However, restricting postconditions to atomic predicates makes the translation of regulation to logic more difficult. Furthermore, in the presence of negation, the postconditions of laws can lead to the truth of \(\text{says}_{Id}(\bot, j)\), unless the preconditions ensure that this does not happen. As a result, a useful characterization would have to involve restrictions on the preconditions as well. We leave an investi-
gation of these issues to future work. In the rest of this chapter, we will assume that the single copy property holds, and study what advantages it offers in the checking process.

The single copy property allows us to reduce the size of the satisfiability tests. However, we need to perform $O(m \times |O|^k)$ tests for each inference predicate. The question arises as to whether satisfiability tests can be avoided during checking. We answer this question positively in the following section.

3.3.2 The Pre-computation Procedure

In this section, we show that the single copy property gives us a way to assess satisfiability symbolically, and use tests of lower complexity during checking. The strategy we use is as follows. Given a body of regulation, we perform a compilation step which involves: a) testing satisfiability, and b) replacing the predicates $\text{says}_{Id} \varphi$ by equivalent formulas in another logic. We begin by discussing two examples, and then formalize the compilation step.

**Example 1:** Consider our regulatory sentences:

- (3). $o: d(x) \land \neg\text{says}_{[4]} \neg\Diamond test(x) \mapsto \Diamond test(x)$, and

- (4). $p: d(y) \land sp(y) \mapsto \neg\Diamond test(y)$

Consider a state at which $o_1, o_2, ..., o_n$ are source plasma donations. This would result in $\neg\Diamond test(o_1), \neg\Diamond test(o_2), ..., \neg\Diamond test(o_n)$ being available as utterances. To evaluate $\text{says}_{Id} \neg\Diamond test(o_i)$, Algorithm 1 uses all the utterances in the satisfiability test. However, in this case, it suffices to check if $\neg\Diamond test(o_i)$ is present as an utterance. The other utterances are irrelevant. To check if $\neg\Diamond test(o_i)$ is present as an utterance, it suffices to evaluate the precondition of the referenced law, i.e., whether $d(o_i) \land sp(o_i)$ is true (whether $o_i$ is a donation of source plasma). Instead of evaluating $\text{says}_{Id} \varphi$ using satisfiability tests, we will check if the precondition of a referenced law is true.
Informally, the compilation step involves answering the question when does statement 2 provide an exception for statement 1. Equivalently, when does $\neg \diamond \text{test}(y)$ imply $\neg \diamond \text{test}(x)$. The answer is only when $y = x$. We can then evaluate the precondition of 2 with $y$ replaced by $x$, i.e., $d(x) \land sp(x)$. This lets us replace statement 1 with $(3) \circ d(x) \land \neg d(x) \land sp(x)) \iff \diamond \text{test}(x)$, which is equivalent to $(3) \circ d(x) \land \neg sp(x) \iff \diamond \text{test}(x)$. Observe that this is the derived obligation implied by statements 1 and 2, i.e., every non-source plasma donation must be tested.

Example 2: The example above is simple in two ways: a) the number of variables in both statements are the same, and b) the references are acyclic. We discuss the general case in the context of the reachability example we saw in the previous section:

$$(id) \circ \delta(x, z) \lor (\delta(x, y) \land \exists y_1 : P_{id}(\{x/y, y/y_1, z/z\})) \iff \delta^+(x, z)$$

We observe that the precondition is structurally similar to a procedure that checks if a path exists between two nodes $x$ and $z$. That is, if $\delta(x, z)$ then $\delta^+(x, z)$ is true. Otherwise, if there exists $y$ such that $\delta(x, y)$ and there is a path from $y$ to $z$, then $\delta^+(x, z)$ is true, otherwise false.

We will produce a formula which mimics the procedure. There are two pieces of machinery used by the procedure that are not directly available in the logic: a) an existential quantifier over objects (there exists $y$), and b) a mechanism for recursion. To address this, let us consider a logic which extends PredLTL with existential quantifiers, and a function symbol $P_{id}$ for $id \in ID$ (P stands for precondition). $P_{id}$ takes as argument a substitution $\theta : X \rightarrow X$, which is a function from variables to variables. A substitution is represented a set of replacements $x/y$ (read as “$x$ is replaced by $y$”), such that each variable has at most one replacement. We replace the formula above with:

$$(id) \circ \delta(x, z) \lor (\delta(x, y) \land \exists y_1 : P_{id}(\{x/y, y/y_1, z/z\})) \iff \delta^+(x, z)$$

It remains to give this formula a semantics. Given a variable assignment $\nu$ and a substitution $\theta$, $\theta(\nu)$ denotes the variable assignment $\nu'$ such that $\nu'(x) = \nu(\theta(y))$. Given a point $(R, i)$ and regulation $Reg$, the idea is to say that $(R, i, \nu) \models P_{id}(\theta)$ iff
(R, i, \theta(v)) \models \varphi \text{ where } (id).x : \varphi \mapsto \psi \in \text{Reg}. \text{ We now formalize the compilation procedure.}

**Compiling References into Precondition Tests:** We begin by defining the syntax of compiled preconditions:

**Definition 3.9 (Syntax of Compiled Preconditions).** Given sets \( \Phi_1, \ldots, \Phi_n \) (of predicate names), a set of variables \( X \), and a finite set of identifiers \( ID \), the language \( L'_C(\Phi_1, \ldots, \Phi_n, X, ID) \), abbreviated as \( L'_C \), is the smallest set such that:

- If \( t \in \mathcal{B}^3 \), \( t \in L'_C \). And, \( p(y_1, \ldots, y_j) \in L'_C \) where \( p \in \Phi_j \) and \( (y_1, \ldots, y_j) \in X^j \).

- If \( \varphi \in L'_C \), then \( \neg \varphi \in L'_C \) and \( \Box \varphi \in L'_C \). If \( \varphi, \psi \in L'_C \), then \( \varphi \land \psi \in L'_C \)

- If \( \varphi \in L'_C \), for all \( y \in X \), we have \( \exists y : \varphi \in L'_C \).

- For all \( id \in ID \) and substitutions \( \theta : X \to X \), we have \( P_{id}(\theta) \in L'_C \). In addition, for all natural numbers \( i \in N \), \( P_{id}(\theta, i) \in L'_C \)

The syntax of regulatory statements (Definition 2.3) is modified so that the preconditions of laws are statements from \( L'_C \). The set \( L'_C \) together with a set of regulatory statements \( \text{Reg}_C \) is denoted by \( L'_C = L'_C \cup \text{Reg}_C \). We remind the reader that \( L^+ \) and \( L' \) are the languages with the predicate \( \text{says}_{id} \varphi \).

The semantics of \( L'_C \) is defined in a manner similar to \( L^+ \). Rather than using utterances \((u, u')\), we now evaluate statements w.r.t. two sets of assignment functions \((\gamma, \gamma')\). \( \gamma(i, id) \) (resp., \( \gamma'(i, id) \)) is a set of variable assignments for which the precondition of the law with identifier \( id \) is true (resp., true or undetermined). As with utterances, we require that for all \( i \in N \) and \( id \in ID \), \( \gamma(i, id) \subseteq \gamma'(i, id) \). Given an assignment \( v \) and a substitution \( \theta, \theta(v) \) denotes the assignment \( v' \) such that for all \( y \in X \), we have \( v'(y) = v(\theta(y)) \). We can now adapt the \( \eta \) function:

\[
\eta_{(\gamma, \gamma')}(P_{id}(\theta, j), R, i, v) = \begin{cases} 
\top & \text{if } \theta(v) \in \gamma(j, id) \\
\bot & \text{if } \theta(v) \notin \gamma'(j, id) \\
? & \text{otherwise}
\end{cases}
\]
The definitions of fixed points are easily adapted, and we leave the details to the reader.

We now describe the compilation procedure. Given $\varphi \in L^+$, we use $X(\varphi)$ to denote the set of variables appearing in $\varphi$, and $\theta(\varphi)$ to denote the formula obtained by performing the substitution $\theta$ on $\varphi$. Consider $\text{says}_{Id}(\varphi, j)$, which has the single copy property, and variables disjoint from all regulatory statements:

- Let $S(\varphi, id) = \{ \theta | (id).x : \phi \mapsto \psi \in \text{Reg}, \text{ and } \models \theta(\psi \Rightarrow \varphi) \}$.
- For all $\theta \in S(\varphi, id)$, let $\phi' = (id).x : \phi \mapsto \psi \in \text{Reg}$. We define the formula:

$$\varphi_C(\theta, j, id) = \exists z_1, ..., z_m : P_{id}(\theta, j)$$

Where the existentially quantified variables are in one-to-one correspondence with those in $X(\phi') - X(\varphi)$. More formally, $z_j \notin X(\phi') \cup X(\varphi)$ and $\theta$ is a one-to-one function from $\{z_j|1 \leq j \leq m\}$ to $X(\phi') - X(\varphi)$.

- $\varphi_C(\text{says}_{Id}(\varphi, j), id) = \bigvee \{ \varphi_C(\theta, j, id) | \theta \in S(\varphi, id) \}$, and
- $\varphi_C(\text{says}_{Id}(\varphi, j)) = \bigvee \{ \varphi_C(\text{says}_{Id}(\varphi, j), id) | id \in Id \}$

We note that the first step makes crucial use of the single copy property (SCP). In computing $S(\varphi, id)$, it suffices to find substitutions such that $\models \theta(\psi \Rightarrow \varphi)$. If the SCP does not hold, then we need to check if multiple copies of postconditions provide the necessary implication (as in Example 2, Section 3.3.1). For example, we need to check if $\theta(\psi_1 \land ... \land \psi_n \Rightarrow \varphi)$, where $\psi_1, ..., \psi_n$ are copies of the postcondition of a law with the variables renamed. It can be shown that detecting whether the SCP holds is undecidable. In future work, we plan to investigate restrictions on postconditions that make SCP-detection decidable.

To prove the correctness of the compilation procedure, we use a notion of correspondence between utterances and assignments. Let us assume as given a body of regulation $\text{Reg}$ (in $L^+$), a run $R$ and consistent utterances $(u, u')$. Rather than producing a regulation in $L_C^+$, we prove correctness by evaluating formulas in $L_C$.
against Reg. We construct \((\gamma_u, \gamma_u')\) such that for all \(i \in N\) and \(id \in ID\), \(v \in \gamma_u(i, id)\) iff \((id, v(\psi)) \in u(i)\), and \(v \in \gamma_u'(i, id)\) iff \((id, v(\psi)) \in u'(i)\). We can now show the following:

**Lemma 3.1.** Given a body of regulation \(Reg\), a run \(R\), utterance sequences \((u, u')\), and says\(_{id}\) \(\varphi, j\) which has the single copy property, for all \(i \in N\) and assignments \(v\):

\[\eta(u, u')(\text{says}_{id}(\varphi, j), R, i, v) = \eta(\gamma_u, \gamma_u')(\varphi(\text{says}_{id}\theta, j), R, i, v)\]

**Proof.** The proof follows straightforwardly from the construction of \(\gamma_u, \gamma_u'\) and the single copy property. We sketch one of the cases.

Suppose \(\eta(u, u')(\text{says}_{id}(\varphi, j), R, i, v) = \top\). There exists \((id, v'(\psi)) \in u(i)\) such that \(v'(\psi) \land v(\neg \varphi)\) is not satisfiable, or equivalently \(v'(\psi) \Rightarrow v(\varphi)\) is valid. It follows that there exists a substitution \(\theta\) such that \(\theta(\psi \Rightarrow \varphi)\) is valid. By definition \(v' \in \gamma_u(i)\), and hence, \(\eta(\gamma_u, \gamma_u')(\text{P}_{id}(\theta, j), R, i, v') = \top\). We can then argue using the construction that \(\eta(\gamma_u, \gamma_u')(\exists z_1, \ldots, z_m : \text{P}_{id}(\theta, j), R, i, v) = \top\), and as a result, \(\eta(\gamma_u, \gamma_u')(\varphi(\text{says}_{id}(\theta, j)), R, i, v) = \top\). The other cases are handled similarly. \(\square\)

Given \(Reg\) in which all subformulas says\(_{id}\) \(\varphi\) have the single copy property, we can now produce the regulation \(Reg_C\) in \(L_C^+\) with all occurrences of says\(_{id}\) \(\varphi\) replaced by \(\varphi(\text{says}_{id}\varphi)\). It follows from Lemma 3.1 that if \((u, u')\) is a fixed point w.r.t. \(Reg\), then \((\gamma_u, \gamma_u')\) is a fixed point w.r.t. \(Reg_C\). In addition, the truth values assigned to regulatory statements are identical.

The complexity of evaluation depends on the number of disjuncts in \(\varphi(\text{says}_{id}\varphi)\), which in turn depends on the size of the set: \(S(\varphi, id)\). \(|S(\varphi, id)| \leq (2k)^k\), where \(k\) is the maximum number of variables in a regulatory statement. \((2k)^k\) is a bound on the number of equivalence classes, i.e., we have \(2k\) variables (\(k\) in \(\varphi\) and \(k\) in \(\psi\)) and at most one equivalence class for each variable. Hence, the size of \(\varphi(\text{says}_{id}\varphi)\) is \(O(m \times (2k)^k)\), where \(m\) is the number of regulatory statements. Each quantified precondition test can be evaluated in \(O(|O|^k)\) time, where \(O\) is the set of objects. As a result, the time complexity for evaluating \(\varphi(\text{says}_{id}\varphi)\) is \(O(m \times (2k)^2k \times |O|^k)\). In the sections of the CFR that we have formalized, \(|S(\varphi, id)| \leq 1\), and the observed
time complexity is $O(m \times |O|^k)$. As a result, it is much more efficient to check the compiled formulas, as opposed to testing satisfiability during checking.

### 3.4 Case Study

We now discuss a case study conducted using a prototype implementation of the checker. In Section 3.4.1, we describe the interface between the regulation and traces (schemas), and how the logic operates over instances of schemas. We then turn to the formalization of Section 610.40 of the FDA CFR in ReFL (Section 3.4.2). Section 3.4.3 shows that the approach is practicable by considering the performance of the checker on (synthetic) states with a large number of objects.

#### 3.4.1 Schemas, Predicates, and the SAT Solver

Schemas form the interface between the regulation and trace. A schema is a set of class and type definitions. Classes can inherit from others, and have attributes which have atomic types, tuples or unions of types, pointers to other objects or sets of values. Figure 3.1 shows the schema definition for donations. The attributes include the material of the donation (blood or source plasma), the donor (which is a pointer to an object carrying information about the donor), the recipient of the donation, and the tests that have been performed.

Predicates are defined using the schema. Figure 3.2 gives two predicate definitions. $p_{\text{DedicatedDonation}}(x)$ is true iff the donation $x$ has an identified recipient. The CFR 610.40 gives permissions not to test certain dedicated donations. The predicate $p_{\text{TestFor}}(x, y)$ is true iff the donation $x$ has been tested for the disease $y$. Predicates are type-checked against the schema, to ensure that checks on attributes have a well-defined meaning.

The syntax for predicates is based in part on the OCaml language. Once the predicate has been type-checked, it is compiled into a modal (description) logic (c.f. [9]).
The compilation target, which we call Graded Hybrid Logic (GHL), is a combination of graded modal logic [48] (which is modal logic with counting quantifiers) with the nominal expressions of hybrid logic [7]. The syntax allows for predicates which cannot be compiled into GHL, and in such cases, the user is warned that the predicate is treated as atomic for the purposes of inference.

The predicate definitions (in GHL) are then combined using connectives and temporal operators. We have implemented a SAT solver for GHL, which uses a standard tableaux algorithm with BDD-based backtracking. The solver is schema-aware, e.g., given a donation \( x \), if \( x \) is not a donation of blood, we will infer that \( x \) is a donation of source plasma. For the temporal operators, we use the tool LTL2BA [52], which produces a Buchi automaton representing all possible runs that an LTL formula accepts. We modified LTL2BA to interact with our solver for GHL, since LTL2BA assumes...
(** Is \( x \) a dedicated donation, i.e., intended for an identified recepient *)

predicate pDedicatedDonation \( (x : cDonation) \) =
  match x.fEndRecepient with
  Some\( (y) \) -> true
  | \_ -> false

(* Donation \( x \) is tested for the disease \( y \) *)

predicate pTestFor \( (x : cDonation) \ (y : cDiseaseConst) \) =
  exists z in x.fTests:
    z.fTestPurpose = y

Figure 3.2: Two predicate definitions. \( pDedicatedDonation(x) \) is true iff the donation \( x \) has an identified recepient. \( pTestFor(x, y) \) is true iff the donation \( x \) has been tested for the disease \( y \)

that propositions are atomic, while our propositions have notions of quantification.

### 3.4.2 Formalizing the Regulation

We have manually translated Sections 610.11 and 610.40 of the CFR into Refl. We given an overview of the formalization. Then, we evaluate Refl qualitatively via a discussion of four examples. Finally, we use the annotation of logical form (see Chapter 5) to provide some quantitative insights into the coverage of the logic.

**Overview:** Table 3.1 gives a breakdown of the various types of rules and formulas that were used in translating CFR 610.11 and 610.40 into Refl. The columns correspond to the number of obligations, permissions, other rules (which typically consist of list items), and intersentential references formalized using the \( \text{says} \) predicate. For example, in CFR 610.40, we formalized 12 obligations, 5 permissions, 6 list items
Table 3.1: Types of rules and formulas used in translating CFR 610.11 and 610.40 into RefL

<table>
<thead>
<tr>
<th>Document</th>
<th>Obligations</th>
<th>Permissions</th>
<th>Other</th>
<th>says</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFR 610.11</td>
<td>10</td>
<td>6</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>CFR 610.40</td>
<td>12</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

corresponding to disease names, and a total of 8 instances of the says predicate were used. There are a total of 42 rules (in both documents), and 21 instances of the says predicate. In other words, the says predicate is used once in every two rules. We now turn to a discussion of examples.

**Example 1:** Figure 3.3 shows the translation of CFR 610.40 Paragraph (a) into RefL. The predicate says is used twice in the precondition. says./.*list(z) is the translation of the phrase “the following communicable disease agents”, and is understood as follows. ./.* is an expression to select the sub-paragraphs. The predicate list(z) is a convenient way to represent lists of noun phrases. For example, the list item “(3) Hepatitis B virus” is represented as:

\[
(3) \text{.z.fName = Hepatitis(B) \mapsto list(z)}
\]

This rule lists the object representing the disease name “Hepatitis B”, and the utterance obtained can be used to access the object. The other says predicate in Figure 3.3 is \(\neg\text{says}_{[c,d]} \neg p\text{TestFor}(x, z)\), which is the translation of the clause “Except as specified in paragraphs (c) and (d) of this section”.

**Example 2:** Figure 3.4 shows the translation of CFR 610.40 Paragraph (d) into RefL. Paragraph (d) provides exceptions to the tests required in Paragraph (a), for “autologous donations”. Autologous donations are those that are made by a donor for herself, e.g., prior to surgery. Paragraph (d)(1) cancels this exemption under certain conditions, with the phrase “must be tested under this section”. A direct translation of Paragraph (d)(1) would give us an obligation of the form:

\[
((d)(1)).o : \varphi(x) \land \text{says}_{610/40/4} \text{test}(x) \mapsto \text{test}(x)
\]
(** Except as specified in paragraphs (c) and (d) of this section, you ... must test each donation ... for evidence of infection due to the following communicable disease agents:

1. Human immunodeficiency virus, type 1;
2. Human immunodeficiency virus, type 2;
3. Hepatitis B virus; ...**)

obligation \((x : \text{cDonation}) (z : \text{cDiseaseConst}) = \)

(** \(x\) is a donation of human blood or blood component, used in a product or device **) \(p\text{HumanDonation} x \land (p\text{UsedInProduct} x || p\text{UsedInDevice} x) \land \)

(** \(z\) is a disease agent listed below and no exception is granted by other laws **) \(\text{(says \((.*) (\text{listDisease} z)) \land !(\text{says \([c, d]\) !(pTestFor} x z))} \)

\(-->

(* If I, II and III hold, \(x\) must be tested for \(z\) *)

\(p\text{TestFor} x z\)

[1 rule \((z : \text{cDiseaseConst}) = z.\text{fName} = \text{HIV(Type1)} \rightarrow \text{list} z\]

[2 rule \((z : \text{cDiseaseConst}) = z.\text{fName} = \text{HIV(Type2)} \rightarrow \text{list} z\]

[3 rule \((z : \text{cDiseaseConst}) = z.\text{fName} = \text{Hepatitis(B)} \rightarrow \text{list} z\]

Figure 3.3: CFR 610.40 Paragraph (a) and its translation in logic

Where \(\varphi(x)\) checks some conditions on the donation. The obligation is understood as “If \(\varphi(x)\) holds and test(\(x\)) is required in this section, then \(x\) must be tested”. This translation of (d)(1) gives rise to a circular reference to Paragraph (a), leaving (a), (d), and (d)(1) ungrounded at the least fixed point. However, there exists a maximal fixed point where the donation is required to be tested. There are three options when faced with such a circular reference:

1. Assess conformance w.r.t. maximal fixed points

67
Figure 3.4: CFR 610.40 Paragraph (d) and its translation in logic. A direct translation of the phrase “tested under this section” in (d)(1) results in a circular reference to paragraph (a), leaving (a), (d), and (d)(1) ungrounded. The cancel predicate is used to break this cycle.

2. Issue a warning if the precondition of an obligation is ungrounded, and the postcondition is false

3. Break the cycle by introducing a new predicate
Figure 3.5: CFR 610.40 Paragraph (a), after the says predicate is compiled out.

The first option of computing maximal fixed points is infeasible, as there can be exponentially many. The second option is supported by our implementation. And, for the last option, we detect cycles and present them to the user, who can then introduce a new predicate. In Figure 3.4, the predicate cancel(x) is used to break
The cycle. We note that the cycle detection is carried out after we compile out the satisfiability tests (as in Section 3.3.2). This ensures that only cycles leading to ungrounded precondition are presented to the user.

**Example 3:** We now discuss some issues in sententially translating CFR 610.11 into Refl. Consider the following sentences:

(18) A general safety test for the detection of extraneous toxic contaminants shall be performed on biological products intended for administration to humans.

(19) The general safety test is required in addition to other specific tests prescribed in the additional standards for individual products in this subchapter, except that, the test need not be performed on those products listed in paragraph (g) of this section.

It is not possible to translate (18) and (19) separately into Refl. (18) conveys a requirement to perform “a general safety test”, and (19) reiterates this requirement, while providing exceptions. The problem arises because: (A) there is an incompleteness in (18), i.e., the exception is not provided, and (B) there is a redundancy in (19), i.e., the requirement is reiterated. To translate these sentences into Refl, we combine them into the following obligation:

(20) Except as specified in paragraph (g) of this section, a general safety test ... shall be performed on biological products ...

It is easy to translate (20) into Refl. We note that the permission given by (19) is restated in paragraph (g) of CFR 610.11.

**Example 4:** Consider the following sentences:

(21) The general safety test shall be performed as specified in this section...

a. The general safety test shall be conducted upon a representative sample of the product in the final container from every final filling of each lot of the product.

b. The duration of the general safety test shall be 7 days for both species, except that a longer period may be established for specific products in accordance with Sec. 610.9.

It is unclear how to translate (21) into Refl, due to the phrase “as specified in this section”. At one level, it is just an informational requirement, i.e., as long as
one conforms to the rest of the requirements (such as (21a) and (21b)), one conforms to (21). However, we will need to interpret (21a) and (21b) as quantifying over all general safety tests. For example, we can interpret (21a) as:

(22) If a general safety test is required to be performed on a product, then it shall be conducted upon a representative sample of the product...

It is easy to translate (22) into \textsc{Refl}. However, the question is whether there is a way to translate (21) into \textsc{Refl}, so that we can avoid such paraphrases. We believe that the central problem is the analysis of the phrase “the general safety test” – what is the test that is being referred to? There are several possibilities for formalization within \textsc{Refl}, and we leave an investigation to future work.

\textbf{Coverage:} The discussion of examples gives us a qualitative way to evaluate \textsc{Refl}. A question arises as to whether there is a more quantitative way to do it. The best test is, of course, to translate large bodies of regulation into \textsc{Refl}, and evaluate how it corresponds to the text. However, doing this would be quite time consuming. In Chapter 5, we describe an annotation of logical form on about 200 sentences of Section 610 of the CFR. While logical form is not quite the same as logic, our hope is that it can serve as an approximation to the logic, in order to evaluate it quantitatively. We take a preliminary step toward this end, here, by studying the distribution of clause and verb phrase modifiers.

<table>
<thead>
<tr>
<th>Number of Instances</th>
<th>Temporal and Conditional</th>
<th>Purpose</th>
<th>Other References to Laws</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>333</td>
<td>29.1%</td>
<td>28%</td>
<td>30.2%</td>
<td>12.7%</td>
</tr>
</tbody>
</table>

Table 3.2: Subtypes of clause and verb phrase modifiers.

Table 3.2 gives a breakdown of clause and verb phrase modifiers found in Section 610 of the CFR. Temporal and conditional modifiers (29.1\%) include, for example, "before", "after", "if", and "except as". The purpose modifiers (28\%) are signalled by "for". Other references to to laws (30.2\%) are introduced by "as", "under", and
“according to”. Note that the temporal and conditional (and exceptional) modifiers introduce references to laws as well. The other modifiers include cases which we have not categorized yet, e.g., “regardless”, “notwithstanding”.

The temporal and conditional/exceptional modifiers are adequately handled by Refl. However, the modifiers conveying references to laws need further investigation. Refl provides no special capabilities for purpose modifiers, but the notions of purpose in the CFR 610.40 are simple, and constant symbols suffice. We conclude that the Refl accommodates approximately 57.1% of modifiers (the first two columns), needs to be extended to accommodate another 30.2%, and the remaining 12.7% needs to be categorized. We note that 57.1% is an upper bound, because the temporal and conditional modifiers may include contain some other constructs that we do not know how to formalize.

3.4.3 Precomputing Satisfiability and Checking Traces

We compiled out the says predicate in the formal versions of CFR 610.11 and 610.40, using the techniques in Section 3.3.2. The compilation step took 14s with 160 satisfiability tests. Figure 3.5 shows the result of compiling out says from Paragraph (a). The original Refl translation is given in Figure 3.3. If we used a logic without inter-sentential references, the representation in Figure 3.5 would have to be created directly. This would involve translating phrases from different portions of the document, as the precondition in Figure 3.5 contains items from Paragraphs (a), (c), and (d). It would be difficult to preserve any correspondence to the document, and correct errors in the logical translation. By using the compilation procedure, we are able to track every subformula to the sentence from which it came.

Our current implementation of the trace-checker is static in the sense that the entire trace is stored on disk (in an NDBM database). The objects at each state belong to classes in a given schema. The regulation, which is type-checked against the same schema, is compiled using the techniques discussed in Section 3.3.2, and
evaluated at each state. We do not have any special optimizations for speed. The objects are stored as strings, and reparsed every time they are loaded into memory. The checker evaluates each obligation w.r.t. all variable assignments, loading into memory a single variable assignment at a time.

We now describe a preliminary evaluation of the checker. Our goal was to check if we could scale to traces with a large number of objects, rather than very long traces. To do this, we checked a number of synthetically generated (final) states for conformance. Given the schema for FDA CFR 610.40, we generated a set of donors by choosing random values for atomic attributes. For each donor we generate a set of donations again choosing attribute values at random. Each donation is randomly tested as follows: with $p = 0.3$ it is tested for all diseases with negative results, with $p = 0.3$ it is test for diseases with a random result, and otherwise it is not tested.

We evaluated performance of the checker against a number of states. The number of disease names was 8, and the number of donations varied. The time taken varied linearly with a number of donations. For states with 100, 1000, 5000, and 10000 donations the conformance check took 12s, 130s, 500s, and 1042s respectively. The performance suggests that the approach is practical for checking short traces. However, for longer traces, re-evaluating the regulation w.r.t. all variable assignments at each state can be expensive. We need to have notions of incrementality, to activate evaluation only when a new object enters the system.

### 3.5 Conclusions

We have described the checking process for ReFL, which allows statements to refer to others for conditions or exceptions. While ReFL give us a way to represent regulation directly, the evaluation of references during checking has high complexity. Algorithm 1 uses satisfiability tests of size polynomial in the number of objects. In Section 3.3, we described an empirically motivated assumption (the single copy property), which lets us replace satisfiability tests with tests of lower complexity. The case study using our
prototype implementation (Section 3.4) suggests that REFL offers a convenient way to formalize regulations, and that the approach is suitable for conformance audits of medium-sized traces. A discussion of examples and a study of the distribution of modifiers, in Section 3.4.2, led us identify avenues for further inquiry.

There are several optimizations that can be applied to our implementation to make it more efficient. For example, when a new object is added to a system or an attribute of an object changes, we would like to find the smallest set of variable assignments under which the regulation needs to be evaluated. Since there are pointers between objects, potentially any object that is pointed to or reachable from a modified object may need re-evaluation. We are currently exploring ways to use the schema and the structure of regulatory statements to determine what to re-evaluate.
Chapter 4

Permission to Speak

4.1 Introduction

In this chapter, we turn to the analysis of obligation and permission, and their interaction with saying. We motivate and develop a formalism by considering problems in – (a) access control, (b) legal powers, and (c) conformance checking. We begin with a discussion of prior work, and the problems that we are addressing.

Access Control: Access control is an important problem in trust management systems. Informally, a trust management system involves a set of actors or principals, and a set of controlled or regulated actions, e.g., accessing medical information, or downloading a song. The goal of such a system is to administrate requests to perform actions. Trust management systems are commonly decomposed into two (interacting) components [1]: (a) authentication - determining the source of a request, and (b) access control - determining whether a request is permitted according to a policy. Abadi et al. [3] cast access control as a problem for logic. We assume as given an action \((p)\), which is controlled by a principal \((A)\), and a request to perform \(p\) from a principal \((B)\). Access is granted if we can prove, using \(A’s\) policy, that \(A\) says that \(B\) is permitted to perform \(p\). In access control logics, such as [1–3, 50, 51], says is treated as a (modal) operator. However, the use of an operator for permission has
not been explored.

The concept of representation is prevalent in access control policies, and it forms the central focus of this work. Representation arises in situations where a principal is held to declarations made on her behalf (cf. [53]). For example, consider a scenario where a software company authorises project managers to permit their team members to access the production server. If a project manager says that a team member is permitted to access the server (on behalf of the company), we conclude that the company says that the team member is permitted to access the server. In such a scenario, project managers represent the company on permitting access to the server. All access control logics provide principals with the capability to let other principals represent them on statements. In the example above, the company would say, in its policy, that “Project managers represent the company on permitting team members to access the production server”. The manner in which such a policy is formally expressed depends on the logic, and we will discuss a few choices in later sections.

In this chapter, we argue for an explicit account of permission in a logic for access control. We motivate and develop a logic that combines saying and permission, using an axiom that permits a principal to speak on behalf of another. The combination leads us to a novel account of representation. In the logics of saying, where there is no notion of permission, representation is accommodated using variants of the hand-off axiom [1]. Abadi [1] pointed out some problematic interactions between the hand-off axiom and classical reasoning, which we will discuss in detail in Section 4.2.2. The use of permission provides a way to avoid these problems. An explicit account of permission leads naturally to an explicit account of obligation, which in turn leads us to examine legal powers and conformance checking. We now introduce these topics.

Legal Power: Representation is a special case of the broader concept of legal power. Hohfeld, in his seminal work, defined the concept of power as follows [69, Page 44]:

A person (or persons) may be said to have the power to effect a change in legal relations, if the change in legal relations results from some super-
added facts that are under his volitional control.

We decompose this definition into three components, to give the main intuitions for our approach:

1. The description of the power - A principal (A) grants the power of representation to another principal (B) on certain statements, if A says that B is permitted to issue those statements (on her behalf).

2. The “superadded facts” by which a power is exercised - B exercises the power of representation by issuing statements on behalf of A.

3. The change in legal relations - If A grants the power of representation to B, and B exercises this power, then we will infer that the statement issued by B is issued by A as well.

The logical analysis of power has been of interest for several years [53, 74, 75, 79, 95]. Our approach is related to two lines of research. With regard to the description of power (Item 1 above), Lindahl [95, Part II] (see also [74]) suggested that various notions of power can be distinguished by nesting obligations and permissions with an action modality. Saying is our analog of the action modality. With regard to the change in legal relations (Item 3 above), Jones and Sergot [75] and Gelati et al. [53] describe general frameworks to reason about situations where an act by a principal counts as a means to create a state of affairs within an institution. We consider a restricted scenario where a statement by one principal counts as an identical statement made by another principal. However, in [53, 75], the concept of counts as is taken to be the description of power itself, and it is independent of the concept of permission. The dependence of power on permission, in our approach, leads us to a novel analysis of recursive notions of power, e.g., “empowerment to empower”. We discuss the differences in Section 4.4.2.

**Conformance Checking:** The problems of access control, representation, and power arose for us while extending our approach to privacy regulation. We briefly reintro-
duce the problem of conformance here, and discuss how the ideas in Chapter 2 relate to this chapter in Section 4.2.4. In conformance, one is interested in checking whether the operations of organizations obey a policy. We are given a policy and a description of an organization’s operations (as a state or trace). An organization \((A)\) is confromant if we can prove that for all \(p\), if the policy \textit{says} that \(A\) is \textit{required} or \textit{obligated} to do \(p\), then \(A\) does \(p\). The design of logics for conformance, notably deontic logic, has been of interest for several years, and we refer the reader to [73, 111] for a broad perspective. In recent years, the focus has been more on tailoring logics for the regulations at hand, and examples include business contracts [4, 54, 56, 60, 62, 88] and health-care regulations [25, 44]. Our focus in this work is on how \textit{power} interacts with the question of \textit{conformance}.

\textbf{Contributions and Outline:} In this chapter, we motivate and design a formalism that combines \textit{saying} and \textit{permission}, with applications to access control and conformance. The combination yields benefits to both applications:

1. For access control, we propose a new decidable axiomatization which accommodates delegation [3, 93] and “speaking for” [2, 3, 50]. Our approach overcomes the problematic interactions with classical reasoning, pointed out by Abadi [1]. “Speaking for” and delegation are obtained as consequences of an axiom that permits a principal to speak on behalf of another.

2. For conformance, the proposed axiomatization is used to reason about declarative powers [53], by nesting saying with obligation and permission. We obtain a novel analysis of recursive notions of power, e.g., “empowerment to empower”. Conformance, as the satisfaction of obligations, is shown to be decidable.

In Section 4.2, we give a detailed motivation and background for our approach in three parts. First, we consider representation in access control, under which we include delegation [3, 93] and “speaking for” [1, 3, 50]. Second, we discuss examples of powers conveyed by nested permissions, and compare our approach to the \textit{counts}
as frameworks for power [53, 75]. And, finally, we discuss how we integrate the work here with the logic from Chapter 2.

Section 4.3 develops a logic in the form of two interacting components. *The inference component* determines what has been said, and involves the choice of appropriate axioms [1, 3, 51]. We introduce two axioms to characterize the interaction between saying and permission. The decidability and complexity of the resulting logic are established. *The saying component* is used to create new utterances. For this component, we extend ReFl, which we developed in Chapter 2. The modularization allows us to use restricted forms of quantification while preserving decidability of access control and conformance. We also prove a non-interference property which is crucial for the distributed policies that arise in access control. While the logic is expressive, the expressive power comes at the price of high complexity. The decision problem is NEXPTIME-complete, i.e., complete for non-deterministic exponential time. We identify an expressive fragment of the logic, which generalizes the logic programming approaches, and has a polynomial time decision procedure.

In Section 4.4, we discuss our formalism in the context of related work. We consider access control examples, and conformance in the presence of nested obligations and permissions. We also connect our work to speech act theories. Section 4.5 concludes.

### 4.2 Permission to Speak

In this section, we motivate the explicit use of saying and permission in a formal language for policy. We begin, in Section 4.2.1, by introducing the syntax and basic axioms of a logic with *saying* and *obligation*, in order to facilitate a precise comparison with prior work. Section 4.2.2 considers the problem of representation in access control, under which we include delegation [3, 93] and “speaking for” [1, 3, 50]. In Section 4.2.3, we discuss examples of powers conveyed by nested permissions. We compare and contrast our approach with the *counts as* approaches to power. Finally,
we discuss how we integrate the work here with the logic developed in Chapter 2, to clarify some methodological decisions (Section 4.2.4).

4.2.1 Preliminaries

We begin by defining the syntax of a logic with *saying* and *obligation*, which we will revise in Section 4.3.2:

**Definition 4.1** (Syntax). Given sets $\Phi$ (of propositions), object names $O$, a finite set of identifiers $ID$, and a function $l : O \rightarrow 2^{Id}$, the language $L(\Phi, X, O, l, ID)$, abbreviated as $L$, is defined as follows:

$$\varphi ::= \Phi \mid \varphi \land \varphi \mid \neg \varphi \mid \text{says}_{Id_y} \varphi \mid \text{O}_y \varphi$$

where, $y \in O$, and $Id_y$ is a non-empty subset of $l(y)$. In addition, we assume that for all distinct $y, y' \in O$, $l(y) \cap l(y') = \emptyset$ and $l(y) \neq \emptyset$, i.e., the assigned identifiers are disjoint, and non-empty.

Disjunction $\varphi \lor \psi = \neg (\neg \varphi \land \neg \psi)$ and implication $\varphi \Rightarrow \psi = \neg \varphi \lor \psi$ are derived connectives. $\bot$ (false) is an abbreviation for $p \land \neg p$ for some $p \in \Phi$, and $\top$ (true) is $\neg \bot$.

$\text{O}_y \varphi$ is read as “$\varphi$ is obligatory for $y$”. Permission is defined as the dual of obligation, i.e., $\text{P}_y \varphi = \neg \text{O}_y \neg \varphi$. For example, “$B$ is permitted to delete the file (del)” is expressed as $\text{P}_B \text{del}$.

$\text{says}_{Id_y} \varphi$ is read as “$y$ says $\varphi$ via the laws labeled with identifiers in $Id_y$”. We will also parphrase this as “$y$ says $\varphi$”. We build on ideas in Chapter 2. Principals speak by introducing *laws*. Given a principal $A \in O$, a *law of A* is a statement $\varphi \in L$ associated with an identifier $id \in l(A)$. Let us consider an example. Suppose a system administrator $A$ has the following statement in her policy:

$$\text{(1}_A) \text{ B is permitted to delete the file}$$
1_A \in l(A) \text{ is an identifier associated with the statement. Formally, we represent the law as:}

\text{says}_{1_A}(P_B \text{del})

In other words, “\text{A} \text{ says } \text{B} \text{ is permitted to delete the file (via the law labeled } \{1_A\} \text{).}”

We will need the machinery developed in Chapter 2 to accommodate conditional laws, and in particular, the cases where \text{says} appears in the antecedent of the condition. We revisit these issues in Section 4.3.3.

How do we evaluate statements in the logic? Most logics that are used in computer science have a compelling semantics. For example, in linear temporal logic, statements are evaluated against a trace of states that arise during computation. In some scenarios, obligations have a good semantics, e.g., \varphi \text{ is obligated by a program if } \neg \varphi \text{ causes the program to raise an exception. However, it is hard to give a semantics for } \text{saying} \text{ (cf. [1]).}

What does it mean for a principal to say \varphi? In the access control setting, it means that a principal has issued a digital certificate (or a law) declaring \varphi. However, a certificate or law is a syntactic notion, and as a result, we are interested in what we can prove from the laws of a principal. For this reason, access control logics are commonly designed axiomatically. Following Garg and Abadi [50], we use semantics as a tool, to show that a statement is not provable. But, it is the theorem proving questions that are of central interest. We give the basic axioms and rules for \text{saying} below (those which are common to all access control logics), and defer the introduction of further axioms to later sections:

**A1** All substitution instances of (classical) propositional tautologies

**A2** \text{says}_{l(A)}(\varphi \Rightarrow \psi) \Rightarrow (\text{says}_{l(A)} \varphi \Rightarrow \text{says}_{l(A)} \psi) \text{ for all } A \in O

**R1** From \vdash \varphi \Rightarrow \psi \text{ and } \vdash \varphi \text{ infer } \vdash \psi

**R2** From \vdash \varphi \text{ infer } \vdash \text{says}_{l(A)} \varphi \text{ for all } A \in O
We say that $\varphi$ is provable, denoted $\vdash \varphi$, if $\varphi$ is an instance of $A1$ or $A2$, or follows from the axioms using the rules $R1$ and $R2$. $\psi$ is provable from a set of formulas $\{\varphi_1, ..., \varphi_n\}$, denoted $\varphi_1, ..., \varphi_n \vdash \psi$, if $\vdash (\varphi_1 \land ... \land \varphi_n) \Rightarrow \psi$.

$A1$ and $R1$ give us classical propositional reasoning. The distribution axiom ($A2$) states that $\textit{says}$ distributes over implications ($\Rightarrow$), and $R2$ is the generalization rule common to all normal modal logics. The appropriateness of $R2$ for $\textit{says}$ is not obvious, as principals do not explicitly say all the theorems in the language. However, it can be understood as the willingness of a principal to admit all theorems as her statements, so that we can infer the consequences of her statements using $A2$ (since all theorems are of the form $\varphi \Rightarrow \psi$).

We now return to the example where the administrator $A$ has permitted $B$ to delete the file, via the law $1_A$. Suppose a principal $C$ requests that the file be deleted. The access control problem is to decide whether we can prove $\text{says}_{l(A)}(P_B\text{del})$ from $A$’s laws, i.e., whether “$A$ says that $C$ is permitted to delete the file”. Suppose $A$’s policy consisted of the single statement $(1_A)$ above, i.e., $l(A) = \{1_A\}$, it can be shown that:

$$\text{says}_{l(A)}(P_B\text{del}) \not\vdash \text{says}_{l(A)}(P_C\text{del})$$

And, hence, $C$ is not permitted to delete the file. We now turn to a discussion of the literature to motivate our choice of additional axioms.

### 4.2.2 Representation in Access Control

All access control logics give a principal the ability to let another principal make statements on her behalf. As an example (based on [50]), consider a file access scenario, where an administrator ($A$) has control the operation of deleting files shared

---

1We note that an (appropriate) non-normal modal logic (cf. [29]) may be a better basis for $\textit{says}$, if we wish to capture the notion of explicitly $\textit{says}$. We have chosen the minimal normal modal logic $K$ because it is adopted by most access control logics in the literature (perhaps by default). The $\textit{counts as}$ approaches to power [53, 75] use non-normal operators. From an application perspective, we do not have motivation for one choice over the other, and leave an investigation to future work.

82
by groups of principals. When there are many shared files in the system, A cannot personally handle all requests. Suppose that the administrator authorises the leader of a group (B) to decide when a particular file is to be deleted (del). In this scenario, we say that B represents A on del, and we wish to conclude that if says\(_l(B)\) (del), then says\(_l(A)\) (del).

How do we accommodate this inference? A naive approach is to introduce:

\[
\psi \equiv \text{says}_l(B) (\text{del}) \Rightarrow \text{says}_l(A) (\text{del})
\]

into A’s policy. However, such statements create an access control risk, because \(\psi\) could be introduced by B, thereby giving B the ability to decide whether any file is to be deleted.

To address this security risk, a principal A is only allowed to introduce statements of the form says\(_l(A)\) \(\psi\). Additional machinery (usually an axiom) is needed to accommodate representation. Abadi [1] discusses several alternatives, involving variants of the hand-off axiom:

**AH** says\(_l(A)\) (\(\phi \Rightarrow \text{says}_l(A) \psi\)) \(\Rightarrow\) (\(\phi \Rightarrow \text{says}_l(A) \psi\))

B represents A on del is expressed as:

\[
\varphi = \text{says}_l(A) (\text{says}_l(B) (\text{del}) \Rightarrow \text{says}_l(A) (\text{del}))
\]

Using the hand-off axiom, we conclude that \(\varphi \vdash \text{says}_l(B) (\text{del}) \Rightarrow \text{says}_l(A) (\text{del})\), using **R1**. However, the hand-off axiom has displeasing consequences in classical logics:

**Proposition 4.1** (Abadi [1]). The following is provable:

\(\vdash \neg \text{says}_l(B) \perp \Rightarrow (\text{says}_l(B) \varphi \Rightarrow \varphi)\)

**Proof.** We proceed as follows:

1. \(\vdash \varphi \Rightarrow (\neg \varphi \Rightarrow \text{says}_l(B) \perp)\) (using **A1**)

2. \(\vdash \text{says}_l(B) (\varphi \Rightarrow (\neg \varphi \Rightarrow \text{says}_l(B) \perp))\) (from (1) using **R2**)
3. $\vdash \text{says}_{\ell(B)}(\varphi) \Rightarrow \text{says}_{\ell(B)}(\neg \varphi \Rightarrow \text{says}_{\ell(B)} \bot)$ (from (2) using A2 and R1)

4. $\vdash \text{says}_{\ell(B)} \varphi \Rightarrow (\neg \varphi \Rightarrow \text{says}_{\ell(B)} \bot)$ (from (3) using AH, A1 and R1)

5. $\vdash \neg \text{says}_{\ell(B)} \bot \Rightarrow (\text{says}_{\ell(B)} \varphi \Rightarrow \varphi)$ (from (4) using A1 and R1)

By Proposition 4.1, if we impose the (reasonable) restriction that principals do not utter contradictions, then we are forced to accept every statement as truth! Halpern and van der Meyden [64] discuss the same problem in the context of a logic for local names:

We certainly want to be able to use the logic to say that if a principal’s statements are not blatantly inconsistent, then certain conclusions follow. While Halpern and van der Meyden [64] address the issue of naming, they exclude notions of representation from their framework. In the context of access control, the solution to the problematic inferences has been to move to an intuitionistic logic [2, 50, 51]. The last step in the deduction above, i.e., from (4) to (5) in the proof of Proposition 4.1, is blocked, since implication is not defined in terms of disjunction in intuitionistic logic. We note that the inference in (4) holds in intuitionistic systems [2, 50, 51], i.e., if a principal makes a false statement, then her statements are inconsistent. Although this does not seem to cause problems in applications, we believe that it is counterintuitive. Why should a mistaken statement or a lie make a principal’s statements inconsistent? Neither (4) or (5) holds in the logic that we develop.

We suggest that the problem is not with classical reasoning, but with the hand-off axiom. The key idea is to reformulate the axiom using the interaction between saying and permission. We now introduce the reformulated version of the axiom, followed by a discussion of its benefits.

We say that $B$ represents $A$ on del, if $A$ says that $B$ is permitted to say del. More formally, the statement $\text{says}_{\ell(A)}(P_B(\text{says}_{\ell(B)} \text{del}))$ is added to $A$’s policy, where
\( \mathcal{P}_B(\text{say}_{B}(\text{del})) \) is read as “\( B \) is permitted to say \( \text{del} \)”. The following are equivalent versions of the axiom of representation:

**AR** If \( A \) says that \( B \) is permitted to say \( \phi \), then if \( B \) says \( \phi \), \( A \) says \( \phi \)

\[
\text{AR} \quad \text{say}_{A} \left( \mathcal{P}_B \left( \text{say}_{B}(\phi) \right) \right) \Rightarrow \left( \text{say}_{B}(\phi) \Rightarrow \text{say}_{A}(\phi) \right)
\]

The axiom of representation is intended for a particular sense of speaking/saying, i.e., speaking on someone’s behalf. This sense of saying is the usual one in access control. To simplify matters, we do not explicitly represent the principal on behalf of whom a statement is being made.

“Speaking for” \([2, 3, 50]\) is a case of representation when one principal represents another on all statements. If \( B \) speaks for \( A \), we wish to conclude \( \text{say}_{B}(\phi) \Rightarrow \text{say}_{A}(\phi) \) for all \( \phi \). “Speaking for” has a compelling definition in our approach. We say that \( B \) speaks for \( A \) if \( A \) permits \( B \) to say anything (\( \bot \)) on her behalf, i.e., \( \text{say}_{A}(\mathcal{P}_B(\text{say}_{B}(\bot))) \).

A novelty in our approach is that “speaking for” and hand-off are both obtained as a consequence of the axiom of representation. In \([2, 3, 50]\), “speaking for” and hand-off are not related, i.e., the former involves an algebra over principals or second-order quantification, and the latter is obtained using an axiom (which implies hand-off). This suggests that the representation axiom is quite different from the hand-off axiom. It is tempting to relate the representation axiom to a restricted version of hand-off:

\[
\bullet \quad \text{say}_{A}(\text{say}_{B}(\phi) \Rightarrow \text{say}_{A}(\phi)) \Rightarrow (\text{say}_{B}(\phi) \Rightarrow \text{say}_{A}(\phi))
\]

However, even for this restricted case, we do not know of a complete semantics for hand-off, which makes it difficult to show that a statement is not provable (Abadi et al. \([3]\) observe similar difficulties).\(^2\) We believe that the representation axiom is a

\(^2\)Garg and Abadi \([50]\) provide a complete semantics for a version of the hand-off axiom which implies but is not equivalent to \( \textbf{AH} \). However, this version of the hand-off leads to the problematic inferences discussed in Proposition 4.1.
persuasive alternative to hand-off, because it yields a decidable logic with a complete semantics, and more importantly, it has an intuitive interpretation.

A restricted version of the axiom of representation has been proposed by Becker et al. [16], in the context of the authorization language SECPAL. In SECPAL, representation is restricted to atomic predicates, and hence, “speaking for” cannot be accommodated. Moreover, the relationship between permission and obligation is not explored, and “permission to say” (called “can say” in SECPAL [16]) is treated as a primitive construct. Our formalism generalizes SECPAL, to accommodate both “speaking for” and obligation. In Section 4.3.6, we show that an extension to SECPAL fragment of the logic is decidable in polynomial time, thereby preserving SECPAL’s computational benefits. We now discuss further motivation for our approach.

4.2.3 Power and Nested Permissions

In this section, we consider examples of powers that arise via nested permissions. We compare and contrast our approach to the counts as approaches to power [53, 75].

We begin by discussing our approach to nested permissions. Consider the following statement: “A hospital (H) permits a patient (A) to permit her mother (B) to access her information”. We will rephrase the permission as follows: H says that A is permitted to say that B is permitted to access her information. Formally, this is expressed as: \(\text{say}_{H}(\mathcal{P}_{A}(\text{say}_{A}(\mathcal{P}_{B}\text{access})))\). If A does indeed permit access to her mother (\(\text{say}_{A}(\mathcal{P}_{B}\text{access})\)), we will conclude \(\text{say}_{H}(\mathcal{P}_{B}\text{access})\) using the axiom of representation, i.e., H permits access to B. As a result, nested permissions are related to representation, i.e., “H permits A to permit B to do \(\varphi\)” iff “A represents H in permitting B to do \(\varphi\)”.

We now turn to the analysis by Gelati et al. [53]. To simplify presentation, we describe their approach using the notation that we have already introduced. In [53], declarative power, which includes representation, is defined formally in terms of a counts as operator/connective:

86
(P1) $DP^H_A(\varphi) = \text{CountsAs}(\text{says}_{l(A)} \varphi, \text{says}_{l(H)} \varphi)$

$DP^H_A(\varphi)$ is read as “$H$ grants $A$ the power to declare $\varphi$ on its behalf”. And, $\text{CountsAs}(\text{says}_{l(A)} \varphi, \text{says}_{l(H)} \varphi)$ is read as “$A$ saying $\varphi$ counts as $B$ saying $\varphi$”. The logic of *counts as* [53, 75] has broad applicability, and a detailed exposition is well beyond the scope of this thesis. For present purposes, it suffices to note that a version of the following is provable:

(P2) $\vdash \text{CountsAs}(\text{says}_{l(A)} \varphi, \text{says}_{l(H)} \varphi) \Rightarrow (\text{says}_{l(A)} \varphi \Rightarrow \text{says}_{l(H)} \varphi)$

Returning to our example of nested permissions, using (P1) and (P2), we can show that:

(P3) $\vdash DP^H_A(P_B\text{access}) \Rightarrow (\text{says}_{l(A)}(P_B\text{access}) \Rightarrow \text{says}_{l(H)}(P_B\text{access}))$

And, (P3) plays the role of the representation axiom. As a result, our approach is quite similar to that of Gelati et al. [53], when there is one level of nesting. However, differences arise when we consider one more level of nesting.

Suppose $H$ says that $A$ is *empowered* to *empower* $B$ to permit $C$ to access her information. Note that *empowerment* can be paraphrased as *permission to say* in our approach, and the analysis would proceed analogously to the previous case. Gelati et al. [53] express this empowerment to empower as: $DP^H_A(DP^H_B(P_C\text{access}))$. Let $\varphi = P_C\text{access}$. Using (P1) and (P2), we obtain:

(P4) $\vdash DP^H_A(DP^H_B(\varphi)) \Rightarrow (\text{says}_{l(A)}(DP^H_B(\varphi)) \Rightarrow \text{says}_{l(H)}(DP^H_B(\varphi)))$

Given $DP^H_A(DP^H_B(\varphi))$, if $A$ exercises this power by empowering $B$ to declare $\varphi$, we will conclude that $\text{says}_{l(H)}(DP^H_B(\varphi))$, i.e., $H$ says that $B$ is empowered to declare $\varphi$. However, the following is not provable:

(P5) $\not\vdash \text{says}_{l(H)}(DP^H_B(\varphi)) \Rightarrow (\text{says}_{l(B)} \varphi \Rightarrow \text{says}_{l(H)} \varphi)$

Thus, $B$ cannot exercise the power in the same way as $A$. From a technical perspective, there are a variety of ways to augment [53, 75] to accommodate these inferences.
From an intuitive standpoint, we believe that this asymmetry, between primary and recursive powers, arises because the *counts as* operator is taken to be the description of power itself. In our approach, the description of power (using nested permissions) is separate from the changes that arise (via the representation axiom), leading to a symmetric treatment of primary and recursive powers.

### 4.2.4 Exceptions and Doesn’t Say

In Chapter 2, building on Reiter’s Default Logic [126] and Kripke’s theory of truth [85], we expressed laws using labeled conditional statements of the form:

\[(id) \, \varphi \mapsto \psi\]

Our informal interpretation of such statements was “If \( \varphi \) is true, then the regulator *says* \( \psi \) via the law labeled (id)”, where “id” is an identifier for the law. This interpretation of rules has the flavor of the *counts as* connective [53, 75], i.e., \( \varphi \) *counts as* a statement of \( \psi \) from the regulator. Now, we can consider statements of the form:

\[(id1) \, \text{The regulator does not say } \psi \text{ via the law labeled (id)} \mapsto \psi'\]

In other words, “If the regulator *does not say* \( \psi \) via the law labeled (id), then the regulator *says* \( \psi' \) via the law labeled (id1)”. *Does not say* is useful in expressing exceptions to laws, and the law labeled (id) would serve as an exception to the law labeled (id1). We discuss an example in Section 4.3.3.

Exceptions make regulations non-monotonic, in the sense that adding a new exception would prevent certain conclusions that were drawn before. There are also well-established reinterpretations of non-monotonic logics as modal logics, and here, we refer the reader to some classic works on autoepistemic logic [63, 113]. Given these connections, an important question that arises is whether the underlying logic for *saying* should be *non-monotonic*. The approach we take in this work is to start with a *monotonic* logic with *saying*, *obligation*, and *permission* (Section 4.3.2), and
then integrate it into a non-monotonic framework (Section 4.3.3). The idea is that the non-monotonic component resolves exceptions, giving us a consistent set of statements on which to base access control and conformance decisions. This aspect of our approach was motivated purely by methodological convenience, and sufficed for the regulations at hand. A proper non-monotonic treatment of nested modalities is a challenging problem (see [63]), and we leave an investigation to future work.

4.3 A Logic for Access Control and Conformance

In this section, we develop a logic in the form of two interacting components – (a) the inference component, which involves the choice of appropriate axioms, and (b) the saying component, which is used to represent policies. Figure 4.1 shows the interaction between the components of the access control system. There are two kinds of actions of interest – (1) operational acts, e.g., downloading a song, and (2) speech acts. The operational acts are described using a state, which contains the interpretation of predicates, and the speech acts are described using laws.

A principal speaks by introducing laws. In Figure 4.1, the principals $A$ and $B$ introduce the laws 1 and 2 respectively. The laws are evaluated using the axioms to produce a set of utterances, i.e., what the principals say via their laws. A set of laws can be thought of as a logic program, and utterances as the extensions that result from the program (via a fixed point computation). Once we have the utterances, there are...
several decision problems of interest. The access control problem is to decide whether a request is permitted by the set of utterances. The conformance problem is to decide whether operational and speech acts satisfy the obligations imposed by the utterances, and if they do not, violations are reported.

In Section 4.3.1, we introduce an example from privacy regulation, which we will use to illustrate the various definitions. Section 4.3.2 is an overview of the inference component. We describe (axiomatically) a logic with two modalities – saying and obligation. In Section 4.3.3, we adapt the formalism in [44] for the saying component. We extend [44] in two ways. First, we prove a non-interference property which is crucial for the distributed policies that arise in access control (Section 4.3.4). Second, we show that conformance, in the presence of nested obligations and permissions, is decidable (Section 4.3.5). In Section 4.3.6, we describe a fragment of the logic, motivated by the access control literature, for which provability is decidable in polynomial time.

4.3.1 Example

We will use an example from the Health Insurance Portability and Accountability Act (HIPAA) (cf. [25]), to illustrate the various definitions. HIPAA regulates the uses and disclosures of patient health information, and provides a natural test bed for investigating both access control and conformance. The following example is intended to illustrate several subtleties involved in reasoning about rights:

(23) A patient has the right to view his records that are maintained in a designated record set, except for:
   
   a. Psychotherapy notes.
   
   b. Records compiled for a legal proceeding.
   
   c. ...

There are three (types of) principals associated with this right:
• The regulatory authority behind HIPAA, which enforces the right.
• Patients, who can exercise the right, and
• Principals who maintain records of the patient, and have to conform to the right.

Section 4.3.2 is concerned with how to formally express the phrase “has the right”. In Section 4.3.3, we deal with the exceptions. Sections 4.3.4 and Section 4.3.5 consider the access control and conformance aspects.

4.3.2 The Inference Component – Axioms

In this section, we develop a predicate logic with two modalities saying and obligation. We allow formulas with free variables, but no quantifier over objects. The quantification over objects is carried out in the process of saying (Section 4.3.3), which uses provability in the propositional subset of the language defined here. We begin by defining the syntax:

**Definition 4.2** (Syntax). Given sets $\Phi_1, ..., \Phi_n$ (of predicate names), countable sets of object names $O$, principal names $O_P \subseteq O$, variables $X$, variables for principals $X_P \subseteq X$, identifiers $ID$, and a function $l : O_P \rightarrow 2^{ID}$, the language $L(\Phi_1, ..., \Phi_n, O, O_P, X, X_P, l, ID)$, abbreviated as $L$, is defined as follows:

\[
\begin{align*}
\varphi_y & := \alpha \mid \varphi_y \land \varphi_y \mid \neg \varphi_y \mid \text{says}_{Id_y} \psi \\
\psi_y & := \varphi_y \mid \psi_y \land \psi_y \mid \neg \psi_y \mid \text{O}_y \varphi_y \\
\varphi & := \varphi_y \text{ (for all } y \in X_P \cup O_P) \\
\psi & := \psi_y \text{ (for all } y \in X_P \cup O_P)
\end{align*}
\]

where, $y \in X_P \cup O_P$, and $\alpha$ generates atomic predicates of the form $p(z_1, ..., z_j)$ with $p \in \Phi_j$ and $(z_1, ..., z_j) \in (X \cup O)^j$. In addition, $\emptyset \subseteq Id_y \subseteq l(y)$ if $y \in O_P$ and $Id_y = l(y)$ otherwise ($y \in X_P$). We assume that for all distinct $A, B \in O_P,$
l(A) \cap l(B) = \emptyset \text{ and } l(A) \neq \emptyset, \text{ i.e., the assigned identifiers are disjoint, and non-empty.}

The set of formulas generated by each BNF rule are referred to as \( L_\phi, L_\psi \), and \( L_\psi \) respectively, and \( L = L_\phi \cup L_\psi \).

Disjunction \( \varphi \lor \psi = \neg(\neg\varphi \land \neg\psi) \) and implication \( \varphi \Rightarrow \psi = \neg\varphi \lor \psi \) are derived connectives.

There is a set of object names \( O \) with a distinguished set \( O_P \subseteq O \) called principals. Principals include individual persons, such as patients and doctors, and institutions, such as hospitals and regulatory authorities. We use upper case letters for principals, e.g., \( A, B \). Other named objects (\( O - O_P \)) include entities with no associated notion of agency, e.g., medical records and songs. We use lower case letters for these objects, except for the letters \( \{x, y, z\} \) which are reserved for variables. It is useful to divide the objects in \( O - O_P \) further into sorts, but we avoid it to simplify notation. Variables are divided into two sorts as well, i.e., all variables \( X \), and variables for principals \( X_P \). In a slight abuse of notation, we will use the symbols for variables, i.e., \( x, y \) and \( z \), to stand for a generic element in \( X \cup O \) or \( X_P \cup O_P \).

\( O_y \varphi \) is read as “\( \varphi \) is obligatory for the principal \( y \)”. Permission is defined as the dual of obligation, i.e., \( P_y \varphi = \neg O_y \neg \varphi \).

The saying operator is understood as follows. Principals speak by introducing identified laws, as shown in Figure 4.1. The function \( l \) assigns non-empty and disjoint sets of identifiers to each principal, and for example, \( l(A) \) denotes the set of identifiers for laws introduced by the principal \( A \in O_P \). \( \text{says}_{Id_y} \varphi \) is read as “\( y \) says \( \psi \) via the laws \( Id_y \)” . In the case where \( Id_y = l(y) \), \( \text{says}_{l(y)} \psi \) is read as “\( y \) says \( \psi \) via her laws”, or briefly “\( y \) says \( \psi \)” .

We give some examples to clarify the notation for identifiers. Given \( A \in O_P \), let \( l(A) = \{1, 2\} \). The formulas \( \text{says}_{l(A)} \varphi \) and \( \text{says}_{\{1,2\}} \varphi \) are identical. In many examples, we will have need only for the notation \( \text{says}_{l(A)} \varphi \). \footnote{The assumptions about assignment of identifiers are purely (and hopefully) for clarity. We do not
will be used to accommodate exceptions to laws (Section 4.3.3). Exceptions are often conveyed by phrases such as “except as specified in Section 120 of HIPAA” [25, 44], and a subset of identifiers would correspond to the laws in “Section 120 of HIPAA”. Given a variable over principals $x \in X_P$, we will only use the notation $\text{say}_{l(x)} \varphi$. This is useful, for example, to grant powers to a class of principals, e.g., patients of a hospital.

We now mention a peculiarity of Definition 4.2. The BNF rules ensure the alternation of obligation and saying modalities, e.g., $\mathcal{O}_y \text{say}_{l(y)} \mathcal{O}_z \varphi \in L$, but $\mathcal{O}_y \mathcal{O}_z \varphi \notin L$. Following von Wright [147], we understand obligations as applying to actions and their consequences. The language $L_{\varphi_y}$ (obtained from the first BNF rule) is used to describe actions of a principal $y$ – (a) atomic actions, (b) combinations of actions (using connectives), or (c) saying, which is (a consequence of) a speech act. An obligation is an opinion, which is created via a speech act, but is not an act by itself. These restrictions are similar in spirit to the logics of power [53, 74, 75, 95].

The statements in $L$ will be used in the inference component of access control, i.e., to determine what has been said. In other words, we will be given a set of utterances $U$ and a question $\psi$, and we need to determine whether $\psi$ is provable from $U$. We focus on provability for the propositional subset of $L$, i.e., without variables. The propositional subset of $L$ has the modalities $\text{say}_{l(A)} \varphi$ and $\mathcal{O}_A(\varphi)$ (for all $A \in O$ and $\text{Id}_A \subseteq l(A)$).

We adopt the axiomatization in Figure 4.2. $\textbf{A1}$ and $\textbf{R1}$ give us propositional reasoning. $\textbf{A2}$ and $\textbf{R2}$ are common to both saying and obligation. $\textbf{A3}$ and $\textbf{A4}$ are specific to saying and obligation respectively. Finally, $\textbf{A5}$ and $\textbf{A6}$ describe the interaction between the two modalities.

The notion of provability is of crucial interest. We say that $\varphi$ is provable (denoted $\vdash \varphi$), if $\varphi$ is an instance of the axioms $\textbf{A1-A6}$ or follows from the axioms using the
A1. All substitution instances of propositional tautologies.

A2. \( Q(\varphi \Rightarrow \psi) \Rightarrow (Q(\varphi) \Rightarrow Q(\psi)) \) (for all modalities \( Q \))

A3. says\(_{Id_A}\) \( \varphi \Rightarrow \) says\(_{Id_A}\) \( \varphi \) (for all \( A \in O_P \) and \( Id_A \subseteq Id'_A \subseteq l(A) \))

A4. \( O_A\varphi \Rightarrow P_A\varphi \) (for all \( A \in O_P \))

A5. says\(_{Id_A}\)(\( P_B \) says\(_{Id_B}\) \( \varphi \)) \Rightarrow (says\(_{Id_B}\) \( \varphi \) \Rightarrow says\(_{Id_A}\) \( \varphi \)) (for all \( \{A, B\} \subseteq O_P, \ Id_A \subseteq l(A), \) and \( Id_B \subseteq l(B) \))

A6. says\(_{Id_A}\)(\( P_A \) says\(_{Id_A}\) \( \varphi \)) \Rightarrow says\(_{Id_A}\) \( \varphi \) (for all \( A \in O_P, \) and \( Id_A \subseteq l(A) \))

R1. From \( \vdash \varphi \Rightarrow \psi \) and \( \vdash \varphi \), infer \( \vdash \psi \)

R2. From \( \vdash \varphi \), infer \( \vdash Q(\varphi) \) (for all modalities \( Q \))

Figure 4.2: Axiomatization of the propositional fragment of \( L \). The set of modalities \( Q \) consists of says\(_{Id_A}\) \( \varphi \) and \( O_A(\varphi) \) (for all \( A \in O \) and \( Id_A \subseteq l(A) \)).

rules R1 and R2. Given a finite set of formulas \( \Delta \), we say that \( \varphi \) is provable from \( \Delta \), denoted \( \Delta \vdash \varphi \), if \( \vdash (\bigwedge \Delta) \Rightarrow \varphi \).

We mention some provable statements that we will use in the example from HIPAA (Section 4.3.2):

**Proposition 4.2.** The following are provable:

1. \( \vdash says_{l(A)}(O_B \) says\(_{l(B)}\) \( \varphi \)) \Rightarrow (says\(_{l(B)}\) \( \varphi \) \Rightarrow says\(_{l(A)}\) \( \varphi \))

2. \( \vdash says_{l(A)}(O_A \) says\(_{l(A)}\) \( \varphi \)) \Rightarrow says\(_{l(A)}\) \( \varphi \)

3. \( \vdash says_{l(A)}(P_B \) says\(_{l(B)}\) \( \bot \)) \Rightarrow (says\(_{l(B)}\) \( \varphi \) \Rightarrow says\(_{l(A)}\) \( \varphi \))
The proofs are easy and we leave the details to the reader. Items 1 and 2 show that versions of axioms \( \text{A5} \) and \( \text{A6} \) hold for obligation. Item 3 gives us *speaking for*, i.e., \( B \) speaks for \( A \), as we discussed in Section 4.2.2.

The rest of the section is organized as follows. We begin by discussing the various axioms in the context of related work. We then analyse some aspects of the example from HIPAA (introduced in Section 4.3.1, and the various subtleties involved in reasoning about rights. We present a complete Kripke semantics for the axioms, and use it to show that provability is decidable.

**Discussion of Axioms**

We now discuss the axioms. The axioms \( \text{A1} \) and \( \text{A2} \), together with the rules \( \text{R1} \) and \( \text{R2} \), gives us the modal logic \( \text{K} \). The \( \text{K} \) axiomatization was used by Abadi et al. [3] as a basis for all (classical) access control logics. From \( \text{A3} \), it follows that if \( A \) says \( \varphi \) via the laws \( (\text{Id}_A) \), then \( \varphi \) also holds w.r.t. a larger set of laws issued by \( A \) \( (\text{Id}_A') \).

As we discussed in Chapter 2 (Section 2.4), these axioms arise from the predicative definition of *says*, which we are extending to a modal definition.

The \( \text{K} \) axiomatization, together with \( \text{A4} \), gives us the the modal logic \( \text{KD} \). This axiomatization is common to many systems, giving it the name Standard Deontic Logic (SDL) (c.f. [73]). We note that SDL is a very simplistic system of obligation, and several objections can be raised. The most serious objection is that SDL does not cope with contrary-to-duty (CTD) obligations (see, e.g., [57, 98, 123]). A CTD obligation is one that arises when another has been violated. This is useful, for example, in business contracts to describe mitigating actions [56, 60, 62, 88], e.g., “paying a fine”, upon failure to deliver goods. We do not address CTD structures in this work, as they are not as prevalent in privacy regulation as they are in contracts. Governatori and Rotolo [57] propose that CTDs are not a problem with obligations per se, but can be understood as a special kind of exception. We agree entirely with their perspective. However, accommodating these kinds of exceptions involves the
introduction of a preference operator, and we leave this to future work.

As we discussed in Section 4.2.2, A5 is needed to accommodate notions of representation in access control. The self-respecting axiom, A6, is read as “If A permits herself to say ϕ, then A says ϕ”. We discuss the use of A6 in the example from HIPAA.

Example

We consider the example from HIPAA, introduced in Section 4.3.1. In this section, our focus is on the utterances obtained from the laws of the various principals.

Let H stand for (the regulator who wrote) HIPAA. And, let Alice (A) be a patient whose records (r) are maintained by an insurance company run by Bob (B). Let us assume further that A has the right to access her records. The utterance obtained from H’s laws would be:

\[(u1) \text{says}_{l(H)} \, \mathcal{P}_A \text{says}_{l(A)} \, \mathcal{O}_B \text{says}_{l(B)} \, \mathcal{P}_A \text{access}(A, r)\]

The direct reading of (u1) in English is unwieldy, i.e., we get “H says that A is permitted to say that B is required to say that A is permitted to access her records”. A better reading is obtained by eliding all occurrences of say that appear immediately above an obligation or permission, except for the outermost one. Applying this ellipsis to (u1), we get: \text{says}_{l(H)} \, \mathcal{P}_A \ldots \mathcal{O}_B \ldots \mathcal{P}_A \text{access}(A, r), which is read as: “HIPAA says that Alice is permitted to require Bob to permit her to view her records”. We will use such readings henceforth.

The word right does not have a unique translation into logic. Hohfeld [69] pointed out that the word right is used in different senses, and depending on the context, it can entail a permission, claim, or power.\(^4\) The formulation in (u1) corresponds to the power interpretation. As we mentioned in Section 4.1, our descriptions of powers

\(^4\)Hohfeld [69] describes a claim as the correlative of obligation, i.e., when a claim is invad an obligation is violated. For example, a patient has a claim that hospitals notify her of disclosures of her health information. And, the claim is equivalent to an obligation on the hospital to notify her.
follows the suggestion of Lindahl [95, Part II] (see also [74]), in terms of nesting obligations and permissions with an action modality.

How does Alice exercise this right? In our approach, rights are exercised by the introduction of a law. The specific mechanism for introducing such laws is application dependent. For example, if Alice sends an email to Bob requiring him to grant her access, then this may count as Alice exercising her right (see [53, 75]). Alice’s attempt to exercise her right would be result in the following utterance:

\[(u2) \text{ says}_{l(A)} \mathcal{O}_B \text{ says}_{l(B)} \mathcal{P}_A \text{ access}(A, r)\]

In other words, Alice says that Bob is required to permit her to view her records.

How does Bob comply with this right? In our approach, this happens via Bob’s access control policy. Suppose Bob wants to permit a patient to view their records only if HIPAA requires it. Bob’s policy is represented as follows:

\[(u3) \text{ says}_{l(B)} \mathcal{P}_H \text{ says}_{l(H)} \mathcal{O}_B \text{ says}_{l(B)} \mathcal{P}_A \text{ access}(A, r)\]

In other words, Bob permits HIPAA to require him to permit Alice to view her records. Note that Bob has no regard for Alice’s requirement to see her records, but only what HIPAA says.

Let \( \Delta \) consist of the utterances (u1), (u2), and (u3) above. Since Alice wants to view her records, the access control system tries to prove that \( \Delta \vdash \text{ says}_{l(B)} \mathcal{P}_A \text{ access}(A, r) \). The derivation proceeds as follows:

\[(d1) \Delta \vdash \text{ says}_{l(H)} \mathcal{O}_B \text{ says}_{l(B)} \mathcal{P}_A \text{ access}(A, r) \) (from (u1) and (u2) using \textbf{A5}).

\[(d2) \Delta \vdash \text{ says}_{l(B)} \mathcal{O}_B \text{ says}_{l(B)} \mathcal{P}_A \text{ access}(A, r) \) (from (u3) and (d1) using Proposition 4.2 item 1)

\[(d3) \Delta \vdash \text{ says}_{l(B)} \mathcal{P}_A \text{ access}(A, r) \) (from (d2) using Proposition 4.2 item 2)

Step (d1) is understood as HIPAA enforcing Alice’s right, i.e., HIPAA requires Bob to permit Alice to view her records. In step (d2), Bob acknowledges HIPAA’s authority
by requiring himself to permit Alice to view her records. Step (d3) shows the utility of A6, i.e., by forcing Bob to say what he requires himself to say. Due to (d3), Alice is indeed permitted to view her records! In the Section 4.3.5, we will show how blame can be assigned to Bob if he fails to introduce (d3) or something that implies it.

In summary, to reason about a right, we had to use utterances from the enforcer (HIPAA), the person exercising the right (Alice), and the person complying with it (Bob). The precise manner in which Alice’s utterance is obtained is left unspecified. In assessing violations of rights, the issue in question is often whether the right was exercised. For example, Bob may claim that Alice did not request to see her records. We do not believe that this is a problem for logic, but it is a problem in implementing a system that allows principals to exercise their rights. However, we do believe that the logic provides a good intuition for the inferences involved, given the appropriate utterances.

The reasoning involved in this example is outside the scope of prior access control logics [1–3, 33, 50, 51, 93], because obligation is not accommodated. We believe that the reasoning can be accommodated by the counts as frameworks for power [53, 75], but as discussed in Section 4.2.3, some reformulation is needed.

**Semantics, Soundness, and Completeness**

In this section, we provide a Kripke semantics for which the axiomatization is sound and complete. Semantic completeness is used mainly as a tool, for example, to show that a statement is not provable. Identifying a compelling semantics for says is an important open problem in access control logics (see [1]), and we do not address it in this work.\(^5\) We begin by defining models (Kripke structures):

**Definition 4.3** (Models). *Given countable sets of object names \(O\), principal names \(O_P \subseteq O\), \(\Phi_1, \ldots, \Phi_n\) (where \(\Phi_j\) is a set of predicate names of arity \(j\)), identifiers for

\(^5\)We speculate that a good semantics for says has to come from an application other than access control and conformance. In these applications, saying arises via policies, which are expressed using formulas. There does not seem to be a corresponding computational interpretation.
rules $ID$, and $l : O_P \rightarrow 2^{|ID|}$, a model $M(O, O_P, \Phi_1, ..., \Phi_n, ID, l)$, abbreviated as $M$, is the tuple $(S, I_{\Phi_1}, ..., I_{\Phi_n}, \delta_L, \delta_O)$ where:

- $S$ is a set of states
- $I_{\Phi_j} : \Phi_j \times S \rightarrow 2^{O_j}$ is the interpretation of predicates of arity $j$. Given $p \in \Phi_j$, we will say that $p(o_1, ..., o_j)$ is true at state $s$ iff $(o_1, ..., o_j) \in I_{\Phi_j}(p, s)$.
- $\delta_L : S \times 2^{|ID|} \rightarrow 2^S$. $\delta_L(s, Id)$ corresponds to a description of $s$ according to the laws labeled with identifiers in $Id$ (taken conjunctively).
- $\delta_O : S \times O_P \rightarrow 2^S$. $\delta_O(s, A)$ corresponds to an idealization of $s$, for which the principal $A$ is held responsible.

For the axioms $A_3$-$A_6$ we need the following constraints $C_3$-$C_6$ (resply). For all $s \in S$:

$C_3$ $\delta_L(s, Id_A) \supseteq \delta_L(s, Id_A')$ for all $A \in O_P$ and $Id_A \subseteq Id_A' \subseteq l(A)$

$C_4$ $\delta_O(s, A) \neq \emptyset$ for all $A \in O_P$

$C_5$ For all \( \{A, B\} \subseteq O_P \), $Id_A \subseteq l(A)$, $Id_B \subseteq l(B)$, and $s' \in \delta_L(s, Id_A)$:

1. $s' \in \delta_L(s, Id_B)$, or

2. There exists $s_1 \in \delta_L(s, Id_A)$ such that for all $s_2 \in \delta_O(s_1, B)$, $s' \in \delta_L(s_2, Id_B)$

$C_6$ For all $A \in O_P$, $Id_A \subseteq l(A)$, and $s' \in \delta_L(s, Id_A)$:

There exists $s_1 \in \delta_L(s, Id_A)$ such that for all $s_2 \in \delta_O(s_1, A)$, $s' \in \delta_L(s_2, Id_A)$

$C_5$ and $C_6$ can be understood in the context of soundness (Lemma B.1). Given the object names $O$, $O_P \subseteq O$, predicate names $(\Phi_1, ..., \Phi_n)$, identifiers $ID$, and the function $l$, the space of models is denoted by $M(O, O_P, \Phi_1, ..., \Phi_n, ID, l)$, abbreviated as $\mathcal{M}$. We can now define satisfaction and validity, and we restrict attention to the propositional fragment of $L$: 
Definition 4.4 (Semantics). Given a model $M = (S, I_{\Phi_1}, ..., I_{\Phi_n}, \delta_L, \delta_O)$, $s \in S$ and a propositional $\varphi \in L$, the relation $(M, s) \models \varphi$ is defined inductively as follows:

- $(M, s) \models p(o_1, ..., o_j)$ iff $(o_1, ..., o_j) \in I_{\Phi_j}(p, s)$.
- The semantics of conjunction and negation is defined in the usual way.
- $(M, s) \models \text{says}_{Id} \varphi$ iff $(M, s') \models \varphi$, for all $s' \in \delta_L(s, Id)$.
- $(M, s) \models \text{O}_A \varphi$ iff $(M, s') \models \varphi$, for all $s' \in \delta_O(s', A)$.

We can now define validity:

- $\varphi$ is valid in a model $M$ ($M \models \varphi$) iff for all $s \in S$, $(M, s) \models \varphi$
- $\varphi$ is valid ($\models \varphi$) iff for all $M \in \mathcal{M}$, $M \models \varphi$

We state the main results that can be proved using the semantics. Proofs can be found in Appendix B.

Theorem 4.1 (Soundness and Completeness). Given a propositional $\varphi \in L$:

$\vdash \varphi$ iff $\models \varphi$

Corollary 4.1 (Compactness). An infinite set of formulas is satisfiable iff every finite subset is satisfiable.

Theorem 4.2 (Decidability). Given a propositional $\varphi \in L$, the problem of checking whether $\vdash \varphi$ is decidable.

Theorem 4.3 (Complexity). Given a propositional $\varphi \in L$, deciding whether $\varphi$ is satisfiable is NEXPTIME-complete.

NEXPTIME is the class of problems that can be decided in non-deterministic exponential time. In Section 4.3.6, we will identify a fragment of the logic (motivated by access control examples), for which provability can be decided in polynomial time.

We now turn to the formalization of policies.
4.3.3 The Saying Component - Policies

In this section, we modify ReFl to accommodate the new axioms. The formalization is essentially identical to that in Chapter 2. We briefly review the definitions and then discuss the example from HIPAA introduced in Section 4.3.2. The syntax of regulation is defined as follows:

**Definition 4.5 (Syntax of Regulation).** Given countable sets of identifiers \( ID \), principal names \( OP \), and a function \( l : OP \rightarrow 2^{ID} \), a law is a statement of the form 

\[
(id) \varphi \mapsto \psi, \quad \text{where} \quad \varphi \in L_\varphi, \ \psi \in L_\psi, \ \text{and there exists} \ A \in OP \ \text{such that} \ id \in l(A).
\]

The set of all possible laws is denoted by \( \text{Laws}(OP, l, L) \), abbreviated \( \text{Laws} \).

A body of regulation \( \text{Reg} \subseteq \text{Laws} \) is a finite set such that for all \( id \in ID \), there exists at most one pair \( (\varphi, \psi) \in L_\varphi \times L_\psi \) such that 

\[
(id) \varphi \mapsto \psi \in \text{Reg}
\]

\( (id) \varphi \mapsto \psi \) is read as: “If \( \varphi \) is true, then \( A \) says \( \psi \) via the law \( (id) \)”, where \( id \in l(A) \). To evaluate laws, we need a way to evaluate preconditions (\( \varphi \in L_\varphi \)). There are two kinds of atoms in \( L_\varphi \) – predicates and formulas of the form \( \text{says}_{Id_\varphi} \varphi \). The predicates are evaluated against a state, and formulas of the form \( \text{says}_{Id_\varphi} \varphi \) are evaluated provability (as defined in Section 4.3.2) from a set of utterances. We begin by defining states:

**Definition 4.6 (States and Assignments).** Given countable sets \( O \) of object names, principal names \( OP \subseteq O \), and predicate names \( \Phi_1, ... , \Phi_n \), a state \( s(O, OP, \Phi_1, ..., \Phi_n) \), abbreviated \( s \), is the tuple \( (I_{\Phi_1}, ... , I_{\Phi_n}) \) where \( I_{\Phi_j} : \Phi_j \rightarrow 2^{O_j} \) is the interpretation of predicates of arity \( j \). Given \( p \in \Phi_j \), we will say that \( p(o_1, ..., o_j) \) is true at state iff 

\[
(o_1, ..., o_j) \in I_{\Phi_j}(p).
\]

The set of all states is denoted by \( S \).

Given a set of variables \( X \), and principal variables \( XP \), an assignment is a function \( v : X \rightarrow O \), such that for all \( x \in XP \), we have \( v(x) \in OP \). The set of all assignments is denoted by \( V(X, XP, O, OP) \), abbreviated \( V \).

A state \( s \in S \) is a description of operations, and gives us information, for example, about the accesses to records that actually happened. The definition of utterances
relies on propositionalizing formulas:

**Definition 4.7 (Propositionalization).** Given $\phi \in L$ and an assignment $v \in V$, the propositionalization of $\phi$ w.r.t. $v$, denoted $v(\phi)$, is defined inductively as follows:

- $v(p(y_1, \ldots, y_n)) = p(o_1, \ldots, o_n)$, where $o_i = v(y_i)$ if $y_i \in X$ and $o_i = y_i$ otherwise ($y_i \in O$).
- $v(\varphi \land \psi) = v(\varphi) \land v(\psi)$, and $v(\neg \varphi) = \neg v(\varphi)$
- $v(O y \varphi) = O_A(v(\varphi))$, where $A = v(y)$ if $y \in X$ and $A = y$ otherwise.
- $v(says_{Id_y} \varphi) = says_{Id_A}(v(\varphi))$, where $Id_A = l(v(y))$ if $y \in X$ and $Id_A = Id_y$ otherwise.

We can now define utterances:

**Definition 4.8 (Utterances).** Given a body of regulation $\text{Reg}$, and an assignment $v \in V$, an utterance is a statement $v(says_{\text{id}} \psi)$ such that $\text{id} \in \text{ID}$ and $(\text{id}) \varphi \mapsto \psi \in \text{Reg}$. The set of all utterances is denoted by $U(\text{Reg}, V)$.

Next, we define the function $\eta$ which assigns truth values from $\mathcal{B}^3 = \{\top, ?, \bot\}$ to preconditions:

**Definition 4.9 (Evaluating Preconditions).** Given a body of regulation $\text{Reg}$ and a pair utterance sets $(U, U')$ such that $U \subseteq U' \subseteq U(\text{Reg}, V)$, the function $\eta(U, U') : L_\varphi \times S \times V \to \mathcal{B}^3$ is defined as follows:

Predicates are evaluated to true or false. Conjunction and negation are handled using the Kleene semantics.

$$
\eta(U, U')(\text{says}_{Id_y} \psi, s, v) = \begin{cases} 
\top & \text{if } U \vdash v(\text{says}_{Id_y} \psi) \\
\bot & \text{if } U' \not\vdash v(\text{says}_{Id_y} \psi) \\
? & \text{otherwise}
\end{cases}
$$

102
One observation about Definition 4.9 is that we are using the notion of syntactic provability (e.g., $U \vdash v(says_{id_{\psi}})$) in a place where semantic entailment is more appropriate (e.g., $U \models v(says_{id_{\psi}})$). From a technical perspective, the two notions are equivalent due to soundness and completeness (Theorem 4.1). However, from an intuitive standpoint, we wish to illustrate that we determine whether or not something has been said by looking at other statements. In other words, it is theorem proving questions that are of central interest. While semantics is useful as a tool to establish non-provability and decidability, assigning an intuitive semantics to saying is an open problem in access control.

As in Chapter 2, we are interested in utterances that are sound and complete w.r.t. a state:

**Definition 4.10** (Sound and Complete Utterances). Given a regulation $Reg$ and a state $s \in S$, the utterance pair $(U, U')$ is sound w.r.t. $s$ iff for all $\phi \in U(Reg, V)$:

**US1** If $\phi \in U$, then there exists $(id) \varphi \mapsto \psi \in Reg$ and $v \in V$ such that $v(says_{id} \psi) = \phi$ and $tv_{(U, U')}(\varphi, s, v) = \top$.

**US2** If $\phi \notin U'$, then for all $(id) \varphi \mapsto \psi \in Reg$ and $v \in V$ such that $v(says_{id} \psi) = \phi$, we have $tv_{(U, U')}(\varphi, s, v) = \bot$.

Similarly, $(U, U')$ is said to be complete w.r.t. $s$ iff for all $\phi \in U(Reg, V)$:

**UC1** If there exists $(id).x : \varphi \mapsto \psi \in Reg$ and $v \in V$ such that $v(says_{id} \psi) = \phi$ and $\eta_{(U, U')}(\varphi, s, v) = \top$, then $\phi \in U$.

**UC2** If for all $(id).x : \varphi \mapsto \psi \in Reg$ and $v \in V$ such that $v(says_{id} \psi) = \phi$ and $\eta_{(u, u')}(\varphi, R, i, v) = \bot$, then $\phi \notin U'$.

As we showed in Chapter 2, the space of sound utterance pairs has a least and maximal fixed points. The fixed points are sound and complete pairs. A state $s$ together with a sound utterance pair forms the basis for all decision problems. We define a notion of validity at a state, which we will use to formalize access control and conformance decisions (in Sections 4.3.4 and 4.3.5 resply.):
Definition 4.11 (Validity at a State). Given a state $s$, a body of regulation $\text{Reg}$, a consistent utterance pair $(U, U') \in C_{U(\text{Reg}, V)}^s$ and a propositional $\varphi \in L_\varphi$, we say that $\varphi$ is valid at $s$ w.r.t. $\text{Reg}$ and $(U, U')$, denoted $(s, \text{Reg}) \models (U, U') \varphi$, iff $\text{tv}_{(U, U')}(\varphi; s, v) = \top$ for all $v \in V$.

The choice of which utterance pair to use depends on the application. If there is a unique (least) fixed point, then it is the appropriate choice. However, matters are not so clear when there are multiple fixed points. We conclude this section with a discussion of the example from HIPAA to build intuition about the definitions of access control and conformance.

Example 1: Consider the statements from HIPAA in Section 4.3.1. Let $H \in \mathcal{OP}$ stand for the regulatory authority behind HIPAA, and $l(H) = \{1, 1a, 1b\}$. As we discussed in Section 4.3.2, the phrase *has the right* is analysed as a power. We use the following abbreviation:

$$\text{hasRight}(x, z, \varphi) = \mathcal{P}_x \text{say}_{l(x)} \mathcal{O}_x \text{say}_{l(z)} \mathcal{P}_x \varphi$$

The HIPAA rule is formalized as follows:

- (23) $\text{pat}(x) \land \text{rec}(y, x, z) \land \neg \text{say}_{\{23a, 23b\}} e(y) \mapsto \text{hasRight}(x, z, \text{access}(x, y))$
- (23a) $\text{psyNotes}(y_1) \mapsto e(y_1)$
- (23b) $\text{compForLegal}(y_2) \mapsto e(y_2)$

The law (23) is read as follows: “If $x$ is a patient ($\text{pat}(x)$), and $y$ is a record of $x$ maintained by $z$ ($\text{rec}(y, x, z)$), and HIPAA does not say that there is an exception applying to $y$ ($\neg \text{say}_{\{23a, 23b\}} e(y)$), then HIPAA says that $x$ has the right to access her records via the law (23)”.

The law (23a) is read as follows: “If $y_1$ is a record of psychotherapy notes ($\text{psyNotes}(y_1)$), then HIPAA says that an exception applies to $y_1$ via the law (23a)”.

And, the law (23b) is read as follows: “If $y_2$ is a record compiled for legal proceedings ($\text{compForLegal}(y_2)$), then HIPAA says that an exception applies to $y_2$ via the law (23b)”.

104
We set aside the important problem of defining these predicates further, i.e., the definitional aspects of the law [73]. For example, HIPPA provides rules describing who counts as a patient, and the interpretation of \( \text{pat}(x) \) is dependent on these rules. In addition, the predicate \( e(y) \) could be interpreted as a permission to the maintainer of \( y \) not to grant access.\(^6\) Such extensions are easily accommodated.

Suppose Alice (\( A \)) wants to view her records (\( r \)) which are maintained by Bob (\( B \)). Alice introduces the following rule:

\[
(24) \quad \text{Bob must show me my records.}
\]

As discussed in Section 4.3.2, the manner in which this rule arises is left unspecified. For example, Alice may send an e-mail to Bob, requiring to see her records. Alice’s law is formalized as follows. Let \( A \in O_P \) stand for Alice, and \( B \in O_P \) stand for Bob. In addition, \( l(A) = \{24\} \). Law (24) is formally expressed as:

\[
(24) \quad \top \mapsto O_B \text{ says}_{l(B)} P_B \text{ access}(A,r)
\]

Bob complies with this request via his access control policy. Suppose Bob’s policy consists of the following rule:

\[
(25) \quad \text{HIPAA is permitted to require me to permit a patient to access her records.}
\]

Let \( l(B) = \{25\} \). Law (25) is formally expressed as:

\[
(25) \quad \text{pat}(x_3) \land \text{rec}(y_3, x_3, B) \mapsto P_H \text{ says}_{l(H)} O_B \text{ says}_{l(B)} P_{x_3} \text{ access}(x_3, y_3)
\]

Table 4.1 shows a state together with the fixed point utterances obtained from \( l(H), l(A), \) and \( l(B) \). Here, \( r \) is a record about Alice maintained by Bob (\( \text{rec}(r, A, B) \)), which has been compiled for legal proceedings (\( \text{compForLegal}(r) \)). The precondition of HIPAA’s law (23b) is true, and we obtain the utterance \( \text{says}_{(23b)} e(r) \). As a result, the precondition of (23) is false, and no right is granted to Alice. The preconditions of Alice’s and Bob’s laws ((24) and (25) resp.) are true, and the corresponding utterances are obtained.

\(^6\)Our understanding of the HIPAA rule is that (23a) and (23b) are only meant to cancel the right provided by (23), and do not entail any explicit permission to the maintainer of the records.
Table 4.1: A state and fixed point utterances for the HIPAA example.

Let us consider the questions of access control and conformance informally, given the state and fixed point in Table 4.1. Is Alice permitted to access her record? No, because HIPAA does not require Bob to permit her to access it. Is Bob conformant? On one hand, HIPAA doesn’t require anything of Bob, so yes. On the other hand, Alice says that Bob is required to permit her to access her records, and he does not comply with this request. Thus, conformance is better seen as a relation between two principals w.r.t. a set of laws. In Section 4.3.5, we will say that $B$ conforms to $A$ w.r.t. the laws $l(A)$ iff $B$ satisfies the obligations imposed by those laws.

### 4.3.4 Non-interference in Access Control

An access control decision is made when a principal $A$ requests the performance of action $p$ which is controlled by $B$. Given a state $s$, regulation $Reg$ and fixed point $(U, U')$ resulting from the evaluation of policy, the decision problem is whether $(s, Reg) \models_{(U, U')} says_{l(B)} P_A(p)$, i.e., does $B$ say that $A$ is permitted to perform $p$.

A problem with this definition is that the policies in access control are usually distributed. It is unreasonable to expect $(U, U')$ to reside on a single system. Given that we wish to evaluate $says_{l(B)} P_A(p)$, the question is whether a smaller set of utterances suffice to answer this question. In other words, the evaluation should be carried out locally by $B$ or a designated evaluator for $B$, as in [15].

Non-interference properties are used to obtain such results, and to demonstrate that the logic protects the rights of each principal [2, 51]. In our case, the access control decision is of the form $(s, Reg) \models_{(U, U')} says_{Id_B} \psi$, and this holds iff $U \vdash$
The goal is to identify a subset of utterances \((U^* \subseteq U)\), such that \(U \vdash \text{says}_{Id_B} \psi\) iff \(U^* \vdash \text{says}_{Id_B} \psi\).

Let us consider an example to build some intuition. Suppose we have four principals \(A, B, C,\) and \(D\), with \(l(A) = \{id1\}, l(B) = \{id2\}, l(C) = \{id3\},\) and \(l(D) = \{id4\}.\) Suppose \(C\) is a patient, and \(A\) and \(B\) maintain records about \(C.\) \(A\) only permits \(C\) to access her records, while \(B\) permits \(C\) to permit her mother \((D)\) to access her records. Let \(U\) consist of the following utterances:

\[ (u1) \text{ says}_{l(A)} P_C \text{access}(C, r) \]

\[ (u2) \text{ says}_{l(B)} P_C \text{says}_{l(C)} P_D \text{access}(D, r_1) \]

\[ (u3) \text{ says}_{l(C)} P_D \text{access}(D, r_1) \]

Now, suppose \(D\) wants to access \(C\)'s records that are maintained by \(A.\) It is easy to see that \(U \not\vdash \text{says}_{l(A)} P_D \text{access}(D, r).\) But, do we need all of the utterances is \(U\) to make this determination? Intuitively, \(no,\) because the only utterance from \(A\) is \((u1)\) and there is no representation conveyed via \((u1).\) So, \((u1)\) alone should suffice to make this determination. In this case, we say that \((u2)\) and \((u3)\) do not interfere with the access control decision.

Next, suppose \(D\) wants to access \(C\)'s records that are maintained by \(B.\) It follows that \(U \vdash \text{says}_{l(B)} P_D \text{access}(D, r_1),\) and so \(D\) is indeed granted access. Here, \((u2)\) is certainly relevant, and since it gives the power of representation to \(C,\) \((u3)\) is also relevant. However, no mention of \(A\) is made by \((u2)\) or \((u3),\) and so, \((u1)\) does not interfere with the access control decision.

We begin by defining the subset of utterances that are relevant to an access control decision:

**Definition 4.12 (Reachable Utterances).** *Given a set of utterances \(U\) and a formula \(\text{says}_{Id_B} \psi, U^*_{Id_B}\) is the smallest set such that:*

- If \(id_B \in Id_B\) and \(\text{says}_{(id_B)} \varphi \in U, \text{says}_{(id_B)} \varphi \in U^*_{Id_B} \)
• If \( \text{say}_{\text{Id}_B} \varphi \in U^*_\text{Id}_B \) and \( \text{say}_{\text{Id}_A} \psi' \) is a subformula of \( \varphi \), then \( U^*_\text{Id}_A \subseteq U^*_\text{Id}_B \).

If we think of formulas \( \text{say}_{\text{Id}_B} \psi \) as pointing to utterances in \( U \) (labeled \( \text{Id}_B \)), then \( U^*_\text{Id}_B \) is the set of utterances that are pointed to directly (the first clause), or pointed to by subformulas of utterances that are pointed to (the second clause). In these terms, the computation of the set \( U^*_\text{Id}_B \) corresponds to a reachability computation on a graph, and hence, we call it the set of reachable utterances. We believe that it is reasonable to restrict to the reachable utterances, because given the question \( \text{say}_{\ell(B)} \psi \), \( U^*_\ell(B) \) is determined by \( B \) and the principals to whom she grants the power of representation. We can now show the following:

**Theorem 4.4 (Non-interference).** Given a set of utterances \( U \), for all \( \text{say}_{\text{Id}_B} \psi \in L \), we have \( U \vdash \text{say}_{\text{Id}_B} \psi \) iff \( U^*_\text{Id}_B \vdash \text{say}_{\text{Id}_B} \psi \).

The proof is discussed in Appendix B.5. We note that the distinction between the inference component and the saying component allows us to restrict attention to inferences of the form \( U \vdash \text{say}_{\text{Id}_B} \varphi \), where \( U \) only has formulas of the form \( \text{say}_{\text{Id}_A} \psi \). If the set \( U \) could contain arbitrary formulas, non-interference would have a more complex characterization, as in [51]. For example, if we allowed formulas of the form \( \neg \text{say}_{\text{Id}_A} \psi \) in \( U \), then any principal can render \( U \) inconsistent.

The non-interference theorem also tells us that the axioms provide an exact generalization of the predicative definition of \( \text{say} \) in Chapter 2 (see Section 2.4). If there are no nested occurrences of \( \text{say} \) in \( U \), then \( U \vdash \text{say}_{\text{Id}_B} \psi \) iff \( U^*_\text{Id}_B \vdash \text{say}_{\text{Id}_B} \psi \).

### 4.3.5 Conformance

We now turn to the definition of conformance. While the definition of conformance has some variation between the various formalisms [4, 44, 54, 56, 60, 62, 88], all of them require a principal to satisfy the obligations that are imposed on her. In the context of contracts, several works [56, 60, 62, 88] accommodate reasoning about
mitigating actions such as “paying a fine” if an obligation is not satisfied. The analysis of such mitigating actions is left to future work.

We define conformance as a relation between a principal and another principal w.r.t. a set of laws:

**Definition 4.13** (Conformance). Given a state $s$ with a set of objects $O$, a body of regulation $\text{Reg}$, and $\{A, B\} \subseteq O_P$, we say that $A$ conforms to $B$ w.r.t. the laws $Id_B \subseteq l(B)$ and a fixed point $(U, U')$ with $U = U'$ iff for all propositional $\varphi \in L_{\varphi_A}$:

If $(s, \text{Reg}) \models_{(U, U')} \text{says}_{Id_B} O_A \varphi$, then $(s, \text{Reg}) \models_{(U, U')} \varphi$

In other words, conformance is the satisfaction of all obligations. The syntactic restrictions in Definition 4.2 justify the restriction to $\varphi \in L_{\varphi_A}$, as these are the only formulas that can appear within the scope of $O_A$. The restriction to fixed points $(U, U')$, where $U = U'$, ensures that all formulas are either true or false. Definition 4.13 is not appropriate when $U \neq U'$, since classically provable formulas, e.g. $\varphi \lor \neg \varphi$, may be ungrounded. In such cases, the principal would be found (trivially) non-conformant. One way to accommodate these cases is to modify Definition 4.13 so that if $(s, \text{Reg}) \models_{(U, U')} \text{says}_{Id_B} O_A \varphi$, we require only that $\varphi$ be true or ungrounded. With this modification, our proof of decidability over to the case where $U \neq U'$.

Let us apply Definition 4.13 our example from HIPAA in Table 4.1 (Section 4.3.3). As we discussed, we are interested in the conformance of Bob ($B$). Bob does not conform to Alice ($A$) w.r.t. the laws $l(A) = \{24\}$, because:

$$(s, \text{Reg}) \models_{(U, U')} \text{says}_{l(A)} O_B \text{says}_{l(B)} P_{\text{access}}(A, r) \text{ and } (s, \text{Reg}) \not\models_{(U, U')} \text{says}_{l(B)} P_{\text{access}}(A, r)$$

However, it can be shown that Bob conforms to HIPAA ($H$), w.r.t. the laws $l(H) = \{23, 23a, 23b\}$. Additional examples are discussed in Section 4.4.2.

We can show that conformance checking is decidable:

**Theorem 4.5** (Decidability of Conformance). Given a state $S$, a body of regulation $\text{Reg}$, a fixed point $(U, U')$ where $U = U'$ and $|U|$ is finite, principals $\{A, B\} \subseteq O$, and
identifiers \( I_B \subseteq l(B) \), there is a procedure to decide whether \( A \) conforms to \( B \) w.r.t. the laws \( I_B \).

See Appendix B.6 for the proof. Note that given a state \( S \) and a fixed point \((U, U')\), there are potentially infinitely many formulas \( \varphi \in L_{\varphi_A} \) such that \( S \models_{(U, U')} \text{says}_{I_{\text{db}}} O_A \varphi \). For example, if there is some \( \varphi \) such that \( S \models_{(U, U')} \text{says}_{I_{\text{db}}} O_A \varphi \), then for all \( \varphi' \in L_{\varphi_A} \), we have \((s, \text{Reg}) \models_{(U, U')} \text{says}_{I_{\text{db}}} O_A (\varphi \lor \varphi')\). We prove that it suffices to restrict attention to a single formula, which may be understood as a prime implicant of all the obligations imposed on \( A \) via the laws \( I_B \).

### 4.3.6 A Polytime Fragment

Logic programming approaches to access control and conformance enjoy some computational benefits in comparison to our approach (at the cost of expressive power). The main benefit comes from the restriction of heads/postconditions of rules to atomic predicates. In this section, we identify a fragment of the logic that is decidable in polynomial time. We begin by defining the syntax of chain formulas:

**Definition 4.14 (Chain formulas).** Given a countable set \( \Phi \) (of proposition names), countable sets of principal names \( O_P \), a finite set of identifiers \( I_D \), and a function \( l : O \rightarrow 2^{I_D} \), the language \( L(\Phi, O, O_P, l, I_D) \), abbreviated as \( L \), is defined as follows:

\[
\begin{align*}
\varphi_A &::= \bot \mid p \mid \text{says}_{l(A)} \psi_B \ (\forall B \in O_P) \\
\varphi &::= \varphi_A \ (\forall A \in O_P) \\
\psi_A &::= \bot \mid p \mid O_A \varphi_A \mid P_A \varphi_A \\
\psi &::= \psi_A \ (\forall A \in O_P)
\end{align*}
\]

where \( p \in \Phi \). The set of formulas generated by each BNF rule are referred to as \( L_{\varphi_A} \), \( L_\varphi \), \( L_{\psi_A} \) and \( L_\psi \) respectively, and \( L = L_\varphi \cup L_\psi \).

Given a set of formulas \( \Delta \subseteq L \) and \( \psi \in L \), our goal is to determine if \( \Delta \vdash \psi \)
in polynomial time. Let us consider an example, where $\Delta$ contains the following formulas:

- $\text{says}_{l(A_4)} P_{A_4} \text{ says}_{l(A_3)} P_{A_2} \text{ says}_{l(A_2)} p$
- $\text{says}_{l(A_3)} P_{A_1} \text{ says}_{l(A_1)} \bot$
- $\text{says}_{l(A_1)} p$

It follows that $\Delta \models \text{says}_{l(A_4)} p$, since $A_3$ permits $A_1$ to speak for her. However, we will show that $\Delta \not\models \text{says}_{l(A_4)} p$, since $A_3$ has not established the appropriate delegation chain via $A_2$. We briefly discuss the restrictions imposed by chain formulas, and then turn to the decision procedure.

**Discussion of Restrictions:** Chain formulas are a generalization of the constructions used in the language SECPAL [16]. In particular, we accommodate obligation and speaking for. Many of the examples in the access control literature can be expressed in this fragment. From the conformance perspective, however, we lose the capability to express prohibitions. Consider the following statement:

(26) A bloodbank must not ship a donation, if it tests positive for HIV.

This can be expressed as law, using the formalism in Section 4.3.3. However, the utterances that arise will be of the form: $\text{says}_{(26)} O_B \neg\text{ship}(B, d)$, i.e., the regulator says (via law (26)) that the bloodbank $B$ must not ship the donation $d$. The presence of negation over the atomic proposition ship keeps it outside the chain fragment. We conjecture that negation can be accommodated with polytime decidability, but leave an investigation to future work. We note that even the presence of falsity ($\bot$) poses challenges. When $\Delta \not\models \psi$, we do not know if there is a model of polynomial size to demonstrate that it is not provable. However, the existence of a model (of worst-case exponential size) can be shown, and we can avoid explicitly constructing it.

We now discuss the other restrictions imposed by chain formulas (Definition 4.14). says is restricted to formulas of the form $\text{says}_{l(A)} \psi$ and formulas $\text{says}_{Id_A} \psi$ for $Id_A \subset$
$I(A)$ are not allowed. This is done only to simplify the notation in proofs and all the
techniques that we discuss are adapted easily to accommodate such formulas.

Conjunctions are not allowed within a chain. However, using the following equivalences, we can allow conjunctions under saying:

\[
\vdash \text{says}_{I(A)}(\varphi \land \psi) \iff (\text{says}_{I(A)} \varphi \land \text{says}_{I(A)} \psi)
\]

This equivalence lets us turn formulas with conjunctions into chains, and hence, all the techniques that we discuss are easily adapted to accommodate this case. Conjunctions can also be used within obligations, due to a similar property. However, for permissions, we have:

\[
\vdash P_A(\varphi \land \psi) \Rightarrow (P_A \varphi \land P_A \psi)
\]

But, the converse is not necessarily true. We do not know if conjunctions under permissions can be accommodated with polytime decidability.

The next restriction is the exclusion of negation, and in particular, negation does not appear over atomic propositions. In a modal logic without the axioms $A5$ and $A6$, negations can be easily accommodated in chains (with polytime decidability) due to the tree-model property [146]. However, with $A5$ and $A6$, the models are trees with edges between siblings. The presence of these sibling edges make it difficult to accommodate negation. In fact, the presence of $\bot$ poses challenges as well. When $\Delta \not\vdash \psi$, we do not know if there is a model of polynomial size to demonstrate that it is not provable. However, the existence of a model (of worst-case exponential size) can be shown, and we can avoid explicitly constructing it.

The final restriction is the strict alternation between saying and permission. Formulas of the form $\text{says}_{I(A)} \text{says}_{I(B)} \psi$ are excluded. The algorithm presented below can be extended (with some difficulty) to accommodate this case.

In applications where there are a mix of chain and non-chain formulas, we can use the non-interference (Theorem 4.4) to decide if the polytime procedure can be used for a particular decision, i.e., when the non-chain formulas do not interfere.
Polytime Decision Procedure: We now turn to the design of the algorithm to decide whether $\Delta \vdash \psi$, starting the a notion of structural implication:

**Definition 4.15** (Structural Implication). The relation $\trianglerighteq \subseteq L \times L$ (written infix as $\varphi \trianglerighteq \psi$ and read $\varphi$ structurally implies $\psi$) as the smallest set such that:

1. $\bot \trianglerighteq \varphi$
2. $\varphi \trianglerighteq \varphi$
3. $O_A \varphi \trianglerighteq X_A \psi$ if $\varphi \triangleright \psi$, where $X = O$ or $X = P$
4. $P_A \varphi \trianglerighteq P_A \psi$ if $\varphi \triangleright \psi$
5. $X_A \bot \trianglerighteq \varphi$, where $X = O$ or $X = P$
6. $\text{says}_{i(A)} \varphi \trianglerighteq \text{says}_{i(A)} \psi$ if $\varphi \triangleright \psi$, for all $A \in O$
7. $\text{says}_{i(A)} X_A \text{says}_{i(A)} \varphi \triangleright \psi$ if $\text{says}_{i(A)} \varphi \triangleright \psi$, for all $A \in O$, where $X = O$ or $X = P$

Given a set of formulas $\Delta$ and a formula $\psi$, $\Delta \triangleright \psi$ iff there exists $\varphi \in \Delta$ such that $\varphi \triangleright \psi$.

We mention some properties of structural implication:

**Proposition 4.3.** The following hold:

- If $\varphi \triangleright \psi$, then $\vdash \varphi \Rightarrow \psi$
- If $\Delta \triangleright \psi$, then $\Delta \vdash \psi$
- If $\varphi \triangleright \psi$, $\varphi \triangleright \phi$, then $\varphi \triangleright \phi$

The proof follows easily by induction on the clauses of Definition 4.15. We now define closed sets which form the basis for the polynomial time decision procedure:
Definition 4.16 (Closed sets). A set of formulas $\Delta$ is said to be closed iff for all \{\text{says}_{l(A)} \varphi, \text{says}_{l(B)} \psi\} \subseteq \Delta$ and $A \neq B$, we have:

- If $O_B \text{ says}_{l(B)} \psi \triangleright \varphi$, $\text{says}_{l(A)} \varphi \in \Delta$.
- If $\varphi \triangleright P_B \text{ says}_{l(B)} \psi$, $\text{says}_{l(A)} \psi \in \Delta$.

Given a set of formulas $\Delta$, the closure of $\Delta$, denoted by $\Delta^*$, is the smallest set such that $\Delta \subseteq \Delta^*$ and $\Delta^*$ is closed.

We will prove the following:

Theorem 4.6. Given a finite set of formulas $\Delta$ and a formula $\psi$:

1. $\Delta^* \triangleright \psi$ iff $\Delta \vdash \psi$.
2. $\Delta^*$ can be computed in polynomial time.
3. $\Delta^* \triangleright \psi$ can be decided in polynomial time.

Proof. Item 1: The soundness, i.e., if $\Delta^* \triangleright \psi$, then $\Delta \vdash \psi$, follows easily from the proof of item 2 below. The completeness, i.e., if $\Delta \vdash \psi$, then $\Delta^* \triangleright \psi$, is, as usual, more difficult, and the proof is given in Appendix B.7.

Item 2: We first consider the complexity of computing $\varphi \triangleright \psi$. If we turn Definition 4.15 directly into a (recursive) procedure, we get a worst-case exponential bound (due to clause (7)). This needs to be handled by comparing prefixes of $\varphi$ and $\psi$. Given a formula $\text{says}_{l(A)} \phi$ and $A \in O$, the $A$ prefix of $\phi$, denoted $w^A_\phi$, is defined as follows:

- $w^A_\phi = \epsilon$ if $\phi$ is $\bot$, atomic, of the form $X_B \phi'$, where $B \neq A$, or $X_A \psi$, where $\psi$ is atomic or $\bot$. The $A$ suffix of $\phi$ is $\phi - w^A_\phi = \phi$.
- Otherwise, $w^A_\phi = (X^1_A, \ldots, X^n_A)$, where $\phi = X^1_A \text{ says}_{l(A)} \ldots X^n_A \text{ says}_{l(A)} \phi'$ and $w^A_\phi = \epsilon$. In this case, the $A$ suffix of $\phi$ is $\phi - w^A_\phi = \phi'$. 

114
\[ |w^A_\varphi| \text{ denotes the length of the } A \text{ prefix, where } |\epsilon| = 0. \]
\[ w^A_\varphi(i) \text{ denotes the } i^{th} \text{ element of the prefix for } 1 \leq i \leq |w^A_\varphi|. \]

Given \( \text{says}_{i(A)} \varphi \) and \( \text{says}_{i(A)} \psi \), \( w^A_\varphi \) and \( w^A_\psi \) are the \( A \) prefixes. We say that \( w^A_\varphi(j) \) matches \( w^A_\psi(i) \), denoted \( w^A_\varphi(j) \triangleright w^A_\psi(i) \), if \( w^A_\varphi(j) = \varnothing \), or \( w^A_\varphi(j) = w^A_\psi(i) \). The matching relation is extended to suffixes of strings.

\( (w^A_\varphi, j) \triangleright (w^A_\psi, i) \) if \( j \leq |w^A_\varphi| \) and:

- \( i > |w^A_\psi| \), or
- \( i \leq |w^A_\psi|, w^A_\varphi(j) \triangleright w^A_\psi(i) \) and \( (w^A_\varphi, j + 1) \triangleright (w^A_\psi, i + 1) \), or
- \( i \leq |w^A_\psi|, w^A_\varphi(j) \not\triangleright w^A_\psi(i) \) and \( (w^A_\varphi, j + 1) \triangleright (w^A_\psi, i) \)

And, finally, the matching relation is extended to strings: \( w^A_\varphi \triangleright w^A_\psi \) iff \( (w^A_\varphi, 1) \triangleright (w^A_\psi, 1) \). We can now turn Definition 4.15 into a recursive procedure. Clauses (1)-(5) remain, and clauses (6) and (7) are replaced with the following:

- \( \text{says}_{i(A)} \varphi \triangleright \text{says}_{i(A)} \psi \) if:
  - \( |w^A_\varphi| = 0 \) and \( \varphi \triangleright \psi \), or
  - \( w^A_\varphi \triangleright w^A_\psi \) and \( \varphi - w^A_\varphi \triangleright \psi - w^A_\psi \)

The equivalence to Definition 4.15 is established easily by induction. Note that each application of the third clause of \( (w^A_\varphi, j) \triangleright (w^A_\psi, i) \) corresponds to an application of clause (7) in Definition 4.15. The complexity of this procedure is \( O(n + m) \) where \( n \) is the depth of \( \varphi \) and \( m \) is the depth of \( \psi \).

To compute the closure of \( \Delta \), we initialize \( \Delta^* \) to \( \Delta \) and repeatedly apply the two clauses of Definition 4.16 until convergence. Let \( \mathcal{A} \) be the set such that \( A \in \mathcal{A} \) iff \( \text{says}_{i(A)} \psi \) is a subformula of \( \Delta \). In the worst case, at each iteration, we add just one formula, and achieve convergence at \( |\mathcal{A}| \times s \), where \( s \) is the number of subformulas in \( \Delta \). The complexity of each iteration is \( O(s^2 \times n) \), where \( n \) is the depth of \( \Delta \). And, as a result, the complexity of computing \( \Delta^* \) is \( O(|\mathcal{A}| \times s^3 \times n) \).

**Item 3:** The size of \( \Delta^* \) is \( O(|\mathcal{A}| \times s) \) and the depth is \( n \). Thus \( \Delta^* \triangleright \psi \) can be decided in \( O(|\mathcal{A}| \times s \times m) \) where \( m \) is the maximum of \( n \) and the depth of \( \psi \). \[ \square \]
4.4 Discussion

In this section, we discuss how various constructs from the literature are expressed in our framework. In Section 4.4.1, we discuss access control examples. Section 4.4.2 discusses conformance in the presence of nested obligations and permissions. Finally, in Section 4.4.3, we connect the logical work to speech act theories.

4.4.1 Access Control

We discuss two access control examples in this section. The first example highlights an important restriction of the policies in Section 4.3.3, i.e., a policy lets us conclude what has been said, but not what actually happens. The second example illustrates how the delegation operator of Li et al. [93] can be defined in our framework.

Example 1: We begin with an example from Garg and Abadi [50]. Consider a file-access scenario with an administrating principal (A), a user (B), a file (file1), and the following policy:

1. If A says that file1 should be deleted, then this must be the case.
2. A trusts B to decide whether file1 should be deleted.

We introduce a new principal F for the file system. The following are the utterances (U) obtained at the fixed point:

1. \( P_A \) says_\( F \) \( O_{F}(delfile1) \)
2. \( P_B \) says_\( A \) \( O_{F}(delfile1) \)
3. \( P_B \) says_\( B \) \( O_{F}(delfile1) \)

The first utterance is read as follows: The file system F says that A is permitted to require it (F) to delete file1. The second utterance is the delegation from A to B, and the third utterance is B’s wish to delete file1. Using A5, we will conclude that \( U \vdash \text{says}_{\text{F}} \) \( O_{F}(delfile1) \). In other words, we conclude that the system requires itself to delete file1.

Our analysis differs in an important way from Garg and Abadi [50]. We do not
conclude that file1 is actually deleted, i.e., $U \not\vdash \text{delfile1}$. In fact, we can show that there is no policy (as defined in Section 4.3.3) that lets us make this conclusion. delfile1 is true at a state where $F$ conforms to $l(F)$, as per Definition 4.13. In some cases, it may be warranted to assume/axiomatize self-conformance, i.e., $(\text{says}_{l(F)} O_F(\varphi)) \Rightarrow \varphi$. However, conflicting self-imposed requirements would make $U$ inconsistent.

**Example 2:** The delegation operator of Li et al. [93] has a compelling definition in our framework. The syntax (in Li et al. [93]) for delegation is “$x$ delegates $(\varphi)^d$ to $y$”, where $d$ is the depth of delegation. We define the schema $\text{ps}(\varphi, x, d)$, where $x$ is used to generate variable names, and $d \in \mathbb{N}$:

$$\text{ps}(\varphi, x, 1) = \mathcal{P}_x \text{ says}_{l(x_1)} \varphi$$

$$\text{ps}(\varphi, x, d) = \mathcal{P}_{x_d} \text{ says}_{l(x_d)}(\varphi \land \text{ps}(\varphi, x, d - 1)), \text{ for } d > 1$$

The statement “$A$ delegates (delfile1)$^2$ to $B$” is interpreted as follows: $A$ says delfile1 if $B$ says it or anyone that $B$ trusts says it. Suppose, in addition, that $B$ delegates (delfile1)$^1$ to $C$, and $C$ says delfile1. We express this with the following rules:

1. $(x_2 = B) \mapsto \text{ps(}\text{delfile1}, x, 2)$
2. $(y_1 = C) \mapsto \text{ps(}\text{delfile1}, y, 1)$
3. $\top \mapsto \text{delfile1}$

We assume that $1 \in l(A)$, $2 \in l(B)$ and $3 \in l(C)$. At the fixed point, we will have $U \vdash \text{says}_{l(A)} \text{delfile1}$, i.e., $A$ says delfile1. Further re-delegations by $C$ (by modifying statement 3) will not be attributed to $A$.

In the logic of Li et al. [93], a representation statement is used to grant permission to speak *without consuming delegation depth*. If $C$ represents $B$ on delfile1, then $C$ should be permitted to at most one re-delegation. Statement 2 is modified as follows:

2. $(y_2 = C) \mapsto \text{ps(}\text{delfile1}, y, 2)$

With this modification, a delegation by $C$ will be attributed to $A$. The reader may have noticed the similarity between statement 1 and the modified version of statement 2. In our approach, delegation is just a special kind of representation. $A$ delegates
(\varphi)^d to B represents A on “delegating (\varphi)^{d-1} to anyone”. If C represents B on “delegating (\varphi)^{d-1} to anyone”, then she represents A as well.

As Li et al. [93] point out, in the presence of representation, delegation depth does not have much meaning. For example, A may not wish to trust C to the same extent as B. There are a few options to address this issue by modifying the representation axiom. One way is to keep track of the delegation depth in the axiom, as in the SECPAL language [16]. Yet another way is to keep track of the principal on behalf of whom a statement in made. We avoid these modifications, to simplify presentation.

4.4.2 Nested Obligations and Permissions

We discuss two examples of conformance in the presence of nested obligations and permissions. The first example illustrates how several fine-grained notions of conformance can be captured, and is intended to supplement the example from HIPAA in Section 4.3. The second example points out an important practical difficulty.

Example 1: Consider the following law:

(27) The owners of parking lots ought to forbid parking near the entrance.

What does it mean to conform to (27)? We analyze this sentence as follows: “The owners of parking lots ought to (introduce laws that) forbid parking near the entrance.”. In other words, (27) is an obligation to introduce a prohibition. If the owner introduces such a law, then the person parking is viewed as non-conformant, but it is the owner that needs to conform to (27). We can represent (27) in logic as follows:

(27) own(x) \land p(y) \rightarrow O_x\text{says}_{l(x)}O_y\neg pk(y,x)

Here own(x) is true iff x is the owner of a parking lot, p(y) is true iff y is a person, and pk(y, x) is true iff y parks near the entrance of the lot owned by x. l(x) refers to the laws that are introduced by x.

Let us assume a state S = (I_{\varphi_1}, ..., I_{\varphi_n}) in which A is the owner of a parking lot, and B parks near the entrances of A’s lot. The true predications are: \{own(A), p(B),
In addition, $A$ is assigned the identifier 28, i.e., $I(A) = \{28\}$. We will now consider two scenarios – (a) $A$ does not introduce any laws, and (b) $A$ introduces a law forbidding parking near the entrance. We are interested in the conformance (Definition 4.13) of the owner $A$ and the driver $B$.

**Scenario 1:** Suppose that $A$ does not introduce any laws. The fixed point utterance pair is:

$$ U = U' = \{\text{says}_{(27)} \circ_A \text{says}_{(28)} \circ_B \neg \text{pk}(B, A)\} $$

In this case, $A$ does not conform to $\{27\}$ because:

- $U \vdash \text{says}_{(27)} \circ_A \text{says}_{(28)} \circ_B \neg \text{pk}(B, A)$, but

- $S \not\models (U, U') \text{ says}_{(28)} \circ_B \neg \text{pk}(B, A)$

However, it can be shown that $B$ conforms to $\{27\}$.

**Scenario 2:** Now suppose that $A$ introduces the law:

$$(28) \quad p(y) \rightarrow \circ_y \neg \text{pk}(y, A)$$

The fixed point utterance pair is:

$$ U = U' = \{\text{says}_{(27)} \circ_A \text{says}_{(28)} \circ_B \neg \text{pk}(B, A), \text{says}_{(28)} \circ_B \neg \text{pk}(B, A)\} $$

It can be shown that $A$ conforms to $\{27\}$. What about $B$? It is clear that $B$ does not conform to $\{28\}$, but what about $\{27\}$? Observe that $U \vdash \text{says}_{(27)} \circ_B \neg \text{pk}(B, A)$ (using the transfer axiom $A28$), but $(S, \text{Reg}) \not\models (U, U') \neg \text{pk}(B, A)$. Hence, $B$ does not conform to $\{27\}$. In other words, the statement (27) conveys an obligation to $A$ and if $A$ conforms, the embedded obligation is conveyed to $B$. As we noted in Section 4.2.2, we are formalizing the notion of *speaking on someone’s behalf*, i.e., the obligation (28) issued by $A$ is understood as being on behalf of the issuer of (27). Some applications may need a distinction between the different senses of saying.

**Example 2:** Consider the following example:

$$(29) \quad \text{You are required to allow a patient to see his records.}$$
By our analysis, (29) is an obligation on the hospital to provide a permission. Suppose that a hospital introduces such a permission in its policy. Has it conformed to (29)? The problem arises in distinguishing between claimed permission, and actual permission. A hospital claims that it permits a patient to see his records, by making an appropriate rule. On the other hand, a hospital actually permits a patient to see his records, by taking an action, e.g., sending the records via mail.

We suggest that a formalization of actual permission needs notions of bringing about or seeing to it that (e.g., [17, 70]). If a principal A says that she permits an action p, we need to check if she prevents p either by some other action or non-action. We can capture such notions by introducing laws that require facilitation. For example, (id) says \( (A) \Rightarrow (B)(p) \mapsto \phi \), where \( \phi \) is understood as a prerequisite for p, which is in the control of A. Thus, actual permission can be determined during conformance checking. However, listing the prerequisites for all the actions is quite difficult in practice, and notions of control and responsibility could lead to a more elegant solution.

4.4.3 Speech Act Theory

Speech act theories are concerned with providing an informal but precise account of the consequences of an utterance. The study of speech acts was initiated by Austin [8], and has subsequently received much attention (cf. [133]). In this section, we briefly discuss connections between speech act theory and the logic that we have developed.

Austin [8] distinguished between three kinds of acts that are associated with an utterance – (a) locutionary acts, (b) illocutionary acts, and (c) perlocutionary acts. The three types of acts are best understood using an example. Suppose John and Mary are getting married in a church, and the priest utters the words “I pronounce you man and wife”. The locutionary act is the construction of sounds in making the utterance. The illocutionary act is that John and Mary are married (according to the church and/or state). And, the perlocutionary act is that “People who witness
the wedding consider John and Mary to be married”. How are these acts reflected in the logic? The locutionary acts are obtained during the creation of utterances via policies (Section 4.3.3). The illocutionary and perlocutionary acts/effects are obtained via inferences, as we discussed in the context of reasoning about rights in Section 4.3.2. Note that for the illocutionary and perlocutionary effects to be obtained in the example above, we need corresponding utterances from the church or state, and the people witnessing the wedding, to the effect that they permit the priest to say (on their behalf) that John and Mary are married. The logic does not distinguish between illocutionary and perlocutionary effects.

Searle [135] classifies illocutionary acts into the following types:

(a) **Assertives**, which are used to convey the truth of the expressed proposition. For example, “John has good credit”.

(b) **Directives**, which are used to guide behavior. Examples include commands, advice, etc.

(c) **Commissives**, which commit the speaker to some future action, e.g., promising or making a bet.

(d) **Expressives**, which convey an attitude to a proposition. For example, “I am glad that John and Mary are married”.

(e) **Declaratives**, which change reality in accordance with the proposition of the declarion, e.g., baptisms, pronouncements, etc.

Four of these illocutionary effects that can be achieved using statements in the logic. Facts are *asserted*. For example, a branch of a company may assert the creditworthiness of customers. Obligations give rise to *directives*. The granting of power, via permission to speak, is a *commissive*, as the principal granting the power is committed to making a statement if the power is exercised. For example, if the state permits a priest to say that John and Mary are married, and the priest says so, then we conclude that the state says that John and Mary are married. And, finally, the exercise of power corresponds a *declarative*, e.g., the priest declares John and Mary
to be married on behalf of the state.

We have connected the logic to speech act theory at a coarse-grained level. An investigation of more fine-grained connections is left to future work.

4.5 Conclusions

We have motivated and described a logic for access control and conformance. The focus was on the interaction between saying and permission, as needed for these applications. We proposed two axioms to characterize their interaction (Section 4.3.2), and showed how these axioms could be incorporated into a logic programming approach (Section 4.3.3).

A combined analysis of saying and permission yielded benefits to both applications. For access control, we find a way to avoid the problematic interaction between hand-off and classical reasoning. Our axioms yield a decidable logic with a complete semantics (Section 4.3.2), and we hope that they have intuitive appeal to the reader. For conformance, we provide a novel account of recursive notions of power. We showed, in Section 4.3.5, that conformance checking is decidable. In Section 4.3.6, we identified a fragment of the logic, called chain formulas, in which provability can be decided in polynomial time.

We believe that the joint study of access control and conformance is a rich area for research. In Chapter 6, we will identify several avenues for further inquiry.
Chapter 5

Annotating and Computing Logical Form

5.1 Introduction

Regulatory bodies are large and complex. Manually translating the regulation to logic, as needed for conformance checking, is time-consuming. A long term goal of our work is to use natural language processing (NLP) techniques to assist in this translation. The problem of translating natural language sentences to logic has been of interest for several years in linguistics (cf. [67]), and more recently in NLP [24, 149, 150]. There are many open problems in the design of an appropriate logic and in the translation procedure. In previous chapters, we have motivated and designed features in a logic to accommodate a sentential translation of regulation. In this chapter, we turn our attention to the translation procedure with two objectives:

1. 
   Identifying additional features that are needed in the logic – There are many features that are needed in a logic, to accommodate a sentential translation of regulation. While we have identified some features in previous chapters, an empirical approach is needed to determine whether there are important/frequent constructions which cannot be translated directly.
2. *Defining a component of the translation that can be evaluated* – Since the logic is a subject of research, it is difficult to make progress on the translation procedure at the same time. We will assume that the end goal is to translate sentences into a variant of the logic that we have developed in previous chapters. The short-term goal is to compute some representation from the text that could serve as an intermediate step in the translation to logic.

### 5.1.1 Approach

To achieve the aforementioned objectives, following recent works in NLP [149, 150], we approach the translation to logic by analogy to syntax-based machine translation.\(^1\) A syntax-based translation system can be decomposed into two interacting components:

(A) *The lexical component* provides (a part of) the translation of a word or phrase into logic. Function words may be translated into operators in the logic, e.g., *some* as an existential quantifier and *must* as an obligation modality. Other words, e.g., *donation* and *test*, may be translated as predicates w.r.t. an ontology or schema, as we saw in Chapter 3.

(B) *The structural component* rearranges the constituents of a sentence to be syntactically similar to the logical translation. We focus on this component of the translation, and discuss examples in the following sections.

**The Lexical Component:** Before we narrow our focus to the structural component, we briefly mention a few lines of research that have bearing on the lexical component. The problem of creating/extracting an ontology from text has been studied under a variety of guises. In NLP, there have been several efforts to create lexicons (both manually and automatically) that capture various relationships between words,\(^2\)

\(^1\)In NLP, machine translation refers to the task of translating sentences in one natural language to another, e.g., English to French. Approaches differ in what are considered the basic units of translations, e.g., words, phrases, or syntactic trees.

\(^2\)In the context of the research described here, a lexicon refers to a structured representation of the relationships between words, which can be used to facilitate the translation process.
such as, synonyms, antonyms, and hypernyms (see, for example, [31, 46, 121]). Such lexicons may be used to ensure, for example, that synonyms receive the same translation into logic and that hypernyms correspond to subclasses/subtypes. In requirements and knowledge engineering, the focus has been on extracting class hierarchies from text [27, 32, 109, 118], much like the schemas that we created (manually) for our case study in Chapter 3. The translation of a word or a phrase may be determined by aligning it to the ontology, e.g., a noun may refer to a class or an attribute of a class. There are many challenging problems in adapting such resources and techniques to assist in the translation of regulation to logic, and we leave an exploration of these issues to future work.

**The Structural Component:** In this work, we focus on the structural component of the translation of regulatory sentences to logic. We assume that sentences in regulation are expressed formally as statements of the form:

\[(id) \varphi \mapsto \psi\]

Furthermore, we assume that the language for expression preconditions (\(\varphi\)) and post-conditions (\(\psi\)) is an appropriate extension of the logic developed in Chapter 4. Note that the distinction between preconditions and postconditions is a non-trivial assumption, and entails, for example, that all existential quantification and negation is restricted to being within either the precondition or the postcondition. This distinction is not unique to our work, and most formalisms that accommodate exceptions to laws, e.g., [26, 56, 60, 105, 134, 136], make a similar distinction.

Given the distinction between pre/postconditions, our goal is to transform a regulatory sentence into a structure that lets us determine: (I) the constituents of a sentence that contribute to the pre/postcondition, and (II) the structure of the pre/postcondition. The structures that we use are called abstract syntax trees (ASTs), and we provide a definition using an example in Section 5.2. Linguistic theories have argued for the transformation of a sentence into a structure called logical form (LF) as an intermediate step in translating a sentence to logic [103]. LF encodes the resol-
tion of scope ambiguities. ASTs can be understood as a restricted kind of logical form for regulatory texts. An AST describes, for example, whether negation has scope within the precondition or postcondition of a law.

5.1.2 Contributions and Outline

We see our contributions as follows:

- Annotating ASTs – We develop a modest-sized corpus (of about 200 sentences) of regulatory sentences annotated with ASTs. To our knowledge, ours is the first study in annotating scope phenomena. We place our work in the context of prior work in Section 5.9.

- Computing ASTs – The main step in the computation of ASTs is the ordering or ranking of operators. We adapt learning models for ranking to compute ASTs. Features are designed by studying subproblems, such as, disambiguating between de re and de dicto interpretations. The algorithms are evaluated using adapted versions of metrics developed for parsing.

Sections 5.2-5.4 describe the annotation. In Section 5.2, we define ASTs using an example. The guidelines for annotating ASTs are discussed in Section 5.3. We then give a quantitative description of the corpus in Section 5.4.

Sections 5.5-5.7 describe the computation of ASTs. We begin, in Section 5.5, by giving an overview of our two-step approach to computing ASTs. We then design features for the two subproblems in Section 5.6. In Section 5.7, we consider the problem of finding the optimal AST. We design metrics to evaluate the computed ASTs, and compare the performance of different algorithms.

We evaluate agreement in Section 5.8, discuss related work in Section 5.9, and conclude in Section 5.10.
5.2 Abstract Syntax Trees

In this section, we describe abstract syntax trees (ASTs) using an example from the CFR Section 610.11:

(30) A general safety test for the detection of extraneous toxic contaminants shall be performed on biological products intended for administration to humans.

We will discuss the translation in logic and the AST for the fragment of (30) that appears in black. In most examples in this chapter, we will restrict attention to a fragment of a sentence by graying out the rest. We do this to focus attention on particular phenomena of interest, and to keep the figures to a manageable size.

**Translation in Logic:** The sentence (30) is formally expressed as:

\[
(30) \text{bio}_\text{prod}(x) \implies \text{O}_\text{m}(x)(\exists y : \text{test}(y) \land \text{proc}(y, \text{gen}_\text{saf}) \land \text{ag}(y, m(x)) \land \text{obj}(y, x))
\]

The predicates and function symbols are read as follows. bio\_prod(x) - “x is a biological product”. m(x) denotes the manufacturer of x. test(y) - “y is a general safety test (event)”. proc(y, gen\_saf) - “y is a general safety procedure”. ag(y, m(x)) - “the agent of y is m(x)”, and obj(y, x) - “the object of the event y is x”. The formalized version of the law is read as follows: “If x is a biological product, then the regulator says via law (30) that the manufacturer m(x) is required to perform a general safety test y which has x as its object”.

As we mentioned in Section 5.1, there is an important restriction to existential quantification. We assume that existential quantifiers have scope within the preconditions or postcondition of a law. In the example above, the quantifier (\(\exists y\)) has scope within the postcondition. The formal evaluation procedure for laws (which is similar

---

2 We assume a thematic role-based representation of events [47, 72]. The use of an existential quantifier for the event variable is an oversimplification. For the test procedures in Section 610 of the CFR, it is assumed that exactly one test of a particular type is carried out, and repeat tests are permitted in the case of errors. The assumption of uniqueness is used to prescribe additional properties of the test in separate sentences. We ignore such issues as they do not affect the ASTs.
to logic programming) uses fixed point techniques that do not generalize to existential quantifiers with scope over both the precondition and postcondition. Our guidelines for annotating ASTs aim to preserve this restriction. In the following sections, we will identify constructions for which we do not have an adequate formalization, and these will be left to future work.

**Abstract Syntax Tree:** The AST for (30) is shown in Figure 5.1. There are three types of nodes: terminals, pre-terminals, and internal nodes. We discuss each of these types below, and introduce notation that is useful in discussing ASTs. We remind the reader that we will only be producing the AST shown in Figure 5.1 and *not* the logical translation of the sentence given above. The translations in logic are given purely to motivate the annotation, and we leave the conversion of ASTs to logic to future work.

![Example of an abstract syntax tree (AST).](image)

Figure 5.1: Example of an abstract syntax tree (AST).

### 5.2.1 Terminals

Terminals form the leaves of the AST. There are three types of terminals – explicit tokens, implicit tokens, and variables. We briefly describe each of these types, using examples from Figure 5.1. *Explicit tokens* correspond to single words that appear in a
sentence, e.g., the leaves “A”, “product”, and “perform”. Implicit tokens are inserted during annotation, e.g., “all” and “Post”. We will motivate the use of the implicit tokens in Section 5.2.4. Finally, we have variables which correspond to a contiguous sequence of tokens which have moved, e.g., the leaves \( x \) and \( y \).

### 5.2.2 Preterminals

Preterminals are the parents of a contiguous sequence of one or more terminals. There are three types of preterminals, based on the function of the terminals that they dominate – operators, restrictors, and the main predicate. We illustrate these types using examples from Figure 5.1. Operators are labeled with their part-of-speech, e.g., “D” for determiner, “M” for modal, and “O” for other. The operators in Figure 5.1 are all single words, but multi-word operators are possible, e.g., only if. Implicit operators are introduced as needed. For example, the implicit determiner all is associated with “biological products”. The tokens corresponding to operators are written in sans serif typeface, and implicit operators are underlined. Restrictors are labeled “R”, e.g., the restrictor of the determiner “A” is the noun phrase “general safety test”. The main predicate is an unlabeled node and the parent of the rightmost terminal in the AST, i.e., the parent of the terminals “\( y \) be performed on \( x \)”.

### 5.2.3 Internal Nodes

There are two sorts of internal nodes – internal restrictors and other internal nodes.

Internal restrictors have a unique child which is the root of an AST, except that it need not contain a postcondition marker. Such internal restrictors are needed, for example, for prepositional phrase modification of noun phrases, and we will discuss examples in later sections.

Other internal nodes have at least 2 children. An internal node with \( n + 1 \) children corresponds to an \( n \)-ary operator. The first child of the internal node is the operator, and the remaining \( n \) children are its arguments (which may be preterminals or internal
nodes). We use the term *nuclear scope* to refer to the last \((n^{th})\) argument of the predicate, and the term *restrictor* to refer to any other argument. For example, the phrase “biological products” is in the restrictor of the implicit operator *all*, and the variable \(x\) is in its nuclear scope. The operators in Figure 5.1 have at most one restrictor. We treat co-ordinating conjunctions, e.g., *and*, *or*, and *but*, as operators which have multiple restrictors.

Non-unary operators typically bind a variable (displayed on the internal node, e.g. \(\lambda x\) for *all*). This indicates that the variable \(x\) in the nuclear scope is bound by the operator. Unary operators, e.g., *shall* and *Post*, do not bind a variable, and the corresponding internal node is labeled with \(\lambda\).

### 5.2.4 Implicit Tokens Revisited

Implicit tokens are used to describe operators for which there is no overt word or phrase in the sentence. The two main types of implicit operators are determiners and the postcondition marker. *The implicit determiner* “all” in Figure 5.1 denotes that the noun phrase “biological products” should be interpreted generically/universally.

*The postcondition marker* is the operator associated with the implicit token *Post*, and has the catch-all part-of-speech “O” for other. It corresponds to the postcondition of the logical translation of the sentence, i.e., \(O_{m(x)}(\exists y : \text{gen}_{saf_{test}}(y) \land \text{obj}(y, x))\).

In Figure 5.1, there is a unique postcondition marker. In general, we will use one postcondition marker for every matrix verb phrase, and as a result, sentences with coordinated matrix verb phrases will have a postcondition for each coordinate. We will discuss some other uses of implicit operators in later sections.

### 5.2.5 Notation for Scope Ordering

We develop some notation that is useful in describing ASTs. Given an AST for a sentences, we say that an operator \(o_i\) *scopes over* \(o_j\), denoted \(o_i \gg o_j\), if \(o_j\) appears in the nuclear scope of \(o_i\). For example, in Figure 5.1, we have *all* \(\gg\) *Post*, *all* \(\gg\) *shall*, *all*
In addition, we say that the restrictor of $o_i$ scopes over $o_j$, denoted $R(o_i) \supseteq o_j$, if $o_j$ appears in the restrictor of $o_i$. Such configurations occur, for example, with PP-modification of noun phrases, and we will discuss examples in later sections.

### 5.3 Operators

In this section, we discuss the operators whose scope we are interested in, and our guidelines for determining which operators scope over others. The operators are divided into four types – auxiliary verbs (Section 5.3.1), determiners (Section 5.3.2), clause and verb phrase modifiers (Section 5.3.3), and coordinating conjunctions (Section 5.3.4). In addition to the operations, in each section, we present guidelines to determine the *scopes over* relation w.r.t. other nodes in the AST. The guidelines were developed by making several passes over sentences from Section 610 of the FDA CFR.

#### 5.3.1 Auxiliary Verbs and The Postcondition Marker

The auxiliary verbs in a matrix clause mark the start of the postcondition, and are placed directly below the postcondition marker in the AST (in the order in which they appear in the sentence). The only exception is when negation is involved, and negation can appear before, between, or after all auxiliaries. Consider the following sentences, where the operators of interest are shown in *sans serif* typeface:

(31) The general safety test shall be performed as specified in this section, unless modification is prescribed in the additional standards for specific products, or variation is approved as a supplement to the product license under Sec. 610.9.

(32) The general safety test is required in addition to other specific tests prescribed in the additional standards for individual products in this subchapter, except that, the test need not be performed on those products listed in paragraph (g) of this section.
Upon notification by the Director, Center for Biologics Evaluation and Research, a manufacturer shall not distribute a lot of a product until the lot is released by the Director, Center for Biologics Evaluation and Research.

The relevant fragments of the ASTs for (31)-(33) are shown in Figure 5.2, from left to right. In the AST for (31), we have Post $\gg$ shall $\gg$ be. Since (32) conveys a permission, we have Post $\gg$ not $\gg$ need $\gg$ be, and need is understood as obligation in this sentence. By contrast, (33) conveys a prohibition, and we have Post $\gg$ shall $\gg$ not.

Figure 5.2: A sequence of auxiliary verbs or negation appear directly below the postcondition marker.

### 5.3.2 Determiners – De Re and De Dicto

The most frequent types of determiners in the CFR are: (a) universal or generic, e.g., every, all, (b) existential, e.g., some, and (c) deictic or bound, e.g., the.\(^3\) We will focus on

\(^3\)“The” is interpreted deictically if it draws its interpretation from the context or world knowledge, e.g., in the phrase “the FDA. A bound interpretation is possible when it is dependent on another determiner, e.g., “the test dose for each rabbit”, in a case where each $\gg$ the. We note that “the” can also have a universal interpretation, e.g., “the donations”, and in such cases, the guidelines for universal determiners apply.
on these three types of determiners, and discuss other determiners at the end of this section.

Most of the sentences in the CFR are normative, i.e., they convey obligations and permissions via, for example, the words must and may. Given a determiner $d$ and a modal (auxiliary) $m$, we need to decide whether the determiner has de re scope ($d \gg m$) or de dicto scope ($m \gg d$). The default guidelines for the three types of determiners are as follows:

- A universal or generic determiner $d$ has de re scope, unless there is another operator $w$ that has de dicto scope and $w \gg d$ in the logical translation of the sentence.

- An existential determiner $d$ has de dicto scope. We do not allow for de re scope in this case.

- We do not analyse deictic or bound determiners as having scope properties, and their position in the AST respects the syntax of the sentence with the proviso that all operators that are needed to obtain the reference (for bound determiners) have scope over it. In effect, we defer the analysis of deixis to the (manual) procedure for translating an AST into logic.

The de re scope of universals and the de dicto scope of existentials are consequences of the formalism that we are working with. It may be possible to design a logic that accommodate other scope orders\(^4\), and the AST for such a logic would be different.

The main obstacle in applying the above guidelines is that determiners, e.g., a and any, are ambiguous. Even if we are given the context in which a sentence occurs, there are different translations to logic depending on whether we interpret a determiner generically or existentially. In such cases, we adopt default rules for assigning

\(^4\)To our knowledge, there has not been an attempt to design a logic for conformance checking that gives de dicto scope to universals, while accommodating priorities of exceptions over (default) rules.
interpretations to determiners based on the type of noun phrase or the syntactic configuration. We discuss some of the frequent cases.

Nominalizations: A noun phrase is said to be a nominalization if it denotes an event or a property (approximately, if there is an equivalent verb or adjective phrase). Consider again the sentence (fragment) from CFR Section 610.11:

(30) A general safety test for the detection of extraneous toxic contaminants shall be performed on biological products intended for administration to humans.

The noun phrase a general safety test in (30) is a nominalization. The determiner a is ambiguous between the existential reading (for each product, there must be a test), and a definite reading (a specific kind of test must be performed). We choose the existential reading as the default, and assign de dicto scope to the determiner a, i.e., must ≫ be ≫ a, as shown in Figure 5.1. The motivation for this decision is to treat such sentences isomorphically to the cases where the verb equivalent of the noun is used:

(34) You are not required to test donations of Source Plasma for evidence of infection due to the communicable disease agents listed in paragraphs (a)(5) and (a)(6) of this section.

The AST for (34) is shown in Figure 5.3. Observe that in the AST for (30) in Figure 5.1, and in the AST for (34) in Figre 5.3, the word test appears de dicto, thereby treating the verb and noun forms of “test” isomorphically. In addition, we note that the noun “donations” in (34) is also a nominalization, but it occurs de re, because the implicit determiner associated with the noun is a universal one (all in Figure 5.3). We will discuss the annotation of prepositional phrase modification of noun phrases, e.g., “of source plasma” modifying “donations” in (34), after we consider a few more examples of nominalizations.

The reader may have noticed the implicit operator “VP” in Figure 5.3. We use such implicit operators to provide syntactic structure, and in this case, “VP” corresponds to the verb phrase complement of “required”. We do not claim that verb phrases
are scope taking operators. However, they are annotated as operators, because it simplifies the definition of ASTs. Notice that the operator “all” moves out of the verb phrase, and takes scope over it, i.e., all $\gg$ VP in Figure 5.3. This is because verb phrases are not islands for quantifier scope. However, clauses are typically islands. The only syntactic operators that we have found the need for are clauses, verb phrases, and adjective phrases. In other cases, the necessary syntax is encoded in the AST.

Let us now return to the discussion about nominalizations. The descriptions of tests for products follow a common pattern in the CFR. The sentences in a section describing a test can be divided into the following types:

**S1** A sentence requiring the performance of the test for all products, such as (30) above

**S2** Sentences providing exemptions from performing the test, and

**S3** Sentences describing the procedure for a particular product

The guidelines that we have discussed above suffice for the nominalizations in the sentences of type **S1** and **S2**. The following are sentences of type **S3** that appears after (30) in Section 610.11 of the CFR:
The general safety test shall be conducted upon a representative sample of the product in the final container from every final filling of each lot of the product.

The duration of the test shall be 7 days for both species, except that a longer period may be established for specific products in accordance with Sec. 610.9.

We consider the problem of assigning scope to the uses of the determiner “the” in (35) and (36). In Section 610.11 of the CFR, the noun phrase “the product” in (35) does not refer to a contextually salient product. Rather, we have a sentence of type S1 which requires the performance of tests on certain types of products. We understand (35) as a paraphrase for the following:

\[
\text{(35) If a general safety test is required to be performed on a product (via another law), then the general safety test shall be conducted upon a representative sample of the product in the final container from every final filling of each lot of the product.}
\]

The paraphrase is translated into logic as follows:

\[
\text{gen}_{\text{saf}}(x) = \exists y : \text{test}(y) \land \text{proc}(y, \text{gen}_{\text{saf}}) \land \text{ag}(y, m(x)) \land \text{obj}(y, x)
\]

\[
\text{(35) prod}(x) \land \text{say}_{\text{FDA}}(\text{O}(x) \text{gen}_{\text{saf}}(x)) \iff \text{O}(x) (\exists z : \text{samp}(z, x) \land \text{gen}_{\text{saf}}(z))
\]

The schema \(\text{gen}_{\text{saf}}(x)\) is read as “a general safety test is performed on \(x\) by the manufacturer \((m(x))\)”. The subformula \(\text{say}_{\text{FDA}}(\text{O}(x)\text{gen}_{\text{saf}}(x))\) in the precondition checks whether the test is required via another law, and if so, the law (35) requires the test to be performed on a sample of the product.

Note that “the general safety test” and “the product” are both interpreted de dicto in this analysis. The contribution of “the general safety test” is the schema instance \(\text{gen}_{\text{saf}}(z)\) in the postcondition, and the contribution of “the product” is the variable \(x\) in the postcondition. The differing translations of noun phrases introduced by the is an instance of the broader problem of the interaction between naming and modalities [86].
“The product” is understood as a rigid designator for a particular product, and is translated to a variable. “The general safety test” is not rigid, and refers to a test that is required (but may not exist in non-conforming implementations). It is possible to treat these differences in translation as a problem for scope. However, to our knowledge, current linguistic theories treat it as an orthogonal problem to scope, i.e., as a problem of sense (cf. [125]).

**Prepositional Phrase Modifiers:** We revisit an example that we discussed in the context of nominalizations:

(34) You are not required to test donations of Source Plasma for evidence of infection due to the communicable disease agents listed in paragraphs (a)(5) and (a)(6) of this section.

The logical translation of (34) is given by:

\[(34) \text{don}(x) \land \text{mat}(x, \text{SPlasma}) \rightarrow \neg \text{O}_{bb(x)}(\exists y : \text{test}(y) \land \text{obj}(y, x))\]

The predicates are read as follows. \(\text{don}(x)\) - \(x\) is a donation. \(\text{mat}(x, \text{SPlasma})\) - the material of \(x\) is source plasma. \(\text{test}(y)\) - \(y\) is a test. \(\text{obj}(y, x)\) - the object of \(y\) is \(x\). \(bb(x)\) denotes the bloodbank that collected the donation \(x\).

The noun phrase “donations of source plasma” forms the precondition of the law. The noun “donations” contributes the predicate don, the preposition “of” contributes the predicate mat, and the noun phrase “Source Plasma” contributes the constant symbol “SPlasma”. We interpret the noun phrase “donations of source plasma” as being equivalent to “all donations of the source plasma kind”, with the determiners “all” and “the” being implicit. The AST is shown in Figure 5.3, and the (implicit) determiner “the” has scope inside the restrictor of the (implicit) determiner “all”, denoted \(R(\text{all}) \gg \text{the}\). This ordering respects the surface ordering of the determiners, i.e., \(\text{all}\) appears before \(\text{the}\) both in the noun phrase and in the AST, and we use this as the default scope for prepositional phrase modification.

We now define the default scope schematically, and then discuss examples which do not obey this scope. Figure 5.4 shows an example parse tree of a noun phrase with
Figure 5.4: Example parse tree of a noun phrase with nested prepositional phrase modifications. $d$ with subscripts denote determiners, $n$ with subscripts denote nouns and $p$ with subscripts denote prepositions.

nested prepositional phrase modifications. There are four determiners $d$, $d_1$, $d_2$ and $d_3$ whose scope we are interested in. The default AST for the noun phrase respects the surface order of the determiners, and is shown in Figure 5.5. Since $d_1$ is associated with a prepositional phrase modifying the noun $n$ (which is in turn associated with $d$), $d_1$ has scope within the restrictor of $d$, denoted $R(d) \gg d_1$. Similarly, we have $R(d_1) \gg d_2$, and $R(d) \gg d_3$.

Orders that are different from the surface order are typically needed when an embedded determiner is *universal*. We illustrate this with an example to explain our annotation procedure for ambiguous determiners. Consider the following sentence from CFR Section 610.2:

(37) Samples of any lot of a licensed product, except for radioactive biological products, together with the protocols showing results of applicable tests, may at any time be required to be sent to the Director, Center for Biologics Evaluation and Research.
Figure 5.5: Default AST for the noun phrase in Figure 5.4. The determiners $d$, $d_1$, $d_2$, and $d_3$ appear in the same order as they do in the parse tree/sentence, corresponding to a surface scope ordering.

The translation in logic is as follows:

$$\text{sent}(x) = \exists z : \text{samp}(z, x) \land (\exists e : \text{send}(e) \land \text{ag}(e, m(x)) \land \text{obj}(e, z) \land \text{recep}(e, \text{dir}))$$

(37) $\text{lot}(x) \land (\exists y : \text{prod}(y) \land \text{material}(x, y)) \mapsto P_{\text{dir}} \text{says}_{l(\text{dir})} O_{m(x)} \text{sent}(x)$

sent($x$) is read as “some samples of $x$ are sent by the manufacturer ($m(x)$) to the Director (dir)”.

The law (37) conveys a power via the nested modalities may be required, which is formalized in the postcondition as “the Director (dir) is permitted to say via her laws ($l(dir)$) that the manufacturer ($m(x)$) is required...”.

We refer the reader to Chapter 4, where we discuss examples that involve reasoning about powers.

The AST for (37) is shown in Figure 5.6. The main decisions involve the noun phrase “samples of any lot of a licensed product”. The determiner “any” is interpreted generically, as the Director has the power to require that any lot that she chooses be sent to her. The determiner “a” can be interpreted universally or existentially, and we choose the existential interpretation to preserve the surface scope w.r.t. “any”. As
a result, the noun phrase “samples of any lot of a licensed product” is understood as “some samples of every lot of some licensed product”. By our guidelines for universal and existential determiners, the noun phrase “every lot of some licensed product” is given de re scope, while “some samples” is given de dicto scope.

The scopal interactions between the determiners involved in prepositional phrase modification has been extensively studied in linguistics. The term inverse linking [102] is used to describe cases where a determiner within a prepositional phrase has scope outside the enclosing noun phrase, e.g., any ≫ some in Figure 5.6. Larson [90] has proposed a constraint that no other scopal operators occur between inversely linked determiners. However, in Figure 5.6, we have any ≫ may ≫ some, and “may” occurs between the inversely linked determiners, violating the constraint. The violation occurs because of our guideline to treat universal determiners de re and existential determiners de dicto. In the examples that we have annotated so far, all violations of Larson’s constraint involve the de re-de dicto distinction, and in other cases, we adhere to the constraint. We do not make any claims about the general applicability of Larson’s constraint, as the violations we observe may be an artifact of the restrictions
in the logic that we are translating to. An investigation of these issues is beyond the scope of this work.

**Other cases:** We have discussed the scope of universal, existential, and deictic determiners w.r.t. the modal auxiliaries. There are other kinds of determiners and noun phrases, and we now discuss two examples to illustrate some of the issues. Consider the following sentence:

(38) No lot of any licensed product shall be released by the manufacturer prior to the completion of tests for conformity with standards applicable to such product.

(38) is translated into logic as follows:

\[
\text{released}(x) = \exists z : \text{rel}(z) \land \text{ag}(z, m(x)) \land \text{obj}(z, x)
\]

\[
\text{lot}(x) \land (\exists y : \text{lic}\_\text{prod}(y) \land \text{material}(x, y)) \rightarrow O_{m(x)} \neg \text{released}(x)
\]

released(x) is read as “x is released by the manufacturer (m(x))”. The determiner “no” makes two contributions to the translation – (I) A universal quantification over lots in the precondition (lot(x)), and (II) A negation in the postcondition to convey the prohibition. The AST for (38) is shown in Figure 5.7. To describe both the contributions of no, we split it into two operators \(\forall_{\text{No}}\) denoting the universal quantification (associated with the overt word no), and \(\neg_{\text{No}}\) denoting the negation (assumed to be implicit). Following our previous guidelines, \(\forall_{\text{No}}\) is given *de re* scope, and \(\neg_{\text{No}}\) is given *de dicto* scope. The splitting of no into two operators is motivated purely by the formalism we are translating into. Other cases of split operators arise with certain adverbs, e.g., “only” and “at most”.

We conclude this section with an example involving a measure phrase:

(39) Inject intraperitoneally 0.5 milliliter of the liquid product or the reconstituted product into each of at least two mice, and 5.0 milliliters of the liquid product or the reconstituted product into each of at least two guinea pigs.
Figure 5.7: AST for (38)

(39) is translated into logic as follows:

\[ \text{inj}(x, y) = \exists z : 0.5\text{ml}(z, x) \land \exists e : \text{inject}(e) \land \text{ag}(e, m(x)) \land \text{obj}(e, z) \land \text{pat}(e, y) \]  

\[ (39) \quad \text{liq-prod}(x) \mapsto O_{m(x)} \exists Y : \text{mice}(Y) \land (|Y| \geq 2) \land (\forall y : y \in Y \Rightarrow \text{inj}(x, y)) \]

inj(x, y) is read as “the manufacturer (m(x)) injects 0.5ml of x into y”. The quantification over (appropriate) liquid products in the precondition of the law and the obligation in the postcondition are assumed to be implicit.

The AST is shown in Figure 5.8. First, consider the phrase each of at least two mice. In the logical translation, this phrase is mapped to \( \exists Y : \text{mice}(Y) \land (|Y| \geq 2) \land (\forall y : y \in Y \Rightarrow \text{inj}(x, y)) \). “At least two” is understood as a quantification over sets \( Y \) of cardinality greater than or equal to 2, and “each” quantifies over the elements in this set. Ideally, we would treat “at least two” as a determiner in the AST in Figure 5.8. However, for simplicity, we mark noun phrases with complex determiners with the Penn Treebank syntactic category “QP” (for quantifier phrase). The entire noun phrase “at least two mice” appears in the restrictor of this “QP”. Observe that this QP outscopes the universal determiner “each” in the logical translation of the sentence.
Thus, “each” has de dicto scope in Figure 5.8. Next, consider the measure phrase “0.5ml of the liquid product”, which is translated to (∃z : 0.5ml(z, x) ∧ ...). Here, again, we annotate the syntactic category QP and leave the determiner unspecified.

A proper analysis of such measure phrases needs a refined representation of objects. For example, if we inject 0.5ml of a liquid product into an animal, the volume of the product reduces by 0.5ml. Exceptions can also apply to amounts or measures:

(40) The test dose for each rabbit shall be at least equivalent proportionately, on a body weight basis, to the maximum single human dose recommended, but need not exceed 10 milliliters per kilogram of body weight of the rabbit.

(40) conveys requires the test dose for a rabbit to be the minimum of (I) the equivalent proportionately, on a body weight basis, to the maximum single human dose recommended, and (II) 10 milliliters per kilogram of its body weight. In the logic that we have developed, exceptions are binary in the sense that they either apply or do not apply. Examples, such as (40), suggest that there needs to be a notion of scale or degrees to which exceptions apply. We leave an investigation of these logical
aspects to future work.

5.3.3 Clause and Verb Phrase Modifiers

In addition to auxiliary verbs, we consider the following types of clause and verb phrase modifiers:

- Conditional - which are introduced, for example, by “if”.
- Exceptional - which are introduced by “except for”, “notwithstanding”, etc.
- Temporal - which are introduced by “before”, “after”, etc.
- Purpose - which are introduced by “for”, “so as to”, etc.
- Other - which have a specialized function in the CFR, and are typically introduced by as and in accordance with.

We begin by discussing examples to give an intuition of the issues involved, and then generalize from the examples to form guidelines for annotation.

Example 1: Consider the following sentence from Section 610.1 of the CFR:

(41) Each applicable test shall be made on each lot after the completion of all process of manufacture...

We begin by discussing the logical translation of (41). There are two difficulties – (A) the translation of “applicable”, and (B) the translation of the temporal modifier introduced by after. We discuss each of these in turn.

The phrase “applicable test” is understood as follows. Section 610 of the CFR prescribes tests for various purposes, e.g., “ensuring general safety” and “detecting HIV”. Given a lot \( x \) and a test purpose \( z \), a test is applicable to \( x \) for \( z \) iff \( x \) is required (via other laws) to be tested for \( z \). Formally, we define the following schemas:

\[
\text{test}_{\text{for}}(x, z) = \exists y : \text{test}(y) \land \text{obj}(y, x) \land \text{purp}(y, z)
\]
\text{appl\_test\_for}(x, z) = \text{says}_{l(FDA)}O_{m(x)}(\text{test\_for}(x, z))

\text{test\_for}(x, z) \text{ is read as “}x\text{ is tested for }z\text{”, and } \text{appl\_test\_for}(x, z) \text{ is read as “a test is applicable to } x \text{ for } z\text{”. The use of says in the schema } \text{appl\_test\_for}(x, z) \text{ captures the intuition that a test is applicable if it is required by other laws } (l(FDA)).

Next, we consider the temporal modifier. (41) is paraphrased as follows:

(41) For each lot \(x\), if all processes of the manufacture of \(x\) have been completed at time \(t\), then each applicable test shall be made on \(x\) after \(t\).

Thus, we need to check if all processes of manufacture have been completed, and if so, we assign the time of completion to \(t\). The logic needs to be extended to accommodate such assignments of times. We sketch how it can be defined:

\[
\text{compl}(x) = \forall y : \text{proc\_of\_manuf}(y, x) \Rightarrow \text{complete}(y, x)
\]

\[
\text{compl\_time}(x) = \sup\{\text{end\_time}(y, x) \mid \text{proc\_of\_manuf}(y, x)\}
\]

\[
\text{assign}(t, x) = (\text{compl}(x))? (t := \text{compl\_time}(x); \top) : \bot
\]

\text{compl}(x) \text{ is read as “all processes of the manufacture of } x \text{ have been completed”}. \text{compl\_time}(x) \text{ is a function denoting the time of completion (which is the supremum of end times of all the processes).} \text{assign}(t, x) \text{ is read as “if } \text{compl}(x)\text{, assign the completion time to } t \text{ and return true, else return false”}.

We are now ready to complete the translation:

\[
\text{test\_after}(x, z, t) = \exists y : \text{test}(y) \land \text{obj}(y, x) \land \text{purp}(y, z) \land \text{time}(y) \geq t
\]

\[
(41) \text{appl\_test\_for}(x, z) \land \text{lot}(x) \land \text{assign}(t, x) \mapsto O_{m(x)}(\text{test\_after}(x, z, t))
\]

The schema \text{test\_after}(x, z, t) \text{ is read as “}x\text{ is tested for } z \text{ after time } t\text{”}.

We now turn our attention to the AST for (41) shown in Figure 5.9. In the logical translation, “after” is analysed in a similar way to \textit{de re} determiners. There are two pieces – (A) The conditional assignment of the time in the precondition (assign\((t, x))
analogous to the position of a de re determiner and its restrictor, and (B) The comparison of times in the postcondition \((\text{time}(y) \geq t)\) in the schema test \(\text{after}(x, z, t)\) analogous to the variable associated with the determiner in the nuclear scope. As a result, in Figure 5.9, \(\text{after} \gg \text{Post}\) and “the completion of all processes of manufacture” appears in the restrictor of \(\text{after}\), i.e., \(R(\text{after}) \gg \text{the}\).

We note that for (41) a de dicto analysis of “after” is also possible. However, we prefer the de re interpretation as it keeps the logic for the postconditions simple. Since the postconditions enter into validity tests (e.g., for exceptions), we need the postcondition logic to be decidable. Preconditions are evaluated against a state or trace of an organization’s operations, and we need only the decidability of the model-checking question.

**Example 2:** We now consider examples of modifiers that are given de dicto scope:

(42) The contents of a final container of each filling of each lot shall be tested for identity after all labeling operations have been completed.

(43) The general safety test shall be performed as specified in this section.
The relevant operators are in **sans serif** typeface, and the restrictors are *italicized*. In (42), the operator “for” is given *de dicto* scope, i.e., \textbf{Post} \gg \textbf{shall} \gg \textbf{be} \gg \textbf{for}, and in (43), the operator “as” is given *de dicto* scope. We do not know if there is an appropriate *de re* interpretation for such operators. In addition, we note that there is a strong preference for such operators to appear (immediately) after the main verb of the sentence, e.g., “for” appears after “tested” in (42). By contrast, the conditional, temporal and exceptional operators can also appear at the start of the sentence. This syntactic preference suggests that the *de re* interpretation is not available.

At this time, we only have an approximate analysis of the modifier introduced by “as” in (43). The sentence (43) is understood as a requirement to conform to the other laws in Section 610 of the CFR. In previous chapters, we studied conformance as decision problem, but it can also be understood as a predicate. For example, `conform(A,l(B))` can be used to denote the conformance of principal A to principal B w.r.t. the laws l(B). However, (43) also conveys a notion of topic, i.e., it is a requirement to conform to “the requirements about the general safety test”. The notion of conformance needs to be extended to capture this notion of topic, and we leave an investigation to future work.

**Guideline:** Modifiers can be understood as operators taking two arguments, written schematically as \textbf{op}(\varphi, \psi), where \varphi is the restrictor, and \psi is the nuclear scope. The operator \textbf{op} is from one of the five types – conditional, exceptional, temporal, purpose, and other. Using this schema, we divide operators (informally) into two classes:

- Operators for which \textbf{op}(\varphi, \psi) \textit{does not entail} \psi - The conditional, temporal and exceptional operators fall into this class. And, for such operators we use the *de re* scope.

- Operators for which \textbf{op}(\varphi, \psi) \textit{entails} \psi - Purpose and other operators typically fall into this class. And, for such operators we use the *de dicto* scope.

The operators that appear within the syntactic argument of a modifier typically have scope within the restrictor. The only exception is when a *de re* determiner appears
in the argument of a modifier that has \textit{de dicto} scope, in which case, the determiner
scopes over the operator.

5.3.4 Co-ordinating Conjunctions

Co-ordinating conjunctions, e.g., “\textit{and}”, “\textit{but}”, and \textit{or}, are analysed as operators that
have two or more (in the case of lists) restrictors. The restrictors can be constituents,
e.g., clauses, verb phrases, or noun phrases, or parts of constituents, e.g., nouns,
verbs, or adjectives. We discuss two examples to develop intuition, and then discuss
the guidelines.

Example 1: Consider the following sentence:

\begin{quote}
(44) Bulk and final container material shall be tested for sterility as described
above in this section, except as follows:
\end{quote}

In (44), \textit{and} is said to have a \textit{distributive} interpretation (cf. [148]), which means
that the sentence can be paraphrased by the following two sentences taken conjunc-
tively:

\begin{quote}
(44a) Bulk material shall be tested for sterility

(44b) Final container material shall be tested for sterility
\end{quote}

(44a) and (44b) can be translated into logic as follows:

\begin{quote}
(44a) \texttt{bulk\_mat}(x) \mapsto O_{m(x)}(\exists y : \texttt{test}(y) \land \texttt{obj}(y, x) \land \texttt{purp}(y, \textit{sterility}))

(44b) \texttt{final\_cont\_mat}(x) \mapsto O_{m(x)}(\exists y : \texttt{test}(y) \land \texttt{obj}(y, x) \land \texttt{purp}(y, \textit{sterility}))
\end{quote}

The predicates are read as follows: \texttt{bulk\_mat}(x) – \texttt{x} is bulk material, \texttt{final\_cont\_mat}(x)
– \texttt{x} is final container material, \texttt{test}(y) – \texttt{y} is a test, \texttt{obj}(y, x) – the object of \texttt{y} is \texttt{x}, and
\texttt{purp}(y, \textit{sterility}) – the purpose of \texttt{y} is to ensure sterility. Note that the postconditions
of both laws are identical, and the variation is in the precondion. In the AST shown
in Figure 5.10, “\textit{and}” appears in the restrictor of the implicit determiner \textit{“all”}, i.e.,

148
the noun phrase is interpreted as “all bulk and final container material”. We refer
the reader to [129] for a discussion of how distributive conjunctions are mapped from
logical form to logic.\(^5\)

\[
\begin{align*}
\lambda x_1 & \quad \lambda x_2 \\
\lambda x_1 & \quad \lambda x_2 \\
\lambda x_1 & \quad \lambda x_2 \\
\end{align*}
\]

Figure 5.10: AST for (44).

**Example 2:** Consider the following sentence:

(45) The inoculum and medium shall be mixed thoroughly and incubated at a
temperature of 30 to 35 deg.C for a test period of no less than 14 days and
examined visually for evidence of growth on the third, fourth, or fifth day,
and on the seventh or eighth day, and on the last day of the test period.

In (45), the conjunction “and” in the phrase “the inoculum and medium” has
a *collective* interpretation (cf. [148]), which is signalled by the verb “mixed”. In
this case, we cannot split the co-ordinated nouns into separate sentences, e.g., “the
inoculum shall be mixed” is ungrammatical. Note, however, that the second occurrence

\(^5\)Approximately, translating a distributive conjunction to logic involves creating two new ASTs.
Each new AST would contain one restrictor of the conjunction, and could be translated to logic
separately.
of “and” in (45) is distributive, and we can paraphrase the sentence with the following two sentences taken conjunctively:

(45a) The inoculum and medium shall be mixed thoroughly.

(45b) The inoculum and medium shall be incubated at a temperature of 30 to 35 deg.C, after mixing thoroughly.

In (45b), we add the temporal modifier after mixing thoroughly, because there is an addition temporal relation conveyed by and (paraphrasable by and then). We refer the reader to Section 5.3.3 for the logical translation of such modifiers. Here, we consider the other aspects of the translation:

\[
\text{mix}(y,Y) = \exists e : \text{mix}_\text{tho}(e) \land \text{ag}(e,y) \land \text{obj}(e,Y)
\]

\[
(45a) \text{liq}_\text{prod}(x) \land X = \{\text{inoc}(x), \text{med}(x)\} \mapsto \mathcal{O}_{m(x)}(\text{mix}(m(x),X))
\]

\[
\text{incub}(y,Y) = \exists e : \text{incub}(e) \land \text{ag}(e,y) \land \text{obj}(e,Y) \land 30^\circ C \leq \text{temp}(e) \leq 35^\circ C
\]

\[
(45b) \text{liq}_\text{prod}(x) \land X = \{\text{inoc}(x), \text{med}(x)\} \mapsto \mathcal{O}_{m(x)}(\text{incub}(m(x),X))
\]

The schema mix\((y,Y)\) is read as “\(y\) mixes \(Y\) thoroughly” and incub\((y,Y)\) is read as “\(y\) incubates \(Y\) at a temperature of 30 to 35 deg.C”. As with previous examples, we assume that there is an implicit universal quantification over appropriate liquid products in the translation of the paraphrases. The collective phrase, i.e., the inoculum and medium, is understood as denoting a set which represents their mixture \((X = \{\text{inoc}(x), \text{med}(x)\})\). We note that additional machinery may be needed to give a proper treatment of plurals and collectives (see, for example, [148]). However, in the CFR, the problems associated with collectives seem to be largely independent of scope, and we leave the details of their logic to future work.

The AST for (45) is left as an exercise to the reader. The scope of both conjunctions is determined by the scope of their syntactic arguments. We conclude this section by describing the guideline for coordinating conjunctions.
**Guideline:** The scope of coordinating conjunctions are determined by the scope of the operators in their syntactic arguments. The operators in both arguments often have the same scope. If they do not, the conjunction is given the scope of the lowest operator, and all other operators scope over the conjunction. For example, in the phrase “every man and his mother”, we would have every ≫ and.

### 5.4 Corpus and Distribution of Operators

In previous sections, we have described the annotation of ASTs using examples. In this section, we give a more quantitative description of the corpus. We classify operators into types and subtypes and discuss their distribution. This classification of operators plays a useful role in the computation of ASTs.

We have annotated ASTs on Section 610 of the CFR. A total of 195 ASTs are available. There are 6599 tokens (and 6013 tokens excluding punctuation) in these 195 sentences. Thus, we have an average of 33.8 tokens (and 30.8 tokens excluding punctuation) per sentence.

The operators in an AST are divided into the following types – determiners, modal auxiliaries, clause and verb phrase modifiers, coordinating conjunctions, and other operators (which includes negation, adverbs like “only”, and the postcondition marker “Post”).

<table>
<thead>
<tr>
<th>Type</th>
<th>Number of Instances</th>
<th>D</th>
<th>M</th>
<th>S/VP Mod</th>
<th>C</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit</td>
<td>1827</td>
<td>37.6%</td>
<td>20.7%</td>
<td>18.2%</td>
<td>15.2%</td>
<td>8.3%</td>
</tr>
<tr>
<td>Implicit</td>
<td>797</td>
<td>71.1%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>28.9%</td>
</tr>
</tbody>
</table>

Table 5.1: Types of Operators. D stands for “determiner”, M for “modal auxiliary”, S/VP Mod for “clause and verb phrase modifiers”, and C for “coordinating conjunctions”.

151
There are 2624 operators of which 1827 are explicit and 797 are implicit, as shown in Table 5.1. Explicit operators are those which are associated with a word or phrase. Implicit operators are used mainly for noun phrases without an explicit determiner, and the postcondition marker. The implicit operators comprise mainly of determiners (71.1%) and other operators (28.9%). Each type of operator is further divided into subtypes that are useful in computing the AST, and we will discuss these subtypes below.

<table>
<thead>
<tr>
<th>Number of Instances</th>
<th>Universal</th>
<th>Existential</th>
<th>Ambiguous</th>
<th>Deictic</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit</td>
<td>687</td>
<td>9.6%</td>
<td>0%</td>
<td>29.8%</td>
<td>60.6%</td>
</tr>
<tr>
<td>Implicit</td>
<td>562</td>
<td>17.1%</td>
<td>20.6%</td>
<td>0%</td>
<td>42.3%</td>
</tr>
</tbody>
</table>

Table 5.2: Subtypes of determiners.

Table 5.2 shows the distribution of the subtypes of determiners. Universal determiners contain the words “all”, “every”, and “each”, and comprise 9.6% of the explicit determiners, and 17.1% of the implicit determiners. Note that the implicit determiners are inserted during annotation, and determining the subtype is a sense disambiguation problem that needs to be resolved during the computation of the ASTs. The existential determiner “some” is never used explicitly, but comprises 20.6% of the implicit determiners. The determiners are “a” and “an” are ambiguous between existential and universal/generic senses, and comprise 29.8% of the explicit determiners. The deictic determiners include “the” and “those”. We note that “the” can also have a universal/generic flavor, e.g., “the products” can be used to mean “all the products”. The counts reported in Table 5.2 are based on the occurrence of strings, and do not distinguish between various senses.

Table 5.3 shows the subtypes of modal auxiliaries, which are all explicit. Obligation (35.8%) is signalled by “must”, “shall”, or “need”. Permission (10.6%) is signalled by “may”. Not all obligations and permissions are conveyed by auxiliary verbs. The
words “required” and “permitted” occur as main verbs, and are not treated as operators. Auxiliary forms of the verb “be” comprise 46.4% of all auxiliaries. And, the other auxiliaries (7.2%) mostly consist of forms of “have”.

<table>
<thead>
<tr>
<th>Number of Instances</th>
<th>Obligation</th>
<th>Permission</th>
<th>Be</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>378</td>
<td>35.8%</td>
<td>10.6%</td>
<td>46.4%</td>
<td>7.2%</td>
</tr>
</tbody>
</table>

Table 5.3: Subtypes of modal auxiliaries.

Table 5.4 breaks down clause and verb phrase modifiers into subtypes. Temporal and conditional modifiers (29.1%) include, for example, “before”, “after”, “if”, and “except as”. The purpose modifiers (28%) are signalled by “for”. Modifiers which refer to laws (30.2%) are introduced by “as”, “under”, and “according to”. The other modifiers include cases which we have not categorized yet, e.g., “regardless”, “notwithstanding”.

<table>
<thead>
<tr>
<th>Number of Instances</th>
<th>Temporal and Conditional</th>
<th>Purpose</th>
<th>References to Laws</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>333</td>
<td>29.1%</td>
<td>28%</td>
<td>30.2%</td>
<td>12.7%</td>
</tr>
</tbody>
</table>

Table 5.4: Subtypes of clause and verb phrase modifiers.

Table 5.5: Types of syntactic categories annotated. S stands for sentence or clause, VP stands for verb phrase, QP stands for quantifier phrase, and ADJP stands for adjective phrase.

<table>
<thead>
<tr>
<th>Number of Instances</th>
<th>S</th>
<th>VP</th>
<th>QP</th>
<th>ADJP</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>316</td>
<td>60.1%</td>
<td>16.8%</td>
<td>13%</td>
<td>6%</td>
<td>4.1%</td>
</tr>
</tbody>
</table>

In addition to the operators, there are some syntactic annotations in the corpus,
which are used to capture scope islands. Table 5.5 nodes gives the distribution of the syntactic categories used. Clauses (60.1%) mostly consist of relative clauses. Verb phrases (16.8) are used for the complements of certain verbs, e.g., “required” and “permitted”. We borrow the term quantifier phrase (13%) from the Penn Treebank, where it is used for noun phrases with complex determiners, e.g., “no more than 5”. The remaining cases comprise mostly of adjective phrases and possessive noun phrases.

5.5 Computing ASTs – Overview

In this section, we give an overview of our approach to computing ASTs. We will assume as given a Processed Parse Tree (PPT) of a sentence, with the operators and their restrictors identified. Examples are given below. Given such a PPT, the AST is computed in two steps:

1. Finding the preterminal at which an operator takes scope
2. Ordering the operators associated with a preterminal.

We begin by describing the second step in Section 5.5.1, and then consider the first step in Section 5.5.2. The steps are described in reverse order, because in most cases, the operators associated with a preterminal are determined directly by syntactic attachment.

5.5.1 Ordering Operators

The problem of learning to order a set of items is not new. Cohen et al. [35] give a learning theoretic perspective, and Liu [96] surveys information retrieval applications. The approach that we use can be seen as a probabilistic version of [35]. We will discuss other connections to the literature in Section 5.9.

We explain the step of ordering operators using an example. Consider again the following sentence from the CFR 610.13:
A general safety test for the detection of extraneous toxic contaminants shall be performed on biological products intended for administration to humans.

Input: We will assume as input a Processed Parse Tree (PPT) of a sentence, as shown in Figure 5.11. Given the correct parse of the sentence, the PPT is obtained by identifying the operators and their restrictors. For example, the determiner “a” has the restrictor general safety test. The phrase biological products has no explicit determiner associated with it, and the corresponding operator in the processed parse tree is labeled “IMP” for implicit. In addition, the postcondition marker “Post” is also identified. We will not investigate the computation of parse trees or their conversion to PPTs in this work.\(^6\)

There are two main types of nodes in the PPT – operators and preterminals. The nodes labeled with the symbol λ, e.g., \(\lambda_x\) and \(\lambda\), correspond to operators. The root of the PPT and the restrictors of the operators, are the preterminals.

We now introduce some notation which is useful in describing the computation of ASTs. We will use \(\tau\) (with subscripts) to denote PPTs, \(p\) (with subscripts) to denote preterminal nodes, and \(o\) (with sub/superscripts) to denote operators. A PPT \(\tau\) is viewed as a set of preterminal nodes, and we will write – (a) \(p \in \tau\) to denote that \(p\) occurs in \(\tau\), and (b) \(|\tau|\) to denote the number of preterminals in \(\tau\). A preterminal \(p\) is viewed as an ordered set of operators \(p = (o^1, \ldots, o^{|p|})\). For example, in Figure 5.11, the root preterminal \(p\) has \(|p| = 5\), and the operators \(o^1 = \text{Post}, o^2 = A, o^3 = \text{shall}\), and so on.

Output: Given the PPT in Figure 5.11, the goal is to compute the AST in Figure 5.12. This involves determining, for example, that the implicit determiner associated with biological products is universal, and hence, we have \(\text{IMP} \gg \text{Post}\). However, the determiner “A” associated with general safety test is interpreted existentially, and hence, we have \(\text{Post} \gg A\).

\(^6\)The PPTs, in this work, are obtained by removing all scope decisions from the AST. To a first approximation, we start by removing all operators from the AST, and then, replace the corresponding variables by the operators.
We now develop some notation to describe the scopal ordering of operators. Given a preterminal \( p = (o_1, \ldots, o_{|p|}) \), we use \( r(p) \), with subscripts, to denote an ordering or ranking of the operators in \( p \). A ranking \( r(p) = (i_1, \ldots, i_{|p|}) \) is a permutation of indices indicating the order in the AST, i.e., for all \( 1 \leq j < k \leq |p|, i_j \neq i_k \), \( 1 \leq i_j \leq |p| \) and \( 1 \leq i_j \leq |p| \). We will abuse notation and write \( r(p) = (o_{i_1}^1, \ldots, o_{i_{|p|}}^{|p|}) \) to denote a ranking of \( p \).

ASTs are denoted by \( \alpha \) (with subscripts). An AST \( \alpha \) contains a ranking of operators associated with each preterminal, denoted \( r_\alpha(p) \). Let \( p = (o_1, \ldots, o_5) \) be the root preterminal of the PPT in Figure 5.11. The reordering of the AST \( \alpha \) in Figure 5.12 is given by \( r_\alpha(p) = (o_1^2, o_5^2, o_3^3, o_4^4, o_5^5) \). For example, \( o_5^5 = A \) denotes that the determiner “A” appears second in the surface order (Figure 5.11) and fifth or lowest in the scope order (Figure 5.12). Similarly, \( o_5^5 = IMP \) denotes that the implicit determiner appears fifth or last in the surface order (Figure 5.11) and first or highest in the scope order (Figure 5.12). Note that given a preterminal node and an AST either the subscript or superscript suffices to uniquely identify the operator, and we will often use only the sub or superscripts. In the example, \( o^2 \) and \( o_5^5 \) both refer to the determiner “A”.

**Computing ASTs:** The computation of ASTs is investigated in a supervised
machine learning setting. Given a PPT $\tau$, let $A(\tau)$ be the set of all possible ASTs. Our goal is to find the AST which has the highest probability given the PPT:

$$\alpha^* = \arg \max_{\alpha \in A(\tau)} P(\alpha | \tau)$$

The conditional probability of an AST is defined as:

$$P(\alpha | \tau) = \prod_{p \in \tau} P(r_{\alpha}(p) | \tau)$$

$$P(r_{\alpha}(p) | \tau) = \prod_{i=1}^{\lfloor |p|/2 \rfloor} \prod_{j=i+1}^{\lfloor |p|/2 \rfloor} P(o_i \gg o_j | \tau)$$

In other words, $P(\alpha | \tau)$ is modeled as the ranking of each preterminal. And, the probability of a ranking is the product of the probabilities of the pairwise ordering decisions. The model falls under the class of pairwise ranking approaches [96].

As with all machine learning tasks, there are two steps. The training step involves estimating the probabilities $P(o_i \gg o_j | \tau)$. The search step is to use the estimated probabilities to find the best AST. We will consider the training in Section 5.6, and the search in Section 5.7.
5.5.2 Finding the Scope Preterminal

In the example that we discussed in the previous section, there were no embedded operators, i.e., an operator or its variable located in the restrictor of another. An embedded operator can either – (a) take scope within the restrictor of the embedding operator, or (b) outscope the embedding operator. To account for the second case, we need to determine whether it is appropriate to lift an embedded operator to a higher preterminal than the one to which it is associated syntactically.

We discuss an example to illustrate the problem. Consider the following sentence:

(37) Samples of any lot of a licensed product, except for radioactive biological products, together with the protocols showing results of applicable tests, may at any time be required to be sent to the Director, Center for Biologics Evaluation and Research.

The PPT and AST for (37) are shown in Figures 5.13 and 5.14 respectively. Consider the noun phrase “IMP Samples of any lot of a licensed product”. The three operators are related as follows in the AST: \textcolor{red}{any} \gg \textcolor{blue}{IMP} and \textcolor{red}{R(any)} \gg \textcolor{red}{a}. The important observation is that the embedded determiner “any” outscopes the implicit determiner “IMP” associated with \textcolor{red}{samples} in Figure 5.14, but it appears in the restrictor of \textcolor{blue}{IMP} in Figure 5.13. As a result, the PPT in Figure 5.13 cannot be converted to the AST in Figure 5.14 simply by ranking sibling operators (as we did in the previous section).

To resolve this issue, we will convert the PPT in Figure 5.13 to another PPT (shown in Figure 5.15). Observe that the determiner “any” is a sibling of the implicit determiner “IMP” in the PPT in Figure 5.15. As a result, this PPT can be converted to the AST by reordering sibling operators.

\textbf{Notation:} We now develop some notation to describe the process of converting the initial PPT to the second one. Given an initial PPT \( \tau \), a PPT \( \tau' \) is a possible second PPT iff (a) \( \tau' \) has the same preterminals as \( \tau \), and (b) if \( o \in p \) in \( \tau \) and \( o \in p' \) in \( \tau' \), then \( p' = p \) or \( p' \) is an ancestor of \( p \) in both \( \tau \) and \( \tau' \). The second condition ensures that there are only two possibilities for an embedded operator – (I) it remains in
Figure 5.13: PPT for (37)

Figure 5.14: AST for (37)
the restrictor of the embedding operator or (II) it outscopes the embedding operator. The set of possible second ppts for a given initial PPT $\tau$ is denoted by $T(\tau)$. Note that $\tau \in T(\tau)$, i.e., it is possible that the initial PPT remains unchanged.

Given an operator $o$, let $o_\pi$ denote the embedding operator of $o$ in $\tau$ if one exists (otherwise, $o_\pi$ is undefined). For example, given $o = \text{any}$ in Figure 5.13, $o_\pi = \text{IMP}$. However, given $o = \text{IMP}$, $o_\pi$ is undefined as $\text{IMP}$ is not embedded. Given an operator $o$ and $\tau' \in T(\tau)$, we use $p(o, \tau')$ to denote the preterminal such that $o \in p(o, \tau')$ in $\tau'$. Finally, we define two sets of operators. $[p(o, \tau') \downarrow o_\pi]$ is the set of operators encountered on the path from $p(o, \tau')$ to $o_\pi$, provided that $o_\pi$ exists and is a descendant of $p(o, \tau')$. Otherwise, $[p(o, \tau') \downarrow o_\pi]$ is taken to be the empty set. $[\downarrow p(o, \tau')]$ is the set of operators on the path from the root of $\tau'$ to $p(o, \tau')$.

**Computing the second PPT:** Given an initial PPT $\tau$, our goal is to find the second PPT which has the highest probability given the initial PPT:

$$\tau^* = \arg \max_{\tau' \in T(\tau)} P(\tau'|\tau)$$
The conditional probability of a second PPT is defined as:

\[
P(\tau'|\tau) = \prod_{o \in \tau} 1 \times \left( \prod_{o' \in [p(o, \tau') \downarrow o]} P(o \gg o'|\tau) \right) \times \left( \prod_{o' \in [[p(o, \tau')]]} P(R(o') \gg o|\tau) \right)
\]

The probability is expressed as a product of two inner products for each operator. The inner products are understood as follows. For each operator \(o\) and every AST that can be computed from \(\tau'\), we have \(o \gg o'\) for all \(o' \in [p(o, \tau') \downarrow o]\). For example, given \(o = \text{any}\) in Figure 5.15, we have \([p(o, \tau') \downarrow o] = \{\text{IMP}\}\), and we will ensure during reordering that \(\text{any} \gg \text{IMP}\). Similarly, the second inner product captures the fact that in all ASTs computed from \(\tau'\), we have \(R(o') \gg o\) for all \(o' \in [\downarrow p(o, \tau')]\). In Figure 5.15, given \(o = a\), we have \([\downarrow p(o, \tau')] = \{\text{any}\}\), and \(R(o') \gg o\) in all possible ASTs.

We do not impose any linguistically motivated restrictions in the transformation of the initial PPT to the second PPT, and subsequently, to the AST. Larson [90] discusses examples where unrestricted movement of embedded determiners can lead to ASTs (or logical forms) which do not correspond to any possible interpretation of a sentence. An investigation of appropriate restrictions is left to future work. We now turn to the steps of training and search in Sections 5.6 and 5.7 respectively.

5.6 Feature Design

In this section, we consider the training step, i.e., estimating the probabilities that are needed for computing the ASTs. The estimated probabilities are tested on appropriate subproblems. We begin, in Section 5.6.1, by giving an overview of binary classifiers, and narrow our focus to the problem of feature design. We also describe the experimental setup. Section 5.6.2 considers the problem of designing features for classifying operators as de re or de dicto, which is an important subproblem in ordering operators. In Section 5.6.3, we generalize these features to accommodate pairwise ordering of operators, and to handle embedded operators.
5.6.1 Binary Classifier

We give a brief overview of the classifier or model that we use to learn the desired probabilities. The reader familiar with discriminative classifiers may skim or skip this section.

We start with some notation and terminology that is useful in describing the experiments. An observation is a triple \( x = (o, o', \tau) \), where \( o \) and \( o' \) are operators in the PPT \( \tau \). The set of observations for a given experiment is denoted by \( X \). For example, we may consider observations where \( o \) and \( o' \) are associated with the same pre-terminal in \( \tau \). A binary labeling function \( l : X \times A \rightarrow \{0, 1\} \) associates a label with each observation given the AST. For example, an observation \( x = (o, o', \tau) \) may receive the label \( y = l(x, \alpha) = 1 \) iff \( o \gg o' \) in the AST \( \alpha \). Finally, a family of binary feature functions \( F \) are used to predict the label from the observation. Each feature function \( f_i \in F \) has the type \( f_i : X \rightarrow \{0, 1\} \). For example, we may have a feature such that given \( x = (o, o', \tau) \), \( f_i(x) = 1 \) iff \( o \) and \( o' \) are determiners and \( o' \) is in a prepositional phrase modifying \( o \).

The specific model that we will use is the log-linear model, in which the conditional probabilities have the exponential form described below. Such models have the nice property that the maximum likelihood estimate preserves the maximum entropy principle (see [116]), and they are commonly used for NLP tasks. Given an observation \( x \), we associate a weight \( w^y_{\Lambda}(x) \) for each label \( y \in \{0, 1\} \):

\[
w^y_{\Lambda}(x) = \exp \left( \sum_{f_i \in F} \lambda^y_i f_i(x) \right)
\]

Each \( \lambda^y_i \in \Lambda \) is a real-valued parameter associated with a particular feature for a particular label. For example, a high positive value for \( \lambda^y_i \) would indicate that the feature \( f_i \) has a strong correlation with the label \( y \). The conditional probability of the label given the observation is defined as:

\[
P_\Lambda(y|x) = \frac{w^y_{\Lambda}(x)}{w^0_{\Lambda}(x) + w^1_{\Lambda}(x)}
\]
**Training and Testing:** We now describe the experimental set-up that we will use in this section. The input is a set of labeled instances $I$, where each element of $I$ is a label-observation pair $(y, x)$. We partition $I$ into two disjoint sets $I_{tr}$ for training, and $I_{te}$ for test. During the training phase, the learning algorithm attempts to find the parameters which maximizes the probability of the training data:

$$\Lambda^* = \arg \max_{\Lambda} \prod_{(y,x) \in I_{tr}} P_{\Lambda}(y|x)$$

During the test phase, we evaluate the model by comparing the given label to the label predicted by the model. Given $(y, x) \in I_{Te}$, the predicted label is defined as:

$$y^* = \arg \max_{y' \in \{0,1\}} P_{\Lambda^*}(y'|x)$$

The accuracy of the model is the fraction of the test instances for which the predict label is correct, i.e., $y^* = y$. We note that there are well-established algorithms for training the models described here. We use the implementation provided by the Mallet toolkit [104]. Our only focus is on the design of appropriate features.

**Cross-validation:** We will use the method of cross-validation for all our experiments. Cross-validation is typically used when the number of instances are few, to avoid “lucky” choices of training and test data. In 10-fold cross-validation, the labeled instances $I$ are divided into 10 equal sized partitions ($I_1, I_2, ..., I_{10}$). We perform 10 iterations of training an testing. In the $j^{th}$ iteration, $I_j$ is treated as the test set, and $I - I_j$ is treated as the training set. We will report the average accuracy over the 10 iterations.

### 5.6.2 De Re vs De Dicto

In this section, we consider an important subproblem in the pairwise ordering of operators. The observations and labels are defined as follows:

1. Observations $x = (o, o', \tau)$ are such that there is a preterminal $p \in \tau$, $\{o, o'\} \subseteq p$, and $o' = \text{Post}$. In other words, we are considering operators $(o)$ that are siblings of the postcondition marker $(o')$.  

163
2. Labels \( y = l(x, \alpha) = 1 \) iff \( o \gg o' \) in \( \alpha \). Otherwise, \( y = 0 \).

If an observation \( x = (o, o', \tau) \) has the label 1, then \( o \) is said to have \textit{de re} scope, i.e., \( o \) outscopes the postcondition marker \( o' = \text{Post} \). Otherwise, \( o \) is said to have \textit{de dicto} scope. For example, in the AST in Figure 5.12, the implicit determiner “IMP” has \textit{de re} scope \( \text{IMP} \gg \text{Post} \), while the determiner “A” has \textit{de dicto} scope \( \text{Post} \gg A \).

<table>
<thead>
<tr>
<th>Operator Type</th>
<th>Number of Instances</th>
<th>De Re Scope Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determiner</td>
<td>277</td>
<td>59.9%</td>
</tr>
<tr>
<td>Modal Auxiliaries</td>
<td>268</td>
<td>0%</td>
</tr>
<tr>
<td>Clause/VP Modifier</td>
<td>132</td>
<td>68.2%</td>
</tr>
<tr>
<td>Coordinating Conjunctions</td>
<td>36</td>
<td>22.2%</td>
</tr>
<tr>
<td>Negation</td>
<td>33</td>
<td>0%</td>
</tr>
<tr>
<td>Other</td>
<td>74</td>
<td>17.6%</td>
</tr>
</tbody>
</table>

Table 5.6: De Re scope distribution. An operator has \textit{de re} scope iff it outscopes the postcondition marker.

We begin by looking at the distribution of \textit{de re} scope, to develop some intuition for the problem. Table 5.6 shows the percentage of each type of operator that has \textit{de re} scope. Modal auxiliaries and negation are unambiguous to this distinction, and always have \textit{de dicto} scope. Note that a type of operator with 50% occurring \textit{de re} is ambiguous, while 0% or 100% are unambiguous. Thus, from Table 5.6, we can conclude that determiners, and clause/VP modifiers are the most ambiguous types. And, more features are needed to disambiguate them.

**Determiners:** Table 5.7 shows the percentage of each subtype of determiner that has \textit{de re} scope. As expected, universal and existential determiners are unambiguous, while ambiguous and deictic determiners are more ambiguous. As we mentioned previously, the deictic determiner \textit{the} can have a universal interpretation in the phrase
<table>
<thead>
<tr>
<th>Determiner Type</th>
<th>Number of Instances</th>
<th>De Re Scope Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal</td>
<td>74</td>
<td>100%</td>
</tr>
<tr>
<td>Existential</td>
<td>12</td>
<td>0%</td>
</tr>
<tr>
<td>Ambiguous</td>
<td>50</td>
<td>28%</td>
</tr>
<tr>
<td>Deictic</td>
<td>127</td>
<td>53.5%</td>
</tr>
<tr>
<td>Other</td>
<td>14</td>
<td>35.7%</td>
</tr>
</tbody>
</table>

Table 5.7: De Re scope distribution for subtypes determiners.

*the products*. There are two main types of features that we need to learn the *de re-de dicto* distinction for determiners:

1. Features to predict whether ambiguous and deictic determiners are universal or not

2. Features to determine the type of implicit determiners – In Table 5.7, we assume that the type of implicit determiners are given. This assumption is unrealistic. Rather, we need to predict the type of such determiners, during the conversion of the PPT to the AST.

<table>
<thead>
<tr>
<th>Clause/VP Modifier Type</th>
<th>Number of Instances</th>
<th>De Re Scope Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temporal and Conditional</td>
<td>73</td>
<td>100%</td>
</tr>
<tr>
<td>Purpose</td>
<td>8</td>
<td>0%</td>
</tr>
<tr>
<td>References to Laws</td>
<td>33</td>
<td>0.9%</td>
</tr>
<tr>
<td>Other</td>
<td>29</td>
<td>65.5%</td>
</tr>
</tbody>
</table>

Table 5.8: De Re scope distribution for S & VP modifiers.
Clause and VP Modifiers: Table 5.8 shows the percentage of each subtype of modifier that has de re scope. Following the guidelines for annotation, the temporal and conditional modifiers are always de re, the purpose modifiers and modifiers conveying references to laws are always de dicto.

Features: We use the following (classes of) features for an observation $x = (o, o', \tau)$:

- TYPE - The type and subtype of the operator $o$ (only for explicit operators). For example, for the type determiner, we have the subtypes universal, existential, ambiguous, deictic and other. The reader is referred to Section 5.4 for the subtypes of the other operators.

- PRE-VERB and POST-VERB - Tracks whether $o$ appears before or after the main verb of the sentence. By default, the determiners before the verb are given de re scope and those that appear after the verb are given de dicto scope.

- PERF - Tracks whether the main verb is perform. The verb perform is frequent in the CFR, and the subject of perform is typically given de dicto scope, as it is the main predicate of the sentence.

- PRE-VERB + PERF - Conjunction of the previous two features

- POS - The part-of-speech of the head word, if $o$ is a DETERMINER. The Pos tags are obtained from the Stanford POS tagger (cite), which is trained on the Penn Treebank (cite). For example, for the noun phrase biological products, the head word is products, and the Pos is NNS (plural common noun). And, this Pos tag may indicate that the noun phrase is interpreted generically/universally.

Experiments: We evaluate the features by performing 10-fold cross-validation. The results are summarized in Table 5.9. The rows describe the subset of observations considered. “All” includes all observations, “No Modals” excludes the modal auxiliaries, “Determiners” includes only the determiners, and “Imp. Determiners” includes only implicit determiners. The columns describe the features used. MAJORITY is the majority baseline, i.e., the accuracy obtained by predicting the most
Table 5.9: De Re vs De Dicto classification. Average accuracies over 10-fold cross-validation. The rows describe the subset of observations considered, and the columns describe the subset of features used.

frequent class or the majority class. The majority class is *de dicto* when all operators are considered (the first row), and *de re* in all other rows. The Type column gives the accuracy when only the type and subtypes are used as features. This column does not apply to implicit determiners, as the subtype information is unavailable. And, finally, the All column gives the accuracy when all features are used.

From Table 5.9, we can conclude that the Type feature is useful in making the *de re-de dicto* distinction, and further gains are obtained by using All features. The most dramatic improvement is for Determiners, and indeed, our features were designed for this case. However, the performance gains are not very high for implicit determiners, and further investigation is needed.

5.6.3 Ordering Sibling and Embedded Operators

In this section, we consider the problem of learning the probabilities for computing ASTs, as described in Section 5.5. We need probabilities for two purposes – (a) pairwise ordering of siblings, and (b) determining whether an embedded operator scopes over its embedding operator. We consider these problems in turn.

**Ordering Siblings:** The observations and labels are defined as follows:

1. Observations $x = (o^i, o^j, \tau)$ are such that there is a preterminal $p \in \tau$, $\{o^i, o^j\} \subseteq$
\( p \), and \( i < j \). In other words, we are considering operators that are siblings. The condition that \( i < j \) implies that \( o^i \) occurs to the left of \( o^j \) in the sentence, and ensures that we have exactly one observation for every pair of operators.

2. Labels \( y = l(x, \alpha) = 1 \) iff \( o^i \gg o^j \) in \( \alpha \). Otherwise, \( y = 0 \).

Observe that this is a generalization of the *de re-de dicto* problem, where one of the operators was restrict to be Post. For each operator, we use the features that we described in the previous section. In addition, we use all pairwise conjunctions of the features. For example, the conjunction of the Pos feature would track where both noun phrases have plural head nouns.

<table>
<thead>
<tr>
<th></th>
<th>Count</th>
<th>MAJORITY</th>
<th>TYPE</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>(All, All)</td>
<td>2793</td>
<td>76.1%</td>
<td>83.3%</td>
<td>87.5%</td>
</tr>
<tr>
<td>(Det, Det)</td>
<td>159</td>
<td>89.9%</td>
<td>92.4%</td>
<td></td>
</tr>
<tr>
<td>(Det, Modal)</td>
<td>332</td>
<td>70.5%</td>
<td>72.6%</td>
<td>77.7%</td>
</tr>
</tbody>
</table>

Table 5.10: Pairwise ordering of sibling operators. Average accuracies over 10-fold cross-validation. The rows describe the subset of observations considered, and the columns describe the subset of features used.

Table 5.10 gives the average accuracies for ordering siblings, using 10-fold cross-validation. As before, rows correspond to subsets of observations. (All, All) includes all pairs of operators, (Det, Det) includes only pairs of determiners, and (Det, Modal) considers pairs where one operator is a determiner and the other is a modal auxiliary. The columns are interpreted identically to Table 5.9. MAJORITY is the majority baseline, which corresponds to a prediction of \( o^i \gg o^j \) for the observation \( x = (o^i, o^j, \tau) \). In other words, the outscopes relation has a tendency to respect the surface order of the sentence. The TYPE column gives the accuracy using only the type feature for each operator and their conjunction. The ALL column gives the accuracy using all features.
The use of **ALL** features provide an improvement over the **TYPE** feature, which in turn improves over the **MAJORITY** baseline. While we obtain significant improvements for all pairs of operators, the gains are less for pairs of determiners. The **MAJORITY** baseline for determiners is quite high (89.9%), making it harder to detect the non-majority cases. The accuracy for determiner-modal pairs is, as expected, comparable to the accuracy obtained for the *de re-de dicto* classification for determiners.

**Embedded Operators:** The observations and labels are defined as follows:

1. Observations \( x = (o, o', \tau) \) are such that \( o' \) occurs in the restrictor of \( o \) in \( \tau \). In other words, \( o' \) is syntactically embedded in (the restrictor of) \( o \).

2. Labels \( y = l(x, \alpha) = 1 \) iff \( R(o) \gg o' \) in \( \alpha \). Otherwise \( (o' \gg o) \), \( y = 0 \).

We refer the reader to Section 5.5.2 for an example. The features used in this task are the same as those used to order siblings. In addition, we use the word appearing immediately prior to the operator. For example, in the noun phrase “Samples of **any** lot”, the word “of” would be the previous word for the operator “**any**”. The idea is that such function words may lead to different biases in the scope of the embedded operator.

<table>
<thead>
<tr>
<th></th>
<th>Count</th>
<th>MAJORITY</th>
<th>TYPE</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>(All, All)</td>
<td>5081</td>
<td>95%</td>
<td>95.3%</td>
<td>96.4%</td>
</tr>
<tr>
<td>(Det, Det)</td>
<td>867</td>
<td>90.3%</td>
<td>91.5%</td>
<td>92.8%</td>
</tr>
<tr>
<td>(VP, Det)</td>
<td>108</td>
<td>70.4%</td>
<td>68.4%</td>
<td>87%</td>
</tr>
</tbody>
</table>

Table 5.11: Scope of embedded operators w.r.t. embedding operators. Average accuracies over 10-fold cross-validation. The rows describe the subset of observations considered, and the columns describe the subset of features used.

Table 5.11 gives the average accuracies over 10-fold cross-validation. The rows are understood as follows. (All, All) includes all pairs, (Det, Det) includes only pairs of
determiners, and (VP, Det) considers pairs where the embedding operator is a verb phrase and the embedded operator is a determiner. We note that, strictly speaking, verb phrases are not really operators. But, it is convenient to treat them as such for the purposes of the experiments, as syntactic nodes serve as islands for quantifier scope.

Using ALL features provides an improvement over the MAJORITY baseline. For the (VP, Det) pairs, we see a dramatic improvement, partly because the MAJORITY baseline is lower. In general, the overwhelming majority of cases respect syntactic attachment, and this may be a function of the guideline to preserve syntactic scope by default. We hope to undertake a more careful investigation of this issue in future work.

5.7 Search

We now consider the search problem, i.e., computing the AST given the estimated probabilities. Section 5.7.1 describes the algorithms used. In Section 5.7.2, we extend the metrics developed for parsing [21] to evaluate ASTs. We conclude, in Section 5.7.3, by evaluating different algorithms using the metrics developed.

5.7.1 Algorithms

In this section, we describe the algorithms that we use to compute the AST, given the appropriate probabilities. We begin by discussing the intractability of the problem of ranking or ordering operators. Then, we sketch a beam search procedure to compute a ranking. The ranking so computed is not optimal in general, but is optimal for most instances in the corpus (which have certain size restrictions). Finally, we briefly discuss the heuristic used to handle embedded operators.

Intractability: The decision version of the ranking problem is NP-complete. A similar result is established by Cohen et al. [35] in the context of a boosting approach.
to ranking.

**Theorem 5.1.** The following problem is NP-complete:

**Input:** A PPT $\tau$, a preterminal $p \in \tau$, probabilities $P(o^j \gg o^i|\tau)$, and $c \in [0, 1]$

**Output:** Yes, if there is an ordering $r$ such that $P(r(p)|\tau) \geq c$

The proof, given in Appendix C, is by reduction from **ACYCLIC SUBGRAPH** [80].

**Ordering Siblings:** We now describe the beam search procedure that we use to compute a ranking. Let us assume as given a PPT $\tau$ and a preterminal $p \in \tau$ such that $p = (o^1, o^2, o^3)$. In other words, we have a preterminal consisting of three operators. Suppose, in addition, that we have trained a classifier, as described in Section 5.6, to give us the probabilities shown in the following matrix:

$$
M = \begin{pmatrix}
- & .8 & .2 \\
.2 & - & .55 \\
.8 & .45 & -
\end{pmatrix}
$$

The entry $M(i,j)$ corresponds to $P(o^i \gg o^j|\tau)$ for $i \neq j$. As a result, $M(i,j) = 1 - M(j,i)$. The specific probabilities can be informally understood as follows. It is likely that $o^1 \gg o^2$, since $M(1,2) = .8$. It is also likely that $o^3 \gg o^1$, since $M(3,1) = .8$. However, the model is not sure whether $o^2 \gg o^3$, since $M(2,3) = .55$.

As we discussed in Section 5.5.1, the goal is to compute a ranking of the operators. A ranking of $p$ is a sequence $r(p) = (i_1, i_2, i_3)$, where the items are all distinct and between 1 and 3. The idea is that $i_j$ corresponds to the position of $o^j$ in the AST. For example, $(2, 3, 1)$ corresponds to reordering the operators as $(o^3, o^1, o^2)$. And, the probability of a reordering is the product of the probabilities of the pairwise ordering relationships:

$$
P((o^3, o^1, o^2)|\tau) = \prod_{j \in \{1,2\}} P(o^3 \gg o^j|\tau) \times P(o^1 \gg o^2|\tau)
$$

We leave it to the reader to verify that this reordering is indeed the best one. The main observation, for this example, is that even though the model is unsure how to
order $o^2$ and $o^3$ w.r.t. each other, it can use information about their ordering w.r.t. $o^1$ to select one of the two orders.

We now cast reordering as a search problem. A state $s = (U, R)$ is a pair, where $U \subseteq p$ is a set of unranked operators, and $R \subseteq \{1, ..., |p|\} \times \{1, ..., |p|\}$ is a partial ranking such that – (a) for all $(j, k) \in R$, $1 \leq k \leq |p| - |U|$, and (b) $(j, k) \in R$ iff $o^j \in p - U$. For example, a state $s$ with $U = \{o^1, o^2\}$ and $R = \{(3, 1)\}$ is one where $o^1$ and $o^2$ have not been ordered w.r.t. each other, but $o^3$ has been assigned the rank 1, since $(3, 1) \in R$. As a result, $o^3 \gg o^1$ and $o^3 \gg o^2$ w.r.t. $s$. Writing $U$ and $R$ explicitly for example states is verbose, and we will adopt an abbreviated notation for states. For the example state $s$, we will simply write $s = (o^1, o^2, o^3_1)$ – only the operator(s) with subscripts have a rank assigned, and in this example $o^3$ has the rank 1.

Given a state $s = (U, R)$, the state $s' = (U', R')$ is a candidate next state iff $U \supset U'$ and $|U' - U| = 1$. In other words, at a next state, exactly one more operator has been ordered. We use $\text{next}(s)$ to denote the set of next states from $s$. The (unique) initial state $s_0 = (p, \emptyset)$, i.e., all operators are unordered. And, a final state has the form $s_F = (\emptyset, R)$, i.e., all operators are ordered. The set of all final states is denoted by $F$.

We will now define the cost of a state, which is the probability of the pairwise decisions that have been made. Given $s = (U, R)$ and an operator $o^i$ such that $(i, k) \in R$, let $J(i) \subseteq \{1, ..., |p|\}$ be the set such that $j \in J(i)$ iff $o^j \in U$ or there exists $(j, k') \in I$ and $k < k'$. Then, the cost of a state:

$$g(s) = 1 \times \prod_{o^j \in p - U} \prod_{j \in J(i)} P(o^i \gg o^j | \tau)$$

For a final state $s_F \in F$, $g(s_F)$ is the probability of the corresponding ranking. Given this set-up, we can now use breadth-first search to enumerate all rankings:

1. Let $i = 0$ and $S$ be a set of states (called the beam), initialized to $s_0$, i.e., $S = \{s_0\}$.
2. Repeat while $i \leq |p|$

   (a) Let $S'$ be the set such that $s' \in S'$ iff $s' \in \text{next}(s)$ for some $s \in S$

   (b) Let $S = S'$ and $i = i + 1$

3. Return $s^*$ such that $s^* \in S$ and for all $s \in S$, $g(s^*) \geq g(s)$

It is easy to see that the algorithm terminates, and in theory, it computes the optimal reordering. However, in the worst case, we may have to examine $O(|p|!)$ states, i.e., the factorial of the number of operators. To address this issue, we adopt the standard technique of placing a bound on the beam width. When we compute $S'$, we sort the vertices in descending order of cost ($g$), and keep at most the best $n$. We used $n = 10^4$ in our experiments. In most cases, the number of operators per preterminal is less than 6, and a size of $10^4$ is sufficient to find the optimal solution (since $10^4 > 6!$).

Figure 5.16: Portion of the state space encountered during beam search.

Figure 5.16 shows a portion of the state space encountered while ordering the operators in our example preterminal. We remind the reader that superscripts on the operators correspond to their surface order, and subscripts correspond to their scope order or rank. The root node is the initial state $s_0$. Since there are three operators,
it has three successors (|next(s₀)| = 3). The first child, s₀₁ = (o₁, o₂, o₃), places o₁ in the first position (indicated by the subscript), and the cost is \(P(o₁ \gg o₂|τ) \times P(o₁ \gg o₂|τ)\), given by the matrix \(M\) above. Similarly, the last child, s₃ = (o₁, o₂, o₃) places o₁ in the first position (again indicated by the subscript). The algorithm then computes the next states for s₀₁, s₀₂, and s₀₃. We show only the children form s₀₃ in Figure 5.16. Since there are only two unordered operators remaining, s₀₃ has only two children. s₀₃₁ = (o₂, o₂, o₁) ranks o₁ second. The search proceeds, and at the end, we obtain the optimal ranking s₀₃₁₂ = (o₂, o₂, o₁).

**Embedded Operators:** To handle embedded operators, we use a simple greedy heuristic. We enumerate the operators in the initial PPT \(τ\) corresponding to an inorder traversal of the PPT. For each operator, we attach it to the most likely ancestor, given the attachment decisions for the previous nodes. This heuristic is optimal for the case where the depth of embedding is at most 1, which is the common case.

### 5.7.2 Metrics

In this section, we adapt the metrics that have been developed for parsing [21] to evaluate the computation of ASTs. Let \(τ\) be the initial PPT, \(α\) the correct AST, and \(α^*\) the computed AST. We define accuracy at various levels.

The simplest metric is to define accuracy at the level of ASTs, i.e., by computing the fraction of cases for which \(α = α^*\). However, this metric is harsh, in the sense that it does not give algorithms partial credit for getting a portion of the AST correct.

The next possible metric is to define accuracy at the level of preterminals. Let \(p\) be a preterminal. Note that \(τ, α\) and \(α^*\) share the same set of preterminals, but may associate different operators with them. We say that \(p\) is correct in \(α^*\), if it is associated with the same set of operators as in \(α\), and for all \(\{o, o'\} \subseteq p\), we have \(o \gg o'\) w.r.t. \(α^*\) iff \(o \gg o'\) w.r.t. \(α\). In other words, the preterminals are identical, both in terms of the set of operators and the ordering between pairs of operators. While preterminal-level accuracy gives partial credit, it is still a little harsh, in the sense
that an algorithm which makes one ordering mistake at a preterminal is penalized
the same as an algorithm which makes multiple mistakes.

Finally, we consider metrics to define accuracy at the level of pairs of operators,
starting with some notation. Let $p$ be a preterminal. The set $\text{Pairs}(p, \alpha)$ consists
of pairs of operators $(o, o')$ such that $o$ and $o'$ are both associated with $p$ in $\alpha$, and
$o = o'$ or $o \gg o'$. The set $\text{Pairs}(p, \alpha^*)$ is defined similarly using $\alpha^*$ instead of $\alpha$. Note
that we consider pairs $(o, o')$ in which the two elements are identical ($o = o'$) or in
which the first outscopes the second ($o \gg o'$). The reason for including pairs with
identical elements is to extend the evaluation metrics to cases where a preterminal is
associated with just a single operator. Given the sets $\text{Pairs}(p, \alpha)$ and $\text{Pairs}(p, \alpha^*)$, we
can define precision ($p$), recall ($r$), and f-score ($f$) in the usual way:

$$p = \frac{|\text{Pairs}(p, \alpha) \cap \text{Pairs}(p, \alpha^*)|}{|\text{Pairs}(p, \alpha^*)|}, \quad r = \frac{|\text{Pairs}(p, \alpha) \cap \text{Pairs}(p, \alpha^*)|}{|\text{Pairs}(p, \alpha)|}, \quad f = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

Informally, precision is the fraction of the computed ordering relationships that are
correct, recall is the fraction of correct ordering relationships that have been com-
puted, and the f-score is the harmonic mean of precision and recall.

### 5.7.3 Results

We evaluate the following algorithms:

1. **No Embedding** – The AST is computed purely by reordering operators within
   a preterminal in the PPT.
   
   (a) **Surface** – No reordering is performed, i.e., the order of operators in the
       AST respects the surface order
   
   (b) **Type** – Using only type and subtype information for the operators

   (c) **All** – Using all the features described in Section 5.6.3

2. **All + Embed** – The initial PPT is transformed into a second PPT before
   reordering (as described in Section 5.5.2). All features are used in reordering.
Table 5.12: Performance of the algorithms in computing the ASTs. Averaged over 10-fold cross-validation. 195 ASTs in total, an average of 8.6 preterminals per AST, and 1.8 operators per preterminal.

Table 5.12 summarizes the performance of the algorithms, under the various metrics. The accuracies are averaged over 10-fold cross-validation. A total of 195 ASTs are used. The average number of preterminals per AST is 8.6, with an average of 1.8 operators per preterminal. The best number under each metric is shown in bold-face. By adding features, we improve the precision from 86.9% to 90.4% to 92% in moving from Surface to Type to All. By handling embedded operators, we improve the recall from 87.6% to 89.4% in moving from All to All + Embed. As we saw in Section 5.6.3, in 95% of the cases, the embedded operators respects syntactic scope, and as a result, we obtain only modest gains from handling embedded operators.

Table 5.13: Performance of the algorithms in computing the ASTs. We restrict attention to preterminals with at least two operators. The measurements are averaged over 10-fold cross-validation. 195 ASTs in total, an average of 2.6 preterminals per AST, and 3.5 operators per preterminal.
One curious result in Table 5.12 is that the SURFACE order baseline gets 81% of the preterminals correct, but only 4.2% of the ASTs. The number of correct preterminals is inflated by preterminals with only one operator in the initial PPT. A better indication can be obtained by restricting attention to preterminals with at least two operators. And, for this case, the performance of the algorithms under the various metrics is shown in Table 5.13.

We conclude this section by comparing our results to accuracies obtained in dependency parsing. McDonald [107] reports a dependency accuracy of about 80% in parsing biomedical texts, with 200 sentences of training data. We obtain an F-score of 90.6% with training data of similar size. The disparity is due to two reasons. First, the number of ordering decisions per AST (approximately 8) is significantly less than the number of parent-child decisions per dependency tree (approximately 30). The lower number of decisions leads to less cascading of errors in ordering. Second, the F-score on the set of pairwise ordering decisions is inflated by the inclusion of reflexive pairs, of the form \((o, o)\). Such reflexive pairs need to be included to handle embedded operators appropriately. But, since 95% of the cases respect syntactic scope (Section 5.6.3), the F-score is high even for the algorithms which do not handle embedded operators. Thus, it is better to consider the relative improvement in F-score over the SURFACE order baseline. We believe that the most indicative metric, in the absolute sense, is the preterminal accuracy when restricted to those preterminals with two or more operators (Table 5.13).

### 5.8 Agreement

The corpus of ASTs has been annotated by a single annotator (the author of this work). It is desirable to have multiple ASTs for a sentence, annotated by different humans, for two reasons – (a) to evaluate the guidelines, and (b) to provide greater assurances of the quality of annotations. Guidelines can be evaluated by treating one human annotation as “gold standard” and another human annotation as “computed”,

177
and comparing the ASTs using the metrics described in Section 5.7.2. Disagreements between the human annotators would be resolved during an “adjudication” phase, providing greater quality assurance. In this section, we discuss agreement on a small scale, and identify ways to improve the guidelines. We hope to perform a larger scale study of agreement, to give better quality assurance, in future work.

The text used for the agreement evaluation is CFR 610.40. The reason for choosing this particular text is that the sentences are long and complex. A total of 32 sentences were used, with an average of 39.8 tokens per sentence (36.3 tokens excluding punctuation). We considered two agreement experiments.

The first experiment that we consider is self-agreement, which measures the agreement of the author of this work with himself at different points in time! It provides an approximate upper bound on what one can hope to achieve with inter-annotator agreement, i.e., if the intent of the author was well-coded in the guidelines. The first annotation of CFR 610.40, used as “gold standard”, was done in April 2010, and the second annotation, used as “computed”, was done in August 2010.

In the second experiment, we consider inter-annotator agreement, which measures the agreements between different annotators. We used one (other) annotator with a background in linguistic syntax and semantics. A one hour meeting was used to discuss guidelines, demonstrate the tool, and annotate two practice sentences. A further 15 hours was needed by the annotator to annotate the 32 ASTs. While the annotator has a general background in logic, we did not provide too many of the specifics of our formalism. Rather, we only showed sketches of the formalization, as we did earlier in this chapter. We wished to evaluate whether it was necessary to understand all the logical details, in order to annotate the ASTs. During the annotation phase, the only communication between the author and the annotator was about usage of the tool, and no other “advice” was given.

For each experiment, we evaluate the following notions of agreement:

(a) Agreement on the AST given just the sentence,
Table 5.14: Agreement evaluation.

(b) Agreement on the processed parse tree (PPT) given the sentence, and
(c) Agreement on the AST given the PPT.

Note that the experiments that we have described in the previous sections correspond to Item (c). Item (a), which involves computing the AST from the sentence, is a harder problem, as the PPT also needs to be computed. Item (b), which involves just computing the PPT, can be viewed as a task analogous to parsing. The metrics described in Section 5.7.2 are sufficient to compare (agreement between) ASTs. A PPT can be viewed as a set of constituents, where each constituent is an operator and its restrictors. We then have analogous precision and recall measures of ASTs.

Table 5.14 summarizes the results of evaluating agreement. The columns of the table are divided into two groups, corresponding to evaluating ASTs and PPTs. Under the AST group of columns, \( p \), \( r \), and \( f \) correspond to the precision, recall, and F-score respectively of pairwise ordering decisions (as discussed in Section 5.7.2). However, the two ASTs may not agree on the set of operators, and as a result, if there is a mismatch between the PPTs, the errors cascade into the ASTs. For example, an operator in the “gold standard” PPT which is not found in the “computed” PPT would result in false negatives for all pairwise ordering relationships that it is involved in.
The columns $p$ and $\alpha$ give the preterminal and AST level accuracies. Under the PPT group of columns, $p$, $r$, and $f$ correspond to the precision, recall, and F-score respectively of the set of constituents. We remind the reader that a constituent is an operator together with its restrictors. And, the column $\tau$ gives the accuracy at the level of PPTs.

We now discuss the results in Table 5.14, given by the rows. The rows are divided into two groups, those prefixed with “S.” for self-agreement, and those prefixed with “I.” for inter-annotator agreement. The suffixes are understood as follows.

The row suffix All gives the agreement on the AST given the sentence (the first group of columns), and the agreement on the PPT given the sentence (the second group of columns). In the self-agreement experiment, an F-score of 86.6% is obtained for AST agreement, and an F-score of 94.9% for PPT agreement. This corresponds to an accuracy of 34.4% at the level of ASTs and 40.6% at the level of PPTs. Similarly, in the inter-annotator agreement experiment, an F-score of 56.8% is obtained for ASTs, and 82.1% for PPTs. And, here, no ASTs were entirely correct.

To explain the low agreement in the rows suffixed by All, we perform two further evaluations. The row suffix Root is the same type of evaluation as All, except that we restrict attention to operators that take scope at the root preterminal of the PPT. We now obtain F-scores and AST-level accuracies of 91.3% and 68.8% for self-agreement (81.2% and 31.3% for inter-annotator agreement). Analogous improvements are obtained in the PPT accuracies. Thus, we may conclude that many of the problems in the first row are caused by embedded operators.

There are two sorts of errors/disagreements that can occur with (embedded) operators – syntactic disagreements, which reflect in both the PPT and the AST, and scope disagreements, which reflect only in the AST. To get an estimate of the scope (dis)agreement, we corrected the PPTs annotated in the second phase to match the “gold standard”. The correction was done by viewing only the PPT in the “gold stan-

---

7The “gold standard” and “computed” ASTs may not have the same set of preterminals. The total number of preterminals is take to be the union of those found in the two ASTs.
standard” and correcting the “computed” PPT to match it. The results of comparing the corrected ASTs to the “gold standard” are shown in the rows suffixed All*. We now obtain an F-score and AST-level accuracies of 98% and 75% for self-agreement (94.3% and 56.3% for inter-annotator agreement). We conclude that the low AST agreement in the rows suffixed by All are mainly due to syntactic errors.

We now discuss the sorts of syntactic corrections that we needed to make. In the self-agreement experiment, there were two main types of corrections – (a) typos, which were unintentional omissions or mistakes, and (b) real disagreements, which arise mostly in the usual cases of coordination and prepositional/adverbial phrase attachment. In the inter-annotator case, in addition to the aforementioned types of corrections, the main source of disagreement was in the method for annotating relative clauses and verb phrase modifiers. We conclude that the syntactic annotation and/or operator identification is an important deficiency our guidelines. The long sentences in CFR 610.40 make parsing (computing the PPT) a non-trivial task!

The inter-annotator agreement in producing the AST from the PPT (the row I.All*) is lower than the self-agreement (the row S.All*). One repeated case of disagreement was phrases of the form “the communicable disease agents”, which occur at various places in the CFR. These phrases were given de re scope in the “gold standard”, but were given de dicto scope by the annotator. Many of the AST-level disagreements were simply due to this single scope disagreement. The de re scope assigned in the “gold standard” and self-agreement was influenced in part by the author’s knowledge of the workings of the logic. However, the annotator was given the instruction to preserve surface scope by default, and there did not seem to be particularly good reasons to violate it. One of the more difficult aspects of developing guidelines for ASTs is to clarify which occurrences of the determiner “the” are to be treated universally. We hope to improve this aspect of the guidelines in future work.
5.9 Related Work

In this section, we place our work in the context of related work in NLP. Section 5.9.1 relates the annotation of ASTs to other works that translate natural language to logic. In Section 5.9.2, we discuss some alternatives to ASTs, to clarify some methodological decisions. We then discuss work related to the computation of ASTs (Section 5.9.3).

5.9.1 Annotating ASTs

The analysis of scope can be seen as a middle ground between two lines of research in translating sentences to logic.

At one end of the spectrum, we have methods that achieve good accuracy on restricted texts. The two main corpora that have been considered are the GeoQuery corpus [142] and the ATIS-3 corpus [41]. The GeoQuery corpus consists of queries to a geographical database. The queries were collected from students participating in a study and the average sentence length is 8 words. The ATIS corpus is collected from subjects’ interaction with a database of flight information, using spoken natural language. The utterances have been transcribed, and the average sentence length is 10 words [19]. Algorithms, which achieve good accuracy, have been developed to compute the logical translation for these queries [149-151]. The annotated sentences in the FDA CFR Section 610.40 are longer (about 30 words on average), and contain modalities which are not present in these corpora.

At the other end of the spectrum, Bos et al. [24] have developed a broad-coverage parser to translate sentences to a logic based on discourse representation theory. The applications of broad-coverage parsers include tasks like textual entailment, where errors in the logical translation are expected. In addition, there is no method to evaluate the correctness of the translation produced by such broad-coverage parsers.

To summarize, there are NLP techniques that either produce an accurate translation for sentences in a limited domain, or produce some translation for sentences in many domains for applications where errors in the logical translation are expected.
ASTs offer a middle ground between these two approaches. We set aside the problem of translating every phrase into logic, and focus instead on the position of the phrases in the logical translation. This approach lets us make progress on the procedure for translating sentences to logic, while leaving open the investigation of (parts of) the logic itself.

5.9.2 Alternatives to ASTs

As we discussed previously, ASTs are based on May’s idea of logical form [103]. Logical form is one of several different approaches to account for scope phenomena. In this section, we discuss some alternatives to logical form, in order to clarify methodological decisions.

Logical form can be understood as a denotational approach to scope of operators. It is an explicit intermediate representation that encodes the resolution of scope ambiguities. In earlier approaches, such as the Montague grammar [112] and Cooper’s quantifier storage mechanism [38], the resolution of scope is handled operationally, i.e., via the rules for composing the logical translation. The denotational character of logical form makes it a convenient starting point for annotating scope.

Kamp and Reyle’s discourse representation theory [78] (DRT) is designed to provide an account of both sentential scope and some discourse-level anaphora. Our annotation of logical form has the more modest goal of providing an analysis of scope, and in the long term, we hope to extend it with DRT-like mechanisms for handling discourse-level anaphora.

Underspecified representations of scope, such as [39, 68, 128], have been developed to compactly encode all possible interpretations of a sentence. In this work, we have focussed on choosing one interpretation of a sentence, given the context. In cases where there were multiple interpretations, we adopted default rules. For example, in the phrase “the completion of all tests”, we default to the surface scope, which corresponds to the interpretation that there is a single completion for all tests. However,
the inversely linked interpretation is also possible, i.e., where we consider separate completion points for each test. We have not encountered an example where such differing scopes would result in different answers to conformance. Nevertheless, it may be better practice to use an underspecified representation rather than default scope orderings, and we hope to develop such an annotation scheme in future work.

5.9.3 Computing ASTs

Various types of features have been proposed for the scopal ordering of determiners. Examples include syntactic features [71, 124], such as position (subject or object) and voice, semantic features [61, 72], such as thematic roles, and pragmatic features [91, 114, 132, 140], such as class size (discussed below). In a psycholinguistic experiment, Kurtzman and MacDonald [87] evaluate syntactic and semantic features by having humans indicate their preferred reading on sentences such as “every kid climbed a tree”. They found that no single type of feature was sufficient. In a machine learning approach, Srinivasan and Yates [140] used pragmatic information about class size to predict preferred readings. For example, given the sentence “some person came from every city” and the fact that there are more people than cities, we may predict that every $\gg$ some is the preferred reading. We have experimented with lexico-syntactic features in this work, and leave an investigation of semantic and pragmatic features to future work.

The problem of ranking or ordering items has been investigated in a variety of applications. Direct applications include document retrieval [35], collaborative filtering [40, 66], and sentiment analysis [120]. Ranking methods have also been used in NLP in the context of re-ranking for parsing [36, 137], named entity extraction [37], and machine translation [138]. The idea with re-ranking is to reorder the output of an underlying classifier using global features.

Liu [96] divides ranking algorithms into three types – (a) pointwise approaches, which assign a score to each item and then sort the items based on the score, (b)
pairwise approaches, which assign orders to pairs of items, and (c) listwise approaches, which optimize a loss over the entire list. Our approach falls into the class of pairwise approaches, and can be seen as a probabilistic version of the approach in [35].

5.10 Conclusions

We have proposed *abstract syntax trees* (ASTs) as an intermediate step in translating regulatory sentences to logic. In Section 5.3, we discussed guidelines for annotating ASTs. We annotated a corpus of 195 sentences from Section 610 of the FDA CFR (Section 5.4). In Sections 5.5-5.7, we developed and tested algorithms to convert a processed parse tree (PPT) to an AST. The main step in this conversion was to rank or order the operators at a preterminal. We presented a probabilistic model for ranking, investigated the design of features, and developed search heuristics. The best algorithm, which uses all features and handles embedded operators, achieves an F-score of 90.6%. Finally, in Section 5.8, we discussed an agreement evaluation, which let us identify areas in which the guidelines need to be improved.

The experimental results suggest that with improved guidelines and a larger scale annotation, we can achieve reasonable accuracy in converting a processed parse tree to an AST. We conclude that ASTs (and more generally the computation of logical form) may be a feasible next step in moving beyond parsing. There are several avenues for further inquiry, which we discuss in the following chapter.
Chapter 6

Conclusions

In this thesis, we considered the problem of regulatory conformance checking. We started, in Chapter 1, by defining conformance as the satisfaction of obligations. Conformance checking was cast as a problem of runtime verification. Regulations are translated to logic, and organizations are represented as a trace or run. The runtime checker outputs an affirmative answer to conformance, or a counterexample if a violation is detected.

Given this approach, we narrowed our focus to the translation of regulation to logic. We argued for a sentential translation of regulation to logic, in order to help in tracing violations, and also to ease the difficulty in translating regulation to logic (Chapter 2). We studied subproblems in the sentential translation from two angles, toward the long term goal of developing tools to assist in such a translation:

(a) The design of logics (Chapters 2-4), and
(b) The annotation and computation of logical form (Chapter 5)

We now summarize our contributions and discuss avenues for future work. The logic aspects are discussed in Section 6.1 and the logical form in Section 6.2.
6.1 Logic

We have designed logics to accommodate two features of regulatory texts – (1) references between laws, and (2) the concepts of obligation and permission. An underlying theme was the importance of the notion of saying in such constructs. In Chapter 2, we explored a predicative analysis of says, to provide a unified analysis of various kinds of inter-sentential references, e.g., priorities of exceptions over rules, and references to definitions or list items. We then extended the analysis, in Chapter 4, to treat says as a modal. We provided a new decidable axiomatization of representation in access control and recursive notions of legal power. A non-interference property was used to demonstrate that the logic preserves the rights of principals. Conformance checking, in the presence of nested obligations and permissions, was shown to be decidable. We also identified a polytime decidable fragment of the logic that accommodates a variety of access control examples. There are several avenues for future work.

6.1.1 Other Kinds of Intersentential References

As we discussed in Section 3.4.2, there is a frequent class of intersentential references for which there is no adequate formulation in REFL. Consider again the following statement:

(21) The general safety test shall be performed as specified in this section...

At this time, we do not have a good understanding of how to formalize the phrase “as specified in this section” in (21). One can avoid formalizing such requirements by introducing additional inter-sentential references in other sentences (see Section 3.4.2 Example 4 for a discussion). A question of interest is whether the formalization of such requirements would let us avoid these additional references.

6.1.2 Excessively Personal Obligations

Consider again the following sentence and its formal representation (from Chapter 5):

187
A general safety test for the detection of extraneous toxic contaminants shall be performed on biological products intended for administration to humans.

Sentence (30) appears in passive voice. Yet, we have translated it to logic as an obligation on the manufacturer $O_{m(x)}$. This is an artifact of our treatment of obligations in Chapter 4, where there is a principal who is responsible for each obligation. It is possible to interpret (30) as an obligation, where there is some flexibility in assigning blame. Such an interpretation would interact with delegation. For example, if a manufacturer hands the product to an appropriate lab for testing, then the lab can be held responsible for failure to perform the test in an appropriate way. We speculate that a formalization this form of obligation needs notions of bringing about or seeing to it that (e.g., [17, 70]). For example, the manufacturer may see to it that a product is tested by requesting an appropriate lab to do the test. And, the lab assumes the subsequent obligations. As we discussed in Section 4.4.2, notions of seeing to it that are needed for other phenomena as well, and we hope to undertake an investigation in the future.

6.1.3 Permission to Choose a Value

In Chapter 5, we explored a notion of permission to speak, using which a principal can grant the power of representation to another. While this is adequate for access control rights, there are related notions which we have not considered:

(46) Parents have the right to name their children

Informally, (46) gives parents the right to choose a name for the child. And, a choice made by the parents (during an appropriate ceremony) would lead to the name being legally recognized. We simplify the problem to illustrate the key issue. Suppose the following statement appears in a principal $A$’s policy:

(47) $B$ has the right to set the value of $p$
Let us assume further that \( p \) is a boolean variable. There are two choices to represent the right in (47) – corresponding to a conjunctive or disjunctive interpretation:

\[(48)\quad B \text{ has the right to set the value of } p \text{ to true or false}\]

\[(49)\quad B \text{ has the right to set the value of } p \text{ to true and the right to set the value of } p \text{ to false}\]

There are a few ways to formalize (48) and (49) in the logic that we have developed. However, the translations of (48) turn out to be vacuous, i.e., no right gets conveyed to \( B \). And, the translations of (49) turn out to be excessive, i.e., \( B \) can introduce any statement on behalf of \( A \). The problems are reminiscent of those that arise with free choice permissions [77]. Substructural logics, such as linear logic [55], can typically handle such notions of choice, and it is of interest to explore the interaction of the representation axiom with the rich notions of disjunction and conjunction provided by such logics.

### 6.1.4 Contracts

We have focused on examples from privacy regulation. A variety of speech acts arise in the context of business contracts and transactions [81–84]. *Promises* can be understood as a self-imposed obligation, e.g., \( \text{says}_{(A)} \ O_A \varphi \) can be read as “\( A \) promises \( \varphi \)”. Such promises can be enforced by a higher authority by issuing a statement of the form \( \text{says}_{(B)} \ P_A \text{ says}_{(A)} \ O_A \varphi \), i.e., “\( B \) empowers \( A \) to promise \( \varphi \) on her own behalf”. If \( A \) promises \( \varphi \), then we will conclude using \( \text{A5} \) that \( \text{says}_{(B)} \ O_A \varphi \), i.e., “\( B \) requires \( \varphi \) of \( A \)”. Thus, if \( A \) fails to carry out her promise, she will not conform to the laws issued by \( B \). In reasoning about contracts, it is useful to relate events, such as the delivery of goods, to speech acts, such as the promise of delivery (cf. [82]). Such relationships between events and speech acts may lead to other interactions between *saying* and *obligation*. We hope to study contractual examples in more detail in future work.
6.2 Logical Form

In Chapter 5, we described a corpus of regulatory sentences annotated with a variant of logical form called abstract syntax trees (ASTs). To our knowledge, this is the first attempt to annotate structures of this kind. We adapted learning algorithms to compute these ASTs, and obtained an F-score of 90.6% on a PARSEVAL-like evaluation metric. We also evaluated agreement between human annotators. The results suggest that converting a parse tree to an AST is a feasible task at the current state-of-the-art. We conclude by discussing avenues for further research.

6.2.1 Annotation and Computation

There are several avenues for further inquiry in the annotation and computation of ASTs. It is of interest to build a larger scale corpus, to get a better understanding of the effort involved, in terms of annotator effort and cost. Several aspects of the guidelines need to be improved, e.g., the interaction of de-re determiners in subject position with coordination, and the handling of comparatives. We also hope to study different regulatory corpora in future work.

In the context of computing ASTs, there are potential gains to be had from better feature design. Due to the small size of the corpus, we have not used word or phrase level features. We plan to investigate semi-supervised techniques to obtain such information, e.g., by using clusters [122] and appropriate low-dimensional representations [22] obtained from unlabeled data. Identifying pragmatic features, such as information about class size [140], is another important direction for research.

6.2.2 Connecting ASTs and Logic

In Section 3.4.2, we showed that the annotation of ASTs could provide some quantitative evaluation of the coverage of the logic. In particular, the distribution of clause and verb phrase modifiers revealed constructs for which we do not have an adequate...
formalization. An open question is whether ASTs prove useful in discovering other examples of constructs which need to be handled by the logic.

A long term goal of our work is to be able to go from the AST to logic. In the short term, we may be able to extract some approximation to logical translation of a sentence, by specifying some simple rules to convert AST operators to their logical counterparts. Such an approximate translation may contain errors, but there may be information extraction tasks for which it is useful.
Appendix A

REFL: Proofs

Section A.1 proves the existence of fixed points. The complexity of conformance checking is discussed in Section A.2.

A.1 Fixed Point

In this section, we build up to a proof of Theorem 2.1. We assume the notational conventions developed in Chapter 2.

Proposition A.1. Given a poset \((S_R^I, \leq)\), the function \(I_R^\Upsilon\) is:

1. Inflationary - For all \((u_1, u'_1) \in S_R^I\), \((u_1, u'_1) \leq I_R^\Upsilon(u_1, u'_1)\)

2. Well-defined - For all \((u_1, u'_1) \in S_R^I\), \(I_R^\Upsilon(u_1, u'_1) \in S_R^I\)

3. Monotonic - For all \(\{(u_1, u'_1), (u_2, u'_2)\} \in S_R^I\), if \((u_1, u'_1) \leq (u_2, u'_2)\), then \(I_R^\Upsilon(u_1, u'_1) \leq I_R^\Upsilon(u_2, u'_2)\)

Proof. Item 1: Let \((u_2, u'_2) = I_R^\Upsilon(u_1, u'_1)\). We are given that \((u_1, u'_1)\) is sound w.r.t. \(R\). Hence, for all \(i \in N\) and \((id, \phi) \in U\):

- If \((id, \phi) \in u_1(i)\), then by soundness, there exists \((id).x : \varphi \mapsto \psi \in \text{Reg}\) and \(v \in V\) such that \(v(\psi) = \phi\) and \(\eta(u, v)(\varphi, R, i, v) = \top\). Therefore, by the definition of \(I_R^\Upsilon\), \((id, \phi) \in u_2(i)\). We can conclude that \(u_1(i) \subseteq u_2(i)\).
• If \((id, \phi) \notin u_2(i)\), then by soundness, for all \((id,x) : \varphi \mapsto \psi \in \text{Reg}\) and \(v \in V\) such that \(v(\psi) = \phi\), we have \(\eta(u,u')(\varphi,R,i,v) = \bot\). Therefore, by the definition of \(T^R_Y\), \((id, \phi) \notin u_2(i)\). We can conclude that \(u_1(i) \supseteq u_2(i)\).

Hence, by Definition 2.14, \((u_1, u'_1) \leq (u_2, u'_2)\)

**Interlude:** For the second and third items, we need the following observations. Given \(u_1(i) \subseteq u'_1(i) \subseteq U\) and \(u_2(i) \subseteq u'_2(i) \subseteq U\), if \((u_1, u'_1) \leq (u_2, u'_2)\), then for all \(\varphi \in L', i \in N\) and \(v \in V\):

\[
\begin{align*}
\text{(D1)} & \text{ If } \eta(u_1, u'_1)(\varphi, R, i, v) = \top, \text{ then } \eta(u_2, u'_2)(\varphi, R, i, v) = \top \\
\text{(D2)} & \text{ If } \eta(u_1, u'_1)(\varphi, R, i, v) = \bot, \text{ then } \eta(u_2, u'_2)(\varphi, R, i, v) = \bot
\end{align*}
\]

These are established easily by induction over the structure of \(\varphi\). Note that the claims are for all pairs of utterances, and not just the sound ones.

**Item 2:** Let \((u_2, u'_2) = T^R_Y(u_1, u'_1)\). From Item 1, it follows that \((u_1, u'_1) \leq (u_2, u'_2)\). Suppose, for the purpose of contradiction, that \((u_2, u'_2)\) is not sound w.r.t. \(R\). Then, by soundness, there exists \((id, \phi) \in U\) such that:

• \((id, \phi) \in u_2(i)\) and for all \((id,x) : \varphi \mapsto \psi \in \text{Reg}\) and \(v \in V\) such that \(v(\text{says}_{id} \psi) = \phi\), we have \(\eta(u_2, u'_2)(\varphi, R, i, v) \neq \top\). Using (D1), we can conclude that \(\eta(u_1, u'_1)(\varphi, R, i, v) \neq \top\). Therefore, by the definition of \(T^R_Y\), \((id, \phi) \notin u_2(i)\), giving us a contradiction.

• The second case (where \((id, \phi) \notin u'_2\)) is contradicted similarly using (D2).

The proof of Item 3 is along similar lines.

The existence of fixed points is established using Zorn’s lemma, which applies to chain-complete posets. Given the poset \((S^R_Y, \leq)\), a set \(S' \subseteq S_Y\) is called a chain (totally ordered set) if for all \((u_1, u'_1), (u_2, u'_2) \in Y'\), we have \((u_1, u'_1) \leq (u_2, u'_2)\) or \((u_2, u'_2) \leq (u_1, u'_1)\). A poset is chain complete if every chain has a supremum. We now show that \((S^R_Y, \leq)\) is a chain-complete poset:

193
Proposition A.2. \((S^R_T, \leq)\) is a chain-complete poset.

Proof. Given a chain \(S' \subseteq S_T\), consider the pair \((u_s, u'_s)\) defined as follows. For all \(i \in N\):

\[
\begin{align*}
    u_s(i) &= \bigcup_{(u, u') \in S'} u(i) \\
    u'_s(i) &= \bigcap_{(u, u') \in S'} u'(i)
\end{align*}
\]

It is immediate from the construction that \(\forall (u, u') \in S': (u, u') \leq (u_s, u'_s)\). It is also easy to see that if \((u_s, u'_s)\) is sound, then it is the supremum of \(S'\). Thus, it suffices to show that \((u_s, u'_s)\) is sound, and this can be established by an argument similar to the proof of Proposition A.1. \(\square\)

Lemma A.1 (Zorn (c.f. [131])). Every chain complete poset has a maximal element.

The existence of maximal fixed points is immediate from Zorn’s lemma and the fact that \(I_T\) is inflationary, i.e., \((u_1, u'_1) \leq I_T(u_1, u'_1)\). Let \((u_M, u'_M)\) be a maximal element in \(S_T\). Since \((u_M, u'_M)\) is maximal and \((u_M, u'_M) \leq I_T(u_M, u'_M)\) it follows that \((u_M, u'_M) = I_T(u_M, u'_M)\).

To show the existence of a least fixed point, as Kripke [85] notes, we will need to use the observation that \(I_T\) is monotonic (Proposition A.1, Item 3). With monotonicity, we obtain the following corollary to Zorn’s lemma:

Corollary A.1. Given \((u_1, u'_1) \in S^R_T\), let \(\sigma(u_1, u'_1)\) be the smallest set such that:

(a) \((u_1, u'_1) \in \sigma(u_1, u'_1)\),

(b) If \((u, u') \in \sigma(u_1, u'_1)\) then \(I_T(u, u') \in \sigma(u_1, u'_1)\), and

(c) If \(C \subseteq \sigma(u_1, u'_1)\) is a non-empty chain, then \((u_{sc}, u'_{sc}) \in \sigma(u_1, u'_1)\), where \((u_{sc}, u'_{sc})\) is the supremum of \(C\) w.r.t. \(S^R_T\).

Then:

1. \(\sigma(u_1, u'_1)\) is a chain whose supremum is a fixed point of \(I^R_T\)

2. \(\sigma(u_1, u'_1)\) contains a unique fixed point
We have a contradiction to the minimality of \( S \).

We now claim that (Item 2):

(a) \((u_1, u'_1) \leq (u, u')\), then \((u_{s1}, u'_{s1}) \leq (u_{s2}, u'_{s2})\), where \((u_{s1}, u'_{s1})\) and \((u_{s2}, u'_{s2})\) are the suprema of \(\sigma(u_1, u'_1)\) and \(\sigma(u_2, u'_2)\) resply, and

4. \( I^R_T \) has a unique least fixed point.

Proof. Fix \((u_1, u'_1)\) and let \( S' = \sigma(u_1, u'_1) \).

**Item 1:** The fact that \( S' \) is a chain is used to prove Zorn’s lemma, and we refer the reader to [131] for a proof. Let \((u_s, u'_s)\) be the supremum of \( S' \). Since \( S' \) contains its supremum, and \( I^R_T(u_s, u'_s) \in S' \) (by definition), we can conclude that \((u_s, u'_s) = I^R_T(u_s, u'_s)\). Thus, the supremum of \( S' = \sigma(u_1, u'_1) \) is a fixed point of \( I^R_T \).

**Interlude:** For the rest of the items, we will need the following observation:

\((\ast)\) \((u_1, u'_1) \leq (u, u')\), for all \((u, u') \in S'\)

Suppose not. Consider the set \( S'' \subset S' \) such that \((u, u') \in S''\) iff \((u_1, u'_1) \leq (u, u')\). Then:

(a) \((u_1, u'_1) \in S''\),
(b) If \((u, u') \in S''\) then \( I_T(u, u') \in S'' \) (since \((u_1, u'_1) \leq (u, u') \leq I^R_T(u, u')\)), and
(c) If \( C \subseteq S'' \) is a non-empty chain, then \((u_1, u'_1) \leq \sup(C)\), since for all \((u, u') \in C\), we have \((u_1, u'_1) \leq (u, u').\) Hence, \( \sup(C) \in S'' \).

Thus, we obtain a contradiction to the minimality of \( S' \).

**Item 2:** We now claim that \((u_s, u'_s)\) is the unique fixed point in \( S' \). Suppose not. Let \((u, u') \in S' \) be a fixed point. Since \((u, u') \neq (u_s, u'_s)\) and \((u_s, u'_s)\) is the supremum, we have \((u, u') < (u_s, u'_s)\). Consider the set \( S'' \) such that for all \((u_A, u'_A) \in S'\), \((u_A, u'_A) \in S''\) iff \((u_A, u'_A) \leq (u, u')\). Then:

(a) \((u_1, u'_1) \in S''\) (by \((\ast)\))

(b) For all \((u_A, u'_A) \in S''\), we have \( I^R_T(u_A, u'_A) \in S''\) (for if not \((u_A, u'_A) \leq (u, u')\) and \( I^R_T(u_A, u'_A) \not\leq I^R_T(u, u')\), contradicting the monotonicity of \( I^R_T \)), and

(c) The presence of suprema is similarly verified.

We have a contradiction to the minimality of \( S' \). Hence, \((u_s, u'_s)\) is the unique fixed point in \( S' \).
**Item 3:** Given \((u_1, u'_1) \leq (u_2, u'_2)\), let \((u_{s1}, u'_{s1})\) and \((u_{s2}, u'_{s2})\) be the suprema of \(\sigma(u_1, u'_1)\) and \(\sigma(u_2, u'_2)\) resply. We claim that \((u_{s1}, u'_{s1}) \leq (u_{s2}, u'_{s2})\). Suppose not. Consider the set \(S'' \subset \sigma(u_1, u'_1)\) such that \((u_A, u'_A) \in S''\) iff \((u_A, u'_A) \leq (u_{s2}, u'_{s2})\). Then:

(a) \((u_1, u'_1) \in S''\) (by (*))

(b) For all \((u_A, u'_A) \in S''\), we have \(I_R \Upsilon(u_A, u'_A) \in S''\) (for if not \((u_A, u'_A) \leq (u_{s2}, u'_{s2})\) and \(I_R \Upsilon(u_A, u'_A) \not\in I_R(u_{s2}, u'_{s2})\), contradicting the monotonicity of \(I_R \Upsilon\)), and

(c) The presence of suprema is similarly verified.

Again, we have contradicted to the minimality of \(\sigma(u_1, u'_1)\). Hence \((u_{s1}, u'_{s1}) \leq (u_{s2}, u'_{s2})\).

**Item 4:** Let \((u_0, u'_0)\) be the pair such that, where for all \(i \in N\), \(u_0(i) = \emptyset\) and \(u'_0(i) = \mathcal{U}\) (the set of all utterances). It is easy to see that \((u_0, u'_0)\) is sound. Furthermore, for all sound pairs \((u, u') \in S_R\), \((u_0, u'_0) \leq (u, u')\). Hence, the supremum of \(\sigma(u_0, u'_0)\) is the least fixed point. 

\[ \square \]

### A.2 Complexity

We discuss the proofs of the upper and lower bounds of conformance checking w.r.t. the least fixed point:

**Lemma 2.1 (Upper Bound).** Given a finite run \(R\) and regulation \(\text{Reg}\), \(R \models \text{Reg}\) can decided in EXPSPACE (space exponential in the size of \(\text{Reg}\))

**Proof.** (sketch) Corollary A.1 can easily be turned into a decision procedure. We start with the pair \((u_0, u'_0)\) and repeatedly apply \(I_R \Upsilon\) until a fixed point is reached. For all \(i\), \(|u_0(i)| = 0\) and \(|u'_0(i)| = |\text{Reg}| \times |V(X, O)|\). In the worst case, for each application of \(I_R \Upsilon\), there is at most one change. And, \(n \times |\text{Reg}| \times |V(X, O)|\) steps are required to reach a fixed point, where \(|V|\) is the number of variable assignments. Note that \(|V(X, O)| = |O|^k\) where \(O\) is the set of objects and \(k\) is the largest number of distinct variables appearing in a regulatory statement.

196
To apply $I^R_t$ to $(u,u')$, we need to perform validity tests of the form $u(i)_{td} \models \text{says}_{td} \varphi$. The worst-case size of the validity instances is $|\text{Reg}| \times |O|^k$. Since validity for propositional LTL is PSPACE-complete [139], applying $I^R_t$ requires EXPSPACE (due to the $|O|^k$ factor). We note that for the fragment of LTL discussed here (using only $\Box$ and $\Diamond$) validity is NP-complete [139], and for this fragment $R \models \text{Reg}$ can be decided in EXPTIME.

Lemma 2.2 (Lower Bound). Given a finite run $R$ and regulation $\text{Reg}$, $R \models \text{Reg}$ is hard for EXPTIME (time exponential in the size of Reg).

Proof. (sketch) We encode formulas in first-order logic as regulations. Let $\varphi(\vec{x})$ be a first-order formula, where $\vec{x} = x_1,\ldots,x_m$ are free variables. If $\varphi(\vec{x})$ contains no quantifiers, we represent it by a law:

$$(A_\varphi) \varphi(\vec{x}) \mapsto q_\varphi(\vec{x})$$

$q_\varphi(\vec{x})$ is a predicate symbol that doesn’t appear in $\varphi(\vec{x})$. It is easy to see that $(A_\varphi,v(q_\varphi(\vec{x})))$ is available as an utterance iff $\varphi(\vec{x})$ is true w.r.t. $v$.

For quantified statements we proceed inductively. Given $\exists y : \varphi(y,\vec{x})$, we add two laws:

$$(A_{\exists y: \varphi}) \text{ says}_{\{B_{\exists y: \varphi}\}} q'(\vec{x}) \mapsto q_{\exists y: \varphi}(\vec{x})$$

$$(B_{\exists y: \varphi}) \text{ says}_{\{A_\varphi\}} q_\varphi(y,\vec{x}) \mapsto q'(\vec{x})$$

Observe that $\text{says}_{\{B_{\exists y: \varphi}\}} q'(\vec{x})$ is true w.r.t. an assignment $v$ $(B_{\exists y: \varphi},v(q'(\vec{x})))$ is available as an utterance iff $\varphi(\vec{x})$ is true w.r.t. $v$.

For quantified statements we proceed inductively. Given $\exists y : \varphi(y,\vec{x})$, we add two laws:

$$(A_{\exists y: \varphi}) \text{ says}_{\{B_{\exists y: \varphi}\}} q'(\vec{x}) \mapsto q_{\exists y: \varphi}(\vec{x})$$

$$(B_{\exists y: \varphi}) \text{ says}_{\{A_\varphi\}} q_\varphi(y,\vec{x}) \mapsto q'(\vec{x})$$

Observe that $\text{says}_{\{B_{\exists y: \varphi}\}} q'(\vec{x})$ is true w.r.t. an assignment $v$ $(B_{\exists y: \varphi},v(q'(\vec{x})))$ is available as an utterance. And, $(B_{\exists y: \varphi},v(q'(\vec{x})))$ is available as an utterance iff $\varphi(\vec{x})$ is true w.r.t. some variable assignment $v'$ that is identical to $v$ except for $y$. We can then argue inductively that $(A_{\exists y: \varphi},v(q_{\exists y: \varphi}(\vec{x})))$ is available as an utterance iff $\exists y : \varphi(y,\vec{x})$ is true w.r.t. $v$.

Given $\forall y : \varphi(y,\vec{x})$, we use the equivalence $\forall y : \varphi(y,\vec{x}) = \neg \exists y : \neg \varphi(y,\vec{x})$ and proceed as follows:

$$(A_{\forall y: \varphi}) \neg \text{ says}_{\{A_{\exists y: \neg \varphi}\}} q_{\exists y: \neg \varphi}(\vec{x}) \mapsto q_{\forall y: \varphi}(\vec{x})$$

To complete the construction, given $\varphi(\vec{x})$, we add the obligation:

$$(1)_{o} : \neg \text{ says}_{\{A_\varphi\}} q_\varphi(\vec{x}) \mapsto \bot.$$
It can be shown that a run with a single state conforms to the regulation iff \( \varphi \)

is valid at the state. Model-checking for first-order logic is PSPACE-complete (cf. [145]). It follows that computing the least fixed point is PSPACE-hard.

In encoding first-order formulas, we constructed an acyclic regulation. With circular references, one can encode reachability computations which cannot be directly expressed in first-order logic:

\[
(id) \, \delta(x, z) \lor (\delta(x, y) \land \text{says}_{\{id\}} \delta^+(y, z)) \mapsto \delta^+(x, z)
\]

Here, we assume that each point in a run encodes a graph. The edge relation is given by \( \delta \), and \( \delta^+ \) represents the transitive closure of \( \delta \). It can be shown that at the least fixed point \( v(\delta^+(x, z)) \) is available as an annotation iff there is a path from \( v(x) \) to \( v(z) \). We can show an EXPTIME lower bound by a reduction from first-order logic enriched with a least fixed point predicate (the system YF in [145]).
Appendix B

Permission to Speak: Proofs

B.1 Semantics

We repeat the semantic definitions for convenience.

Definition 4.3 (Models). Given countable sets of object names $O$, principal names $O_P \subseteq O$, $\Phi_1, \ldots, \Phi_n$ (where $\Phi_j$ is a set of predicate names of arity $j$), identifiers for rules $ID$, and $l : O_P \rightarrow 2^{ID}$, a model $M(O, O_P, \Phi_1, \ldots, \Phi_n, ID, l)$, abbreviated as $M$, is the tuple $(S, I_{\Phi_1}, \ldots, I_{\Phi_n}, \delta_L, \delta_O)$ where:

- $S$ is a set of states

- $I_{\Phi_j} : \Phi_j \times S \rightarrow 2^{O_j}$ is the interpretation of predicates of arity $j$. Given $p \in \Phi_j$, we will say that $p(o_1, \ldots, o_j)$ is true at state $s$ iff $(o_1, \ldots, o_j) \in I_{\Phi_j}(p, s)$.

- $\delta_L : S \times 2^{ID} \rightarrow 2^S$. $\delta_L(s, Id)$ corresponds to a description of $s$ according to the laws labeled with identifiers in $Id$ (taken conjunctively).

- $\delta_O : S \times O_P \rightarrow 2^S$. $\delta_O(s, A)$ corresponds to an idealization of $s$, for which the principal $A$ is held responsible.

For the axioms A3-A6 we need the following constraints C3-C6 (resply). For all $s \in S$:
C3 \( \delta_L(s, \text{Id}_A) \supseteq \delta_L(s, \text{Id}'_A) \) for all \( A \in O_P \) and \( \text{Id}_A \subseteq \text{Id}'_A \subseteq l(A) \)

C4 \( \delta_O(s, A) \neq \emptyset \) for all \( A \in O_P \)

C5 For all \( \{ A, B \} \subseteq O_P, \text{Id}_A \subseteq l(A), \text{Id}_B \subseteq l(B) \), and \( s' \in \delta_L(s, \text{Id}_A) \):

1. \( s' \in \delta_L(s, \text{Id}_B) \), or
2. There exists \( s_1 \in \delta_L(s, \text{Id}_A) \) such that for all \( s_2 \in \delta_O(s_1, B), s' \in \delta_L(s_2, \text{Id}_B) \)

C6 For all \( A \in O_P, \text{Id}_A \subseteq l(A), \) and \( s' \in \delta_L(s, \text{Id}_A) \):

There exists \( s_1 \in \delta_L(s, \text{Id}_A) \) such that for all \( s_2 \in \delta_O(s_1, A), s' \in \delta_L(s_2, \text{Id}_A) \)

Definition 4.4 (Semantics). Given a model \( M = (S, I_{\Phi_1}, ..., I_{\Phi_n}, \delta_L, \delta_O) \), \( s \in S \) and a propositional \( \varphi \in L \), the relation \( (M, s) \models \varphi \) is defined inductively as follows:

- \( (M, s) \models p(o_1, ..., o_j) \iff (o_1, ..., o_j) \in I_{\Phi_j}(p, s) \).
- The semantics of conjunction and negation is defined in the usual way.
- \( (M, s) \models \text{says}_{\text{Id}} \varphi \iff (M, s') \models \varphi, \) for all \( s' \in \delta_L(s, \text{Id}) \).
- \( (M, s) \models \mathcal{O}_A \varphi \iff (M, s') \models \varphi, \) for all \( s' \in \delta_O(s', A) \).

We can now define validity:

- \( \varphi \) is valid in a model \( M \) \( (M \models \varphi) \) iff for all \( s \in S, (M, s) \models \varphi \)
- \( \varphi \) is valid \( (\models \varphi) \) iff for all \( M \in \mathcal{M}, M \models \varphi \)

B.2 Soundness and Completeness

Lemma B.1 (Soundness). Given a propositional \( \varphi \in L, \) if \( \not\models \varphi, \) then \( \models \varphi \)
Proof. We need to show that the axioms are valid, and that the rules preserve validity. It is well-known that the axioms $A1$ and $A2$ are valid, and that $R1$ and $R2$ preserve validity in all Kripke structures. The validity of $A3$ and $A4$ can easily be shown using $C3$ and $C4$. We discuss the case for $A5$.

Suppose $A5$ is not valid. There exists $M$, $s$, $\varphi$, $A$, $B$, $Id_A$ and $Id_B$ such that:

- $(M, s) \models Id_A(P_B \text{ says } Id_B \varphi)$
- $(M, s) \not\models Id_B \varphi$, and
- $(M, s) \not\models Id_A \varphi$

Since $(M, s) \not\models Id_A \varphi$, there exists $s' \in \delta_L(s, Id_A)$ such that $(M, s') \not\models \varphi$. Since $C5$ holds, there are two cases to consider:

1. If $s' \in \delta_L(s, Id_B)$, then $(M, s) \not\models Id_B \varphi$ giving us a contradiction.

2. If there exists $s_1 \in \delta_L(s, Id_A)$ such that for all $s_2 \in \delta_O(s_1, B)$, $s' \in \delta_L(s_2, Id_B)$, then:
   - $(M, s_1) \models O_B \neg \text{ says } Id_B \varphi$
   - $(M, s) \not\models Id_A(\neg O_B \neg \text{ says } Id_B \varphi)$

Hence, $(M, s) \not\models \text{ says } Id_A(\mathcal{P}_B \text{ says } Id_B \varphi)$ (since $\mathcal{P}_B \varphi = \neg O_B \neg \varphi$), giving us a contradiction.

Hence, $A5$ is valid. The proof for $A6$ is similar. □

Lemma B.2 (Completeness). Given a propositional $\varphi \in L$, if $\models \varphi$, then $\vdash \varphi$

The rest of this section gives the proof. We will use a canonical model argument (c.f. [65]). We show the contrapositive, i.e., if $\not\vdash \varphi$, then $\not\models \varphi$. In other words, if $\not\vdash \varphi$ then there exist $M$ and $s$ such that $(M, s) \not\models \neg \varphi$. We begin with some terminology.

We say that $\varphi$ is consistent if $\neg \varphi$ is not provable ($\not\vdash \neg \varphi$). A finite set of formulas $\{\varphi_1, ..., \varphi_n\}$ is consistent if $\varphi_1 \land ... \land \varphi_n$ is consistent. An infinite set of formulas is
consistent if every finite subset is consistent. A set of formulas $\Delta$ is \textit{maximal consistent} if for all \( \varphi \in L - \Delta \), \( \Delta \cup \{ \varphi \} \) is inconsistent. The following are properties of maximal consistent sets:

**Proposition B.1.** Given a maximal consistent set $\Delta$:

1. For all $\varphi \in L$, exactly one of $\varphi \in \Delta$ or $\neg \varphi \in \Delta$
2. If $\vdash \varphi \Rightarrow \psi$ and $\varphi \in \Delta$, then $\psi \in \Delta$
3. If $\vdash \varphi$, then $\varphi \in \Delta$ and $Q\varphi \in \Delta$ (for all modalities $Q$)

The proof is straightforward. We now define the \textit{canonical model}, in which every consistent formula is true at some state:

**Definition B.1 (Canonical Model).** The canonical model $M = (S, I_{\Phi_1}, ..., I_{\Phi_n}, \delta_L, \delta_O)$ is such that:

- $S$ is the set of all maximal consistent sets
- $(o_1, ..., o_j) \in I_{\Phi_j}(p, \Delta)$ iff $p(o_1, ..., o_j) \in \Delta$
- $\Delta' \in \delta_L(\Delta, Id_A)$ iff for all $\varphi$, if $\text{says}_{Id_A} \varphi \in \Delta$, then $\varphi \in \Delta'$
- $\Delta' \in \delta_O(\Delta, A)$ iff for all $\varphi$, if $O_A \varphi \in \Delta$, then $\varphi \in \Delta'$

We now show that the canonical model satisfies the frame constraints:

**Proposition B.2.** The canonical model satisfies the frame constraints $C3$-$C6$

\textit{Proof.} The proof that $C3$ and $C4$ hold are left to the reader. We discuss the case for $C5$. Given the canonical model $M = (S, I_{\Phi_1}, ..., I_{\Phi_n}, \delta_L, \delta_O)$, $\Delta \in S$, and suppose for the purpose of contradiction that there exists $\Delta' \in \delta_L(\Delta, Id_A)$ such that:

- $\Delta' \not\in \delta_L(\Delta, Id_B)$. By construction, there exists $\text{says}_{Id_B} \psi \in \Delta$ such that $\neg \psi \in \Delta'$. 

202
• For all $\Delta_1 \in \delta_L(\Delta, Id_A)$, there exists $\Delta_2 \in \delta_O(\Delta_1, B)$, $\Delta' \not\in \delta_L(\Delta_2, Id_B)$. By Proposition B.3 (below), there exists says$_{Id_A} P_B$ says$_{Id_B} \varphi \in \Delta$ such that $\neg \varphi \in \Delta'$.

Using Proposition B.1, says$_{Id_A} (\varphi \lor \psi) \in \Delta$ and says$_{Id_A} P_B$ says$_{Id_B} (\varphi \lor \psi) \in \Delta$. So, says$_{Id_A} (\varphi \lor \psi) \in \Delta$, and hence $\varphi \lor \psi \in \Delta'$. That is $\varphi \in \Delta'$ or $\psi \in \Delta'$, which contradicts the fact that $\neg \varphi \in \Delta'$ and $\neg \psi \in \Delta'$. The proof of C6 is similar.  

**Proposition B.3.** Given the canonical model $M = (S, I_{\Phi_1}, \ldots, I_{\Phi_n}, \delta_L, \delta_O)$, for all $\Delta \in S$, $\{A, B\} \subseteq O_P$, $Id_A \subseteq l(A)$, $Id_B \subseteq l(B)$, and $\Delta' \in \delta_L(\Delta, Id_A)$:

- If for all $\Delta_1 \in \delta_L(\Delta, Id_A)$, there exists $\Delta_2 \in \delta_O(\Delta_1, B)$, $\Delta' \not\in \delta_L(\Delta_2, Id_B)$, then there exists says$_{Id_A} P_B$ says$_{Id_B} \varphi \in \Delta$ and $\neg \varphi \in \Delta'$

**Proof.** Fix $\Delta$, $A$, $B$, $Id_A$, $Id_B$ and $\Delta' \in \delta_L(\Delta, Id_A)$. We proceed by contradiction. Suppose for all $\varphi \in L$, if says$_{Id_A} P_B$ says$_{Id_B} \varphi \in \Delta$, then $\varphi \in \Delta'$. Let $F$ be the smallest set such that:

- If says$_{Id_A} \varphi \in \Delta$, then $\varphi \in F$, and
- If $\neg \psi \in \Delta'$, then $O_B \neg$ says$_{Id_B} \psi \in F$.

We claim that $F$ is consistent.\(^1\) Suppose not:

1. There exists $\{\varphi_1, \ldots, \varphi_n, O_B \neg$ says$_{Id_B} \psi_1, \ldots, O_B \neg$ says$_{Id_B} \psi_m\} \subseteq F$ such that:

$\vdash \neg(\varphi_1 \land \ldots \land \varphi_n \land O_B \neg$ says$_{Id_B} \psi_1 \land \ldots \land O_B \neg$ says$_{Id_B} \psi_m)$

2. $\vdash \varphi_1 \land \ldots \land \varphi_n$ $\Rightarrow$ $P_B$ says$_{Id_B} (\psi_1 \lor \ldots \lor \psi_m)$ (from (1) using A1, A4, R1 and R2)

\(^1\)Note that if there exists $\varphi$ such that $\Delta \vdash$ says$_{Id_A} \varphi$ and $\Delta \vdash$ says$_{Id_A} \neg \varphi$, then $\delta_L(\Delta, Id_A) = \emptyset$, and C5 and C6 are vacuously satisfied. In Proposition B.3 (and Proposition B.5 in Section B.3), the contradiction applies only to cases where there exists $\Delta' \in \delta_L(\Delta, Id_A)$, and hence, no such $\varphi$ exists.
(3) \( \vdash \text{says}_{IdA}(\varphi_1 \land \cdots \land \varphi_n \Rightarrow \text{P}_{B} \text{says}_{IdB}(\psi_1 \lor \cdots \lor \psi_m)) \in \Delta \) (from (2) using R2)

(4) By construction, \( \text{says}_{IdA} \varphi_i \in \Delta \) for all \( 1 \leq i \leq n \). So, using A2 and (3), we can derive that \( \text{says}_{IdA} \text{P}_{B} \text{says}_{IdB}(\psi_1 \lor \cdots \lor \psi_m) \in \Delta \). As a result, \( \psi_1 \lor \cdots \lor \psi_m \in \Delta' \), and there exists \( \psi_i \in \Delta' \) where \( 1 \leq i \leq m \).

(5) By construction, \( \neg \psi_i \in \Delta' \) for all \( 1 \leq i \leq m \), which together with (4) contradicts the consistency of \( \Delta' \).

We can extend \( F \) into a maximal consistent set \( \Delta_1 \) such that \( \Delta_1 \in \delta_{L}(\Delta, IdA) \). \( \text{P}_{B} \text{says}_{IdB} \varphi \in \Delta_1 \) iff \( \varphi \in \Delta' \). So, for all \( \Delta_2 \in \delta_{O}(\Delta_1, B) \), if \( \text{says}_{IdB} \varphi \in \Delta_2 \), then \( \varphi \in \Delta' \). This suffices to conclude that \( \Delta' \in \delta_{L}(\Delta_2, IdB) \) for all \( \Delta_2 \in \delta_{O}(\Delta_1, B) \), giving us a contradiction.

The completeness proof is now finished in the usual way (see, for example, [65]). Given the canonical model \( M \) and a state \( \Delta \), it is easy to show that for all \( \varphi \in L \), 
(\( M, \Delta \)) \models \varphi \iff \varphi \in \Delta \). Furthermore, given a consistent \( \varphi \), we can construct a maximal consistent set \( \Delta \) such that \( \varphi \in \Delta \). As a result, for every consistent \( \varphi \), there exists a state \( \Delta \) in the canonical model such that \( (M, \Delta) \models \varphi \). Hence, if \( \not\models \varphi \), then \( \not\models \varphi \).

We observe that compactness follows as a corollary of the existence of the canonical model:

**Corollary 4.1** (Compactness). An infinite set of formulas is satisfiable iff every finite subset is satisfiable.

Given an infinite set of formulas \( \Delta \), if every finite subset is satisfiable, then by soundness, every finite subset of \( \Delta \) is consistent. And, by definition, \( \Delta \) is consistent. We can extend \( \Delta \) into a maximal consistent set, corresponding to a state in the canonical model.
B.3 Decidability

In this section, we adapt the completeness proof to show the bounded-model property, i.e., if \( \phi \) is satisfiable, then it is satisfiable in a model of bounded size (exponential in the size of \( \phi \)). We begin by defining the set of subformulas:

**Definition B.2 (Subformulas).** Given a propositional \( \phi \in L \), the set of subformulas \( \text{sub}(\phi) \) is the smallest set such that:

1. \( \phi \in \text{sub}(\phi) \)
2. If \( \varphi \in \text{sub}(\phi) \), then \( \neg\varphi \in \text{sub}(\phi) \) (\( \neg\neg\varphi \) is identified with \( \varphi \))
3. If \( \varphi \land \psi \in \text{sub}(\phi) \), then \( \varphi \in \text{sub}(\phi) \) and \( \psi \in \text{sub}(\phi) \)
4. If \( \text{O}_A \psi \in \text{sub}(\phi) \) or \( \text{says}_{l_A} \psi \in \text{sub}(\phi) \), then \( \psi \in \text{sub}(\phi) \)
5. If \( \text{says}_{Id_A} \psi_1 \in \text{sub}(\phi) \) and \( \text{says}_{Id'_A} \psi_2 \in \text{sub}(\phi) \) such that \( Id_A \subseteq l(A) \) and \( Id'_A \subseteq l(A) \), then \( \text{says}_{Id_A \cup Id'_A} \psi_1 \in \text{sub}(\phi) \)
6. If \( \text{says}_{Id_A} (\lor \Delta_1) \in \text{sub}(\phi) \) and \( \text{says}_{Id_B} (\lor \Delta_2) \in \text{sub}(\phi) \), then \( \text{says}_{Id_A} (\lor \Delta_2) \in \text{sub}(\phi) \) and \( \text{says}_{Id_A} (\lor (\Delta_1 \cup \Delta_2)) \in \text{sub}(\phi) \)
7. If \( \text{says}_{Id_A} \psi_1 \in \text{sub}(\phi) \) and \( Id_A \subseteq l(A) \), then \( \text{P}_A \text{says}_{Id_A} \psi_1 \in \text{sub}(\phi) \)

The last three clauses in Definition B.2 are used to ensure that \( C5 \) and \( C6 \) hold. Note that in Clause 5, we consider disjunction over sets of formulas \( \Delta_1 \) and \( \Delta_2 \). Formulas which are not disjunctions are understood as disjunctions over singleton sets, e.g., \( \varphi \land \psi = \lor \{ \varphi \land \psi \} \). To obtain the analog of Proposition B.2, we need to ensure that formulas appearing within the scope of \( \text{says} \) are closed under disjunction. We use sets of formulas to ensure that only finitely many disjunctions are introduced, i.e., a disjunct need not be repeated. Due to Clauses 5 and 6, the number of subformulas is exponential in the size of \( \phi \). It is possible to eliminate both these clauses, by filtering...
the model that we construct here. But, this further filtration is not needed for the results proved in this work. Clause 7 is key to obtaining the analog of Proposition B.3.

Given $\phi \in L$, we will consider maximal consistent sets w.r.t. $\text{sub}(\phi)$. A set $\Delta \subseteq \text{sub}(\phi)$ is said to be maximal consistent iff $\Delta$ is consistent and for all $\psi \in \text{sub}(\phi) - \Delta$, $\Delta \cup \{\psi\}$ is inconsistent. We write $\Delta \vdash \varphi$ to denote $\vdash \bigwedge \Delta \Rightarrow \varphi$. The definition of the canonical model needs a few changes:

**Definition B.3 (Canonical Model of $\phi$).** The canonical model of $\phi$, denoted $M_\phi = (S, I_{\phi_1}, ..., I_{\phi_n}, \delta_L, \delta_O)$, is such that:

- $S$ is the set of all maximal consistent sets w.r.t. $\text{sub}(\phi)$
- $(o_1, ..., o_j) \in I_{\phi_j}(p, \Delta)$ iff $p(o_1, ..., o_j) \in \Delta$
- $\Delta' \in \delta_L(\Delta, \text{Id}_A)$ iff for all $\psi \in \text{sub}(\phi)$ and $\text{Id}_A' \subseteq \text{Id}_A$, if $\text{says}_{\text{Id}_A'} \psi \in \Delta$, then $\psi \in \Delta'$
- $\Delta' \in \delta_O(\Delta, A)$ iff for all $\psi \in \text{sub}(\phi)$, if $\text{O}_A \psi \in \Delta$, then $\psi \in \Delta'$.

We will show that the canonical model of $\phi$ satisfies the frame constraints. We adapt Propositions B.2 and B.3 to obtain Propositions B.4 and B.5 resply.

**Proposition B.4.** The canonical model of $\phi$ satisfies the frame constraints C3-C6

*Proof.* The proof that C3 and C4 hold are left to the reader. We discuss the case for C5. Given $M_\phi = (S, I_{\phi_1}, ..., I_{\phi_n}, \delta_L, \delta_O)$, consider some $\Delta \in S$. If $\delta_L(\Delta, \text{Id}_A) = \emptyset$, then C5 is vacuously satisfied. Otherwise, let $\Delta' \in \delta_L(\Delta, \text{Id}_A)$. There are two cases to consider.

First, we have the boundary case, where there is no subformula $\text{says}_{\text{Id}_A'} \varphi' \in \text{sub}(\phi)$ such that $\text{Id}_A' \subseteq \text{Id}_A$. By definition, $\delta_L(\Delta, \text{Id}_A) = S$. Consider the set $F \subseteq \text{sub}(\phi)$ such that $\psi \in F$ iff $\psi$ is of the form $\text{O}_B \neg \text{says}_{\text{Id}_B} \varphi$ and $\neg \varphi \in \Delta'$. We claim that $F$ is consistent. Since $\Delta'$ is consistent, we can construct a model $M'$ with states $S'$, and $s' \in S'$ such that $(M', s') \models \bigwedge \Delta'$. Without loss of generality, we can assume
such that for all $\Delta_2 \in \delta_O(\Delta_1, B)$, we have $\Delta' \in \delta_L(\Delta_2, I_B)$. Since $\delta_L(\Delta, I_A) = S$, we have $\Delta_1 \in \delta_L(\Delta, I_A)$, and $C5$ is satisfied.

For the second case we proceed as follows. Let $I_A^\ast$ be the largest subset of $I_A$ such that there is a subformula $\text{says}_{I_A^\ast} \varphi' \in \text{sub}(\phi)$. The existence of a largest subset is guaranteed by Clause 5 in Definition B.2. Fix $I_B \subseteq l(B)$. If there is no subformula $\text{says}_{I_B^\ast} \psi' \in \text{sub}(\phi)$ with $I_B^\ast \subseteq I_B$, then $\delta_L(\Delta, I_A) \subseteq \delta_L(\Delta, I_B) = S$, and $C5$ is satisfied. Otherwise, let $I_B^\ast$ be the largest subset of $I_B$ such that there is a subformula $\text{says}_{I_B^\ast} \psi' \in \text{sub}(\phi)$. We proceed by contradiction analogous to the completeness proof:

- $\Delta' \not\in \delta_L(\Delta, I_B)$. By construction, there exists $\psi \in \text{sub}(\phi)$ such that $\text{says}_{I_B^\ast} \psi \in \Delta$ for some $I_B^\ast \subseteq l(B)$ and $\neg \psi \in \Delta'$. And, using $A3$, $\Delta \vdash \text{says}_{I_B^\ast} \psi$.

- For all $\Delta_1 \in \delta_L(\Delta, I_A)$, there exists $\Delta_2 \in \delta_O(\Delta_1, B)$, $\Delta' \not\in \delta_L(\Delta_2, I_B)$. By Proposition B.5 (below), there exists $\varphi \in \text{sub}(\phi)$ such that $\text{says}_{I_B^\ast} \varphi \in \text{sub}(\phi)$, $\Delta \vdash \text{says}_{I_A^\ast} \mathcal{P}_B \text{says}_{I_B^\ast} \varphi$ and $\neg \varphi \in \Delta'$.

Since $\Delta \vdash \text{says}_{I_B^\ast} (\varphi \lor \psi)$ and $\Delta \vdash \text{says}_{I_A^\ast} \mathcal{P}_B \text{says}_{I_B^\ast} (\varphi \lor \psi)$, we have $\Delta \vdash \text{says}_{I_A^\ast} (\varphi \lor \psi)$. Using Clause 6 in Definition B.2, there exists $\text{says}_{I_A^\ast} \varphi_1 \in \text{sub}(\phi)$ such that $\vdash \varphi_1 \iff (\varphi \lor \psi)$. As a result, $\text{says}_{I_A^\ast} \varphi_1 \in \Delta$, and hence $\varphi_1 \in \Delta'$. Since $\vdash \varphi_1 \iff (\varphi \lor \psi)$, we have $\varphi \in \Delta'$ or $\psi \in \Delta'$, which contradicts the fact that $\neg \varphi \in \Delta'$ and $\neg \psi \in \Delta'$. The proof of $C6$ is similar.

**Proposition B.5.** Given $\phi \in L$, $I_A \subseteq l(A)$ and $I_B \subseteq l(B)$ such that there are largest subsets $I_A^\ast \subseteq I_A$ and $I_B^\ast \subseteq I_B$ with formulas $\text{says}_{I_A^\ast} \varphi' \in \text{sub}(\phi)$ and $\text{says}_{I_B^\ast} \psi' \in \text{sub}(\phi)$, let $M_\phi = (S, I_{\Phi_1}, ..., I_{\Phi_n}, \delta_L, \delta_O)$ be the canonical model of $\phi$. Then, for all $\Delta \in S$ and $\Delta' \in \delta_L(\Delta, I_A)$:
• If for all $\Delta_1 \in \delta_{L}(\Delta, Id_A)$, there exists $\Delta_2 \in \delta_{O}(\Delta_1, B)$ such that $\Delta' \not\in \delta_{L}(\Delta_2, Id_B)$, then there exists $\phi \in \text{sub}(\phi)$ such that says$_{Id_B} \phi \in \text{sub}(\phi)$, $\Delta \vdash$ says$_{Id_A} \mathcal{P}_B$ says$_{Id_B} \phi$ and $\neg \phi \in \Delta'$

Proof. Fix $\Delta$ and $\Delta' \in \delta_{L}(\Delta, Id_A)$. We proceed by contradiction. Suppose for all $\phi \in \text{sub}(\phi)$ with says$_{Id_B} \phi \in \text{sub}(\phi)$, if $\Delta \vdash$ says$_{Id_A} \mathcal{P}_B$ says$_{Id_B} \phi$, then $\neg \phi \not\in \Delta'$. Let $F$ be the smallest set such that:

• If says$_{Id_A} \phi \in \Delta$ for some $Id_A' \subseteq Id_A$, then $\phi \in F$

• If $\neg \psi \in \Delta'$ and $O_B \neg$ says$_{Id_B} \psi \in \text{sub}(\phi)$, then $O_B \neg$ says$_{Id_B} \psi \in F$.

We claim that $F$ is consistent (see Footnote 1). Suppose not:

1. There exists $\{\phi_1, ..., \phi_n, O_B \neg$ says$_{Id_B} \psi_1, ..., O_B \neg$ says$_{Id_B} \psi_m\} \subseteq F$ such that: $\vdash \neg(\phi_1 \land ... \land \phi_n \land O_B \neg$ says$_{Id_B} \psi_1 \land ... \land O_B \neg$ says$_{Id_B} \psi_m)$

2. $\vdash \phi_1 \land ... \land \phi_n \Rightarrow \mathcal{P}_B \text{ says}_B(\psi_1 \lor ... \lor \psi_m)$ (from (1) using A1, A4, R1 and R2)

3. $\vdash$ says$_{Id_A}^* (\phi_1 \land ... \land \phi_n \Rightarrow \mathcal{P}_B \text{ says}_B(\psi_1 \lor ... \lor \psi_m)) \in \Delta$ (from (2) using R2)

4. By construction, $\Delta \vdash$ says$_{Id_A}^* \phi_i$ for all $1 \leq i \leq n$. So, using A2 and (3), we can derive that $\Delta \vdash$ says$_{Id_A}^* \mathcal{P}_B$ says$_{Id_B}^* (\psi_1 \lor ... \lor \psi_m)$. Using Clause 6 in Definition B.2, there exists $\psi' \in \text{sub}(\phi)$ such that says$_{Id_B}^* \psi' \in \text{sub}(\phi)$ and $\vdash \psi' \Leftrightarrow (\psi_1 \lor ... \lor \psi_m)$. It follows that $\Delta \vdash$ says$_{Id_A}^* \mathcal{P}_B$ says$_{Id_B}^* \psi'$, and by assumption, $\neg \psi' \not\in \Delta'$, i.e., $\Delta' \vdash \psi'$. As a result, $\Delta' \vdash \psi_1 \lor ... \lor \psi_m$, and there exists $1 \leq i \leq m$ such that $\psi_i \in \Delta'$ (since $\psi_i \in \text{sub}(\phi)$).

5. By construction, $\neg \psi_i \in \Delta'$ for all $1 \leq i \leq m$, which together with (4) gives us a contradiction.

We can extend $F$ into a maximal consistent set $\Delta_1$ such that $\Delta_1 \in \delta_{L}(\Delta, Id_A)$. Consider $\Delta_2 \in \delta_{O}(\Delta_1, B)$. We claim that for all $Id_B' \subseteq Id_B$, if says$_{Id_B'} \phi \in \Delta_2$, then
\( \varphi \in \Delta'. \) Suppose not. There exists \( \text{say}_{Id} \varphi \in \Delta_2 \) such that \( \neg \varphi \in \Delta'. \) Using Clauses 5 and 6 in Definition B.2, it follows that \( \text{say}_{Id} \varphi \in \text{sub}(\varphi) \), and using A3, \( \text{say}_{Id} \varphi \in \Delta_2. \) Since \( \text{say}_{Id} \varphi \in \text{sub}(\varphi) \), by Clause 7 in Definition B.2, \( \text{O}_B \neg \text{say}_{Id} \varphi \in \text{sub}(\varphi). \) By construction, \( \text{O}_B \neg \text{say}_{Id} \varphi \in \Delta_1 \), and so, \( \neg \text{say}_{Id} \varphi \in \Delta_2 \), contradicting the consistency of \( \Delta_2. \) This suffices to conclude that \( \Delta' \in \delta_L(\Delta_2, Id_B) \) for all \( \Delta_2 \in \delta_O(\Delta_1, B) \), giving us a contradiction.

A standard argument (see, for example, [65]) can be used to show that for all \( \varphi \in \text{sub}(\varphi) \), \( (M_\varphi, \Delta) \models \varphi \) iff \( \varphi \in \Delta. \) We can now establish decidability:

**Theorem 4.2** (Decidability). *Given a propositional \( \varphi \in L \), the problem of checking whether \( \vDash \varphi \) is decidable*

**Proof.** Decidability is established via the bounded model property:

\[
\phi \text{ is satisfiable in } M_\phi \text{ iff } \phi \text{ is satisfiable}
\]

One direction is trivial, i.e., if \( \phi \) is satisfiable in \( M_\phi \), then \( \phi \) is satisfiable (by definition). For the other direction, we can use a standard filtration argument, to show that \( M_\phi \) can be obtained from the canonical model (Definition B.1).

**B.4 Complexity**

In this section, we show that testing satisfiability in the propositional fragment of the language \( L \) (Section 4.3.2) is NEXPTIME-hard. We proceed by reduction from the halting problem for non-deterministic Turing machines. Specifically, given a Turing machine and input \( w \), we will construct a formula \( \varphi \) such that \( \varphi \) is satisfiable iff the TM halts on \( w \) in at most \( 2^{\|w\|} - 1 \) steps. We begin by giving some intuition for the complexity, and build some machinery that is needed for the proof.

Halpern and Moses [65] give tableaux algorithms for several modal logics (e.g., K, KD, S4, S5), for which the decision problem is PSPACE-complete. The key idea is that while searching for a state satisfying \( \varphi \), we need to keep track of at
most one of the accessible states. Suppose $\text{say}_{Id}$ was a $K$ modality. Then, given $\Delta$ and $\{\Delta', \Delta''\} \subseteq \delta_L(\Delta, Id)$, $\Delta'$ and $\Delta''$ are independent given $\Delta$. In our case, the constraints C5 and C6 introduce dependence between sibling states.

We begin by defining the notion of a witness which is important in subsequent proofs:

**Definition B.4 (Witness).** Given a model $M$, the states $s, s'$ and $s''$ and $\{A, B\} \subseteq O$, we say that $s''$ is a witness for $s'$ w.r.t. $s$, $A$ and $B$, denoted $W_{(A,B)}^s(s'', s')$, if $\{s', s''\} \subseteq \delta_L(s, l(A))$ and for all $s_2 \in \delta_O(s'', B)$, $s' \in \delta_L(s_2, l(B))$.

We mention some properties of witnesses:

**Proposition B.6.** Given a model $M$ and state $s$:

1. For all $s'$ and $s''$ such that $W_{(A,B)}^s(s'', s')$, if $(M, s'') \models O_B \text{say}_{l(B)} \varphi$, then $(M, s') \models \varphi$.

2. Given $s' \in \delta_L(s, l(A))$, there exists $s''$ such that $W_{(A,A)}^s(s'', s')$.

3. Given $s' \in \delta_L(s, l(A))$, for all $B \in O$ such that $\delta_L(s, l(B)) = \emptyset$, there exists $s''$ such that $W_{(A,B)}^s(s'', s')$.

Item 1 is immediate from the definition of witnesses. Items 2 and 3 are consequences of C6 and C5 respy.

The building blocks for the reduction are $n$-bit counters. Given a natural number $p$ such that $0 \leq p \leq 2^n - 1$, we express $p$ in binary, using $n$ propositions $p_{n-1}p_{n-2}...p_0$, where $p_{n-1}$ is the most significant bit, and $p_0$ is the least significant bit. For example, $p = 2^n - 1$ iff $\bigwedge_{i=0}^{n-1} p_i$, and $p = 0$ iff $\bigwedge_{i=0}^{n-1} \neg p_i$. Equality and comparison tests, e.g., $p = i$ and $p > i$, can be expressed by formulas of size polynomial in the number of bits $n$. We will use such formulas without definition.

We now give a schema for an $n$-bit adder, using a technique from [65]. Informally, the goal is to create a formula such that $\forall 0 \leq i < 2^n - 1 : (p = i) \Rightarrow O_A \text{say}_{l(A)}(p = i + 1)$. We start with schemas for flipping a bit (a) and preserving a bit (b):
\[ a(\psi, A) = (\psi \Rightarrow O_A \text{ says}_l(A) \neg \psi) \land (\neg \psi \Rightarrow O_A \text{ says}_l(A) \psi) \]

\[ b(\psi, A) = (\psi \Rightarrow O_A \text{ says}_l(A) \psi) \land (\neg \psi \Rightarrow O_A \text{ says}_l(A) \neg \psi) \]

The adder is given by the following schema:

\[ c_{+1}(p, A, n) = \bigwedge_{i=0}^{n-1} \left( (\bigwedge_{j=0}^i p_j \Rightarrow a(p_i, A)) \land (\neg \bigwedge_{j=0}^i p_j \Rightarrow b(p_i, A)) \right) \]

The first clause says that when \( p_i \) and all the less significant bits are 1, then \( p_i \) is flipped (\( a(p_i, A) \)). Otherwise, the second clause ensures that \( p_i \) is preserved (\( b(p_i, A) \)). We can show the following:

**Proposition B.7.** Given a model \( M \) and state \( s \) such that \( (M, s) \models c_{+1}(p, A, n) \), if \( (M, s) \models p = i \), then \( (M, s) \models O_A \text{ says}_l(A)(p = i + 1) \) for all \( 0 \leq i < 2^n - 1 \)

The proof is straightforward. We now discuss an example to show how the counter \( c_{+1}(p, A, n) \) is used to construct a formula which is satisfied only in models of size exponential in the formula. Consider formulas defined by the following schema:

\[ \phi(\psi, p, A, n) = \phi_1(\psi, p, A, n) \land \phi_2(\psi, p, A, n) \land \phi_3(\psi, p, A, n) \land \phi_4(\psi, p, A, n) \]

\[ \phi_1(\psi, p, A, n) = \text{ says}_l(A)(\neg \psi \Rightarrow O_A \text{ says}_l(A)(\psi \Rightarrow p = 0)) \]

\[ \phi_2(\psi, p, A, n) = \text{ says}_l(A)(\psi \Rightarrow (p \neq 2^n - 1) \Rightarrow c_{+1}(p, A, n)) \]

\[ \phi_3(\psi, p, A, n) = \text{ says}_l(A)(\psi \Rightarrow (p = 2^n - 1) \Rightarrow O_A \text{ says}_l(A) \bot) \]

\[ \phi_4(\psi, p, A, n) = \neg \text{ says}_l(A)(\neg (\psi \land (p = 2^n - 1)) \]

A model satisfying \( \phi(r, p, A, n) \) for some atomic proposition \( r \) is shown in Figure B.1. For each state \( s_i \) (for \( 0 < i \leq 2^n - 1 \)), we have \( W^s_{(A, A)}(s_{i-1}, s_i) \). The following can be shown:

**Proposition B.8.** Given a model \( M \) and state \( s \) such that \( (M, s) \models \phi(r, p, A, n) \) for some atomic proposition \( r \), then:

1. If there exists \( s' \in \delta_L(s, l(A)) \) such that \( (M, s') \models r \land (p = j) \), then there exists \( s'' \in \delta_L(s, l(A)) \) such that \( (M, s'') \models r \land (p = j - 1) \), for all \( 0 < j \leq 2^n - 1 \)

2. There exists \( s' \in \delta_L(s, l(A)) \) such that \( (M, s') \models r \land (p = 2^n - 1) \)

211
Figure B.1: A model satisfying $\phi(r, p, A, n)$. The solid lines correspond to $\delta_L$, e.g., $s' \in \delta_L(s, l(A))$. The dashed lines correspond to $\delta_O$, e.g., $s' \in \delta_O(s', A)$. Only the relevant accessibility relations are shown.

Proof. For Item 1, we proceed as follows. We are given a state $s'$ such that $(M, s') \models r \land (p = j)$ for some $0 < j \leq 2^n - 1$. By Proposition B.6 Item 2, there exists $s''$ such that $W_{(A, A)}^r(s'', s')$. We claim that $(M, s'') \models r \land (p = j - 1)$. Suppose not. There are two cases to consider.

If $(M, s'') \models \neg r$, then $(M, s'') \models O_A \text{says}_{l(A)}(r \Rightarrow p = 0)$ (due to $\phi_1(r, p, A, n)$), and by Proposition B.6 Item 1, $(M, s') \models (p = 0)$, which contradicts the fact that $(M, s') \models (p = j)$ for some $j > 0$.

If $(M, s'') \models r \land (p \neq j - 1)$, then $(M, s'') \models O_A \text{says}_{l(A)}(p = j + 1)$ (due to $\phi_2(r, p, A, n)$). Hence, $(M, s'') \models O_A \text{says}_{l(A)}(p \neq j)$, and by Proposition B.6 Item 1, $(M, s') \models (p \neq j)$, giving us a contradiction.

For Item 2, the existence of the state is guaranteed by $\phi_4(r, p, A, n)$.

To simulate the position of the read-head of the Turing machine, we will need a corresponding subtraction operation. Informally, the goal is to construct a formula such that $\forall 0 < i \leq 2^n - 1 : p = i \Rightarrow O_A \text{says}_{l(A)}(p = i - 1)$. We assume, without loss of
generality, that the tape is bounded on the left, and moving to the left (subtracting) from position 0 leaves the position unchanged. The schema is as follows:

\[ c_{-1}(p, A, n) = \bigwedge_{i=0}^{n-1} \left( \bigwedge_{j=0}^i \neg p_j \Rightarrow a(p_i, A) \right) \land (\neg \bigwedge_{j=0}^i \neg p_j \Rightarrow b(p_i, A)) \]

The first clause says that when \( p_i \) and all the less significant bits are 0, then \( p_i \) is flipped \((a(p_i, A))\). Otherwise, the second clause ensures that \( p_i \) is preserved \((b(p_i, A))\).

We can show the following:

**Proposition B.9.** Given a model \( M \) and state \( s \) such that \((M, s) \models c_{-1}(p, A, n)\), if \((M, s) \models p = i\), then \((M, s) \models O_A \text{ says}_{l(A)}(p = i - 1)\) for all \(0 < i \leq 2^n - 1\).

The counters give us a mechanism to represent the number of steps in the computation and the position of the read-head. We also need a mechanism for reading from and writing to the tape. When a symbol is written to the tape, we will guess the next time at which it will be read. We use (an obligation on) a second principal to hold the guess. The guess is represented by the following schema, denoted by \( c_>(p, B, n)\):

\[ c_>(p, B, n) = \bigvee_{i=0}^{n-1} \left( \bigwedge_{j=i+1}^{n-1} b(p_j, B) \right) \land \neg p_i \land O_B \text{ says}_{l(B)} p_i \]

We can show the following:

**Proposition B.10.** Given a model \( M \) and state \( s \) such that \((M, s) \models c_>(p, B, n)\), if \((M, s) \models p = i\), then there exists \( j \) such that \(0 < j \leq 2^n - 1\) and \((M, s) \models O_B \text{ says}_{l(B)}(p = i + j)\), for all \(0 \leq i < 2^n - 1\).

We now define a Turing machine and its runs:

**Definition B.5 (Turing Machine).** A (non-deterministic) Turing machine is the tuple \( T = (Q, \Sigma, \delta, q_0, F) \), where \( Q \) is a set of states, \( \Sigma \) is a finite alphabet, \( \delta \subseteq Q \times \Sigma \times \Sigma \times \{0, 1\} \times Q \) is the transition relation, \( q_0 \in Q \) is the start state, and \( F \subseteq Q \) is the set of accepting states. We assume that for all \( q \in F \), there is no \((a, b, d, q')\) such that \((q, a, b, d, q') \in \delta\).

**Definition B.6 (Runs of a Turing Machine).** Given \( T = (Q, \Sigma, \delta, q_0, F) \), a configuration is a tuple \( \sigma = (q, i, \Gamma) \), where \( q \in Q \) is the current state, \( i \in N \) is the position of the read head and \( \Gamma : N \to \Sigma \) is the tape.
A run of $T$ of length $m$ on input $w$ is the tuple $(\sigma_0, ..., \sigma_m)$, where $\sigma_0 = (q_0, 0, \Gamma_0)$, where $\Gamma_0(i) = w_i$ for $1 \leq i \leq |w|$, and $\Gamma_0(i) = \_b$ otherwise. In addition, for $\sigma_i = (q_i, p_i, \Gamma_i)$ and $\sigma_{i+1} = (q_{i+1}, p_{i+1}, \Gamma_{i+1})$, we require that there exists $(q_i, a, b, d, q_{i+1}) \in \delta$ such that:

- $\Gamma_i(p_i) = a$ and $\Gamma_{i+1}(p_i) = b$, and for all $j \neq p_i$ $\Gamma_i(j) = \Gamma_{i+1}(j)$
- If $d = 0$, then $p_{i+1} = p_i - 1$ if $p_i > 0$, and $p_{i+1} = 0$ otherwise. If $d = 1$, then $p_{i+1} = p_i + 1$.

A run $(\sigma_0, ..., \sigma_m)$ is said to be accepting iff $\sigma_m = (q, p_m, \Gamma_m)$ for some $q \in F$. $T$ accepts an input $w$ in at most $m$ steps iff there is an accepting run of length at most $m$.

We begin by describing the main ideas in the reduction. Given a turing machine $T$ and input $w$, we will construct a formula $\varphi^{T,w}$ which is satisfiable iff $T$ accepts $w$ in at most $2^{\lvert w \rvert} - 1$ steps. We will use two principals $A$ and $B$. The obligations on $A$ will keep track of time and the position of the tape. The obligations on $B$ will ensure that the next state that reads from the current position obtains the appropriate alphabet. Two counters are used $t$ (for time with $|w| + 1$ bits) and $p$ (for position with $|w|$ bits). The states with time $0 \leq t \leq 2^{\lvert w \rvert} - 1$ will ensure that the input is written at the start of the tape, and blank symbols (\_b) are written to the right up to position $2^{\lvert w \rvert} - 1$. The states with time $2^{\lvert w \rvert} \leq t \leq 2^{\lvert w \rvert} + 1 - 1$ will correspond to transitions of $T$. We will then require the existence of a final state with $2^{\lvert w \rvert} \leq t \leq 2^{\lvert w \rvert} + 1 - 1$.

We now define the various parts. The first part is before the counters are started, which is indicated by the falsity of $r$:

$\varphi_{1}^{T,w} = \text{says}_{\lceil (A)}(\lnot r \Rightarrow \mathcal{O}_{A} \text{says}_{\lceil (A)}(r \Rightarrow (t = 0 \land p = 0)))$  
$\varphi_{2}^{T,w} = \text{says}_{\lceil (A)}(\lnot r \Rightarrow \bigwedge_{j=0}^{\lvert w \rvert-1} \mathcal{O}_{B} \text{says}_{\lceil (B)}((r \land t = j) \Rightarrow w_j))$  
$\varphi_{3}^{T,w} = \text{says}_{\lceil (A)}(\lnot r \Rightarrow \mathcal{O}_{B} \text{says}_{\lceil (B)}((r \land (\lvert w \rvert \leq t \leq 2^{\lvert w \rvert} - 1)) \Rightarrow \_b))$  
$\varphi_{1}^{T,w}$ ensures that a state satisfying $\lnot r$ can only serve as a C6 witness w.r.t. $A$ for the first state in the run (with $t = 0$ and $p = 0$), analogous to the state $s'$ in Figure B.1.
\( \varphi_{2}^{T,w} \) ensures that at \( t = j \) in the run, the symbol \( w_{j} \) is read for \( 0 \leq j \leq |w| - 1 \). And, \( \varphi_{9}^{T,w} \) ensures that the blank symbol is read at \( |w| \leq t \leq 2^{|w|} - 1 \). Next, we specify the incrementation of the time counter during the run:

\[
\varphi_{4}^{T,w} = \text{says}_{l(A)}(r \Rightarrow c_{+1}(t, A, |w| + 1))
\]

The position of the read head is specified as follows:

\[
\varphi_{5}^{T,w} = \text{says}_{l(A)}((r \land (t < 2^{|w|} - 1)) \Rightarrow c_{+1}(p, A, |w|))
\]

\[
\varphi_{6}^{T,w} = \text{says}_{l(A)}(r \land (t = 2^{|w|} - 1) \Rightarrow \mathcal{O}_{A} \text{says}_{l(A)}((p = 0) \land q_{0}))
\]

\[
\varphi_{7}^{T,w} = \text{says}_{l(A)}((r \land (t \geq 2^{|w|})) \land \neg d \land (p = 0)) \Rightarrow \mathcal{O}_{A} \text{says}_{l(A)}(p = 0))
\]

\[
\varphi_{8}^{T,w} = \text{says}_{l(A)}((r \land (t \geq 2^{|w|})) \land \neg d \land (p \neq 0)) \Rightarrow c_{-1}(p, A, |w|))
\]

\[
\varphi_{9}^{T,w} = \text{says}_{l(A)}((r \land (t \geq 2^{|w|})) \land d) \Rightarrow c_{+1}(p, A, |w|))
\]

\( \varphi_{5}^{T,w} \) ensures that the position moves to the right for \( 0 \leq t < 2^{|w|} - 1 \). At \( t = 2^{|w|} - 1 \), \( \varphi_{6}^{T,w} \) resets the read head position \( (p = 0) \) and state \( (q_{0}) \), so that the actual run can start at \( t = 2^{|w|} \). The atomic proposition \( d \) gives the direction in which the head will move (\( \neg d \) for left and \( d \) for right). \( \varphi_{7}^{T,w} \) ensures that moving to the left from position 0 leaves the position unchanged. \( \varphi_{8}^{T,w} \) and \( \varphi_{9}^{T,w} \) do subtraction (resp., addition) on moving to the left (resp., right).

We specify the reading and writing mechanism:

\[
\varphi_{9}^{T,w} = \text{says}_{l(A)}(r \Rightarrow c_{>}(t, B, |w| + 1))
\]

\[
\varphi_{10}^{T,w} = \text{says}_{l(A)}(r \Rightarrow \vee_{j=0}^{|w|-1} b(p_{j}, B) \land b(\neg p_{j}, B))
\]

\[
\text{tr}(q, a, b, v, q') = q \land r \land (d = v) \land \mathcal{O}_{l(A)} \text{says}_{l(A)} q' \land \mathcal{O}_{B} \text{says}_{l(B)} b
\]

\[
\text{notr} = \mathcal{O}_{A} \text{says}_{l(A)} \bot \land \mathcal{O}_{B} \text{says}_{l(B)} \bot
\]

\[
\varphi_{11}^{T,w} = \text{says}_{l(A)}(r \land (t \geq 2^{|w|}) \Rightarrow \bigvee_{(q, a, b, v, q') \in \delta} \text{tr}(q, a, b, v, q') \lor \text{notr})
\]

\( \varphi_{11}^{T,w} \) guesses the time at which the symbol currently written will be read. \( \varphi_{10}^{T,w} \) ensures that the position of the read head is the same as the current time. \( \varphi_{11}^{T,w} \) we specify that either (a) a transition is taken \( (\text{tr}(q, a, b, v, q')) \), or (b) the (final) state cannot be a \( C_{6} \) witness for another state \( (\text{notr}) \).

Finally, we require the uniqueness of state \( (\varphi_{12}^{T,w}) \) and alphabet \( (\varphi_{13}^{T,w}) \), and the presence of an accepting state \( (\varphi_{14}^{T,w}) \):
The conjunct \( \text{say}_{l(B)} \perp \) in \( \phi_{14}^{T,w} \) ensures that all states need to have \( \mathbf{C5} \) witnesses.

We can show the following:

**Proposition B.11.** Given \( T = (Q, \Sigma, \delta, q_0, F) \) and input \( w \), \( \phi^{T,w} = \bigwedge_{i=1}^{16} \phi_{i}^{T,w} \) is satisfiable iff \( T \) accepts \( w \) in at most \( 2^{|w|} - 1 \) steps.

The proof follows straightforwardly from the construction.

## B.5 Non-interference

We repeat the definition of reachable utterances, followed by a proof of the non-interference theorem.

**Definition 4.12 (Reachable Utterances).** Given a set of utterances \( U \) and a formula \( \text{say}_{id_B} \psi \), \( U^{*}_{id_B} \) is the smallest set such that:

- If \( id_B \in Id_B \) and \( \text{say}_{\{id_B\}} \varphi \in U \), \( \text{say}_{\{id_B\}} \varphi \in U^{*}_{id_B} \)
- If \( \text{say}_{\{id_B\}} \varphi \in U^{*}_{id_B} \) and \( \text{say}_{Id_A} \psi' \) is a subformula of \( \varphi \), then \( U^{*}_{id_A} \subseteq U^{*}_{id_B} \)

**Theorem 4.4 (Non-interference).** Given a set of utterances \( U \), for all \( \text{say}_{id_B} \psi \in L \), we have \( U \vdash \text{say}_{id_B} \psi \) iff \( U^{*}_{id_B} \vdash \text{say}_{id_B} \psi \)

**Proof.** One direction follows easily using propositional reasoning, i.e., if \( U^{*}_{id_B} \vdash \text{say}_{id_B} \psi \), then \( U \vdash \text{say}_{id_B} \psi \), since \( U^{*}_{id_B} \subseteq U \).

For the other direction, we proceed by contradiction. Suppose \( U \vdash \text{say}_{id_B} \psi \), and \( U^{*}_{id_B} \nvdash \text{say}_{id_B} \psi \). So, \( \phi = U^{*}_{id_B} \land \neg \text{say}_{id_B} \psi \) is satisfiable. Let \( M = (S, I_{\Phi_1}, \ldots, I_{\Phi_n}, \delta, \delta_O) \) be a model of \( \phi \). Hence:

- There exists \( s^\phi \in S \) such that \( (M, s^\phi) \models \phi \) for some \( s^\phi \in S \), and
• There exists $s^{-\psi} \in \delta_L(s, Id_B)$ such that $(M, s^{-\psi}) \models \neg \psi$

We will construct a new model $M'$ with a state $s^*$ such that $(M', s^*) \models \bigwedge U$ and $(M', s^*) \models \neg \text{says}_{Id_B} \psi$. This would contradict the assumption that $U \models \text{says}_{Id_B} \psi$. The main difficulty with the construction is that $\psi$ may have a subformula $\text{says}_{Id_C} \psi'$ and a statement $\text{says}_{Id_C} \varphi \in U - U'_{Id}$ such that $id_C \in Id_C$. Thus, changing the truth of $\text{says}_{Id_C} \varphi$ could result in a change in the truth of $\text{says}_{Id_C} \psi'$. Handling this case makes the construction involved.

We construct a new model $M' = (S', I_{\Phi_1}, ..., I_{\Phi_n}, \delta'_L, \delta'_O)$ as follows:

**The states $S'$**: For each state $s \in S$, we assign a new state, denoted $c(s)$, which is to be understood as a copy of $s$. We assume that $c(s) \notin S$ and $c(s) = c(s')$ iff $s = s'$. Given $S_1 \subseteq S$, $c(S_1)$ denotes the set of states such that $c(s) \in c(S_1)$ iff $s \in S_1$. In addition, we add two special states $s^*$ (at which the contradiction will be obtained) and $s^W$ (which provides witnesses as needed for $C5$ and $C6$). As a result:

$$S' = S \cup c(S) \cup \{s^*, s^W\}$$

**Interpretation of Predicates**: $I'_{\Phi_1}, ..., I'_{\Phi_n}$ is the same as $I_{\Phi_1}, ..., I_{\Phi_n}$ with the copies of states having the same assignment as the states in $S$. No predicates hold at $s^*$ and $s^W$.

**Accessibility Relation $\delta'_O$**: $\delta'_O$ respects $\delta_O$ for $s \in S$. $\delta'_O(c(s), A) = c(\delta_O(s, A))$, for all $A \in O_P$ and $c(s) \in c(S)$. In addition, $\delta_O(s^*, A) = \{s^*\}$, and $\delta_O(s^W, A) = \{s^W\}$, for all $A \in O_P$.

**Accessibility Relation $\delta'_L$**: This is the main part of the construction. $\delta'_L$ respects $\delta_L$ for $s \in S$. For all $A \in O_P$ and $Id_A \subseteq l(A)$, $\delta'_L(s^W, Id_A) = S'$. We now describe the construction for the other states, starting with some notation.

Given $Id_A \subseteq l(A)$, let $Id'_A$ be the set such that for all $id_A \in Id_A$, $id_A \in Id'_A$ iff $id_A \in Id_B$ or there exists a subformula $\text{says}_{Id'_A} \varphi \in U'_{Id_B}$ such that $id_A \in Id'_A$.

**The state $s^*$**- For all $A \in O$ and $Id_A \subseteq l(A)$, we have the following cases:

• If $Id'_A \neq Id_A$, then $\delta'_L(s^*, Id_A) = \emptyset$
\[ \text{•} \text{ Otherwise, } \delta'_L(s^*, Id_A) = \delta_L(s^\phi, Id_A) \cup c(\delta_L(s^\phi, Id_A)). \]

The first clause is used to ensure that \((M', s^*) \models \text{says}_{\{id_A\}} \varphi\) for all \(\text{says}_{\{id_A\}} \varphi \in U^{-1}_{id_B}\), since \(\{id_A\}^* = \emptyset\). The second clause adds both the states that are accessible from \(s^\phi\) and their copies. The accessibility relations associated with a copy \(c(s^{-\psi})\) will be modified in order to preserve C5.

The copies-For all \(C \in O, Id_C \subseteq l(C)\), and \(c(s) \in c(S)\):

\[ \text{•} \text{ If } c(s) \notin \delta'_O(c(s^{-\psi}), C) \text{ or } Id_A^* = Id_A, \text{ then } \delta'_L(c(s), Id_C) = c(\delta_L(s, Id_C)). \]

\[ \text{•} \text{ Otherwise, } \delta'_L(c(s), Id_C) = \delta_L(s^\phi, Id_C) \cup c(\delta_L(s^\phi, Id_C)) \cup c(\delta_L(s, Id_C)) \cup \{s^W\}. \]

Note that the second clause does not affect the truth of any subformula in \(U^*_{id_B}\), and it ensures that there are witnesses as needed for C5 for the cases where \(\delta'_L(c(s^\phi), Id_A) = \emptyset\).

Frame Constraints: We need to verify that the frame constraints hold in \(M'\). The only difficulty is in showing that C5 holds at the copies and \(s^*\). Fix \(A, C, Id_A\) and \(Id_C\). Given \(c(s) \in c(S)\), there are two cases:

\[ \text{•} \text{ If } c(s) \notin \delta'_O(c(s^{-\psi}), A) \text{ or } Id_A^* = Id_A, \text{ then } \delta'_L(c(s), Id_C) = c(\delta_L(s, Id_A)). \]

Consider \(c(s') \in \delta'_L(c(s), Id_A)\). Since C5 holds at \(s\) in \(M\):

\[ \text{•} \text{ if } s' \in \delta_L(s, Id_C), \text{ in which case } c(s') \in \delta'_L(c(s), Id_C), \text{ or} \]

\[ \text{•} \text{ There exists } s_1 \in \delta_L(s, Id_A) \text{ (resply, by construction, } c(s_1) \in \delta_L(c(s), Id_A)), \text{ such that for all } s_2 \in \delta_O(s_1, C) \text{ (resply, by construction, } c(s_2) \in \delta'_O(c(s_1), C)), \text{ we have } s' \in \delta_L(s_2, Id_C) \text{ (resply, by construction, } c(s') \in c(\delta_L(s_2, Id_C)) \subseteq \delta'_L(c(s_2), Id_C)). \]

\[ \text{•} \text{ if } c(s) \in \delta'_O(c(s^{-\psi}), A) \text{ and } Id_A^* \neq Id_A. \text{ By construction, } s^W \in \delta_L(c(s), Id_A), \text{ and } \delta'_O(s^W, C) = \{s^W\}. \text{ Since } \delta'_L(c(s), Id_A) \subseteq \delta'_L(s^W, Id_C) = S', \text{ C5 is trivially satisfied.} \]

Next we consider the state \(s^*\) for which there are three cases:
1. \( Id_A^* \neq Id_A \). \( \delta_L(s^*, Id_A) = \emptyset \) and \( \textbf{C5} \) is vacuously satisfied.

2. \( Id_A^* = Id_A \) and \( Id_C^* \neq Id_C \). For each \( c(s) \in \delta'_O(c(s^{-\psi}), C) \), we have \( \delta'_L(s^*, Id_A) \subseteq \delta'_L(c(s), Id_C) \), thereby satisfying \( \textbf{C5} \).

3. \( Id_A^* = Id_A \) and \( Id_C^* = Id_C \). In this case, \( \textbf{C5} \) is satisfied because \( \textbf{C5} \) holds in \( M \) and the copies of states are isomorphic.

**Establishing the contradiction:** The following are established easily by induction:

(P1) For all \( s \in S \) and \( \varphi \in L \), \( (M, s) \models \varphi \) iff \( (M', s) \models \varphi \).

(P2) For all \( s \in S \) and \( \varphi \in L \) such that for all subformulas \( \text{says}_{Id_A} \varphi' \) of \( \varphi \), \( Id_A^* = Id_A \), \( (M, s) \models \varphi \) iff \( (M', c(s)) \models \varphi \).

We can now reason as follows:

1. \( (M', s^*) \models \bigwedge U_{Id_B}^* \), since for all \( \text{says}_{id_A} \varphi \in U_{Id_B}^* \), for all \( s \in \delta'_L(s^\varphi, \{id_A\}) \), \( (M', s) \models \varphi \) (using (P1)), and for all \( c(s) \in \delta'_L(c(s^\varphi), \{id_A\}) \), \( (M', c(s)) \models \varphi \) (using (P2)).

2. \( (M', s^*) \models \text{says}_{id_A} \varphi \), for all \( \text{says}_{id_A} \varphi \in U - U_{Id_B}^* \) (by construction, since \( \{id_A\}^* \neq \{id_A\} \) and \( \delta'_L(s^*, \{id_A\}) = \emptyset \)).

3. Hence, \( (M', s^*) \models \bigwedge U \)

4. \( (M', s^*) \models \neg \text{says}_{Id_B} \psi \), since \( s^{-\psi} \in \delta_L(s^*, Id_B) \) and \( (M', s^{-\psi}) \models \neg \psi \) (using (P1)).

The last two items contradict the assumption that \( U \vdash \text{says}_{Id_B} \psi \).

\[ \square \]

**B.6 Conformance**

We repeat the definition of conformance, followed by a discussion of the proof of its decidability:
**Definition 4.13** (Conformance). Given a state $s$ with a set of objects $O$, a body of regulation $\text{Reg}$, and $\{A, B\} \subseteq O_P$, we say that $A$ conforms to $B$ w.r.t. the laws $\text{Id}_B \subseteq l(B)$ and a fixed point $(U, U')$ with $U = U'$ iff for all propositional $\varphi \in L_{\varphi_A}$:

If $(s, \text{Reg}) \models_{(U, U')} \text{Id}_A \varphi$, then $(s, \text{Reg}) \models_{(U, U')} \varphi$

We now discuss the proof of decidability of conformance. Given a state $S$ and a fixed point $(U, U')$, there are potentially infinitely many formulas $\varphi \in L_{\varphi_A}$ such that $S \models_{(U, U')} \text{Id}_A \varphi$. For example, if there is some $\varphi$ such that $S \models_{(U, U')} \text{Id}_A \varphi$, then for all $\varphi' \in L_{\varphi_A}$, we have $(s, \text{Reg}) \models_{(U, U')} \text{Id}_A (\varphi \lor \varphi')$.

We will prove that it suffices to restrict attention to a single formula, which may be understood as a prime implicant of all the obligations imposed on $A$ via the laws $\text{Id}_B$.

The proof relies on properties of the canonical model of a formula (Definition B.3).

We begin with some notation. Given $\phi \in L$, let $M_\phi = (S, I_{\phi_1}, ..., I_{\phi_n}, \delta_L, \delta_O)$ be the canonical model of $\phi$. Recall that each state $\Delta \in S$ is a maximal consistent set of subformulas of $\phi$, i.e., $\Delta \subseteq \text{sub}(\phi)$. Given $\Delta \in S$ and $\text{Id}_B \subseteq l(B)$, $\Delta_{\text{Id}_B}$ is the set such that $\varphi \in \Delta_{\text{Id}_B}$ iff there exists $\text{Id}_B' \subseteq \text{Id}_B$ such that $\Delta_{\text{Id}_B'} \subseteq \Delta_{\text{Id}_B}$ such that $\text{Id}_B' \varphi \in \Delta$. Similarly, given $\Delta \in S$ and $A \in O$, $\Delta_A$ is the set such that $\varphi \in \Delta_A$ iff $\text{Id}_B \varphi \in \Delta$.

We now establish some properties of maximal consistent sets that are useful in the proof.

**Proposition B.12.** Given $\phi \in L$, let $M_\phi = (S, I_{\phi_1}, ..., I_{\phi_n}, \delta_L, \delta_O)$ be the canonical model of $\phi$. The following hold for all $\varphi \in L_{\varphi_A}$, $\psi \in L_\psi$ and $\Delta \in S$:

1. If for all $\Delta' \in \delta_L(\Delta, A)$, $\Delta' \vdash \varphi$, then $\Delta \vdash \text{Id}_A \varphi$

2. If for all $\Delta' \in \delta_L(\Delta, \text{Id}_B)$, $\Delta' \vdash \psi$, then $\Delta \vdash \text{Id}_B \psi$

3. If $\Delta \vdash \text{Id}_B \varphi$, then for all $\Delta' \in \delta_L(\Delta, \text{Id}_B)$, $\Delta' \vdash \varphi$

**Proof.** For the first two items, we will need the following observation. Given $\Gamma \subseteq \text{sub}(\phi)$, let $S_\Gamma \subseteq S$ be the set such that $\Delta \in S_\Gamma$ iff $\Gamma \subseteq \Delta$. Then, for all $\varphi \in L$:

$(\ast)$ $\Gamma \vdash \varphi$ iff for all $\Delta \in S_\Gamma$, $\Delta \vdash \varphi$. 220
This follows using propositional reasoning, since $S$ is the set of all maximal consistent states w.r.t. $\text{sub}(\varphi)$, and $S_T$ is the set of all maximal consistent sets containing $\Gamma$.

**Item 1:** Consider $\varphi \in L_{\varphi,A}$ such that for all $\Delta' \in \delta'_L(\Delta, A)$, $\Delta' \vdash \varphi$. By construction, $\delta_L(\Delta, A) = S_{\Delta A}$, and by $(\ast)$, $\Delta_A \vdash \varphi$. Using R2, $\vdash O_A(\bigwedge \Delta_A \Rightarrow \varphi)$. Since $\Delta \vdash O_A(\bigwedge \Delta_A)$, using A2, $\Delta \vdash \varphi$.

The proof of item 2 is similar.

**Item 3:** We proceed by contradiction. Suppose there exists $\varphi \in L_{\varphi,A}$ and $\Delta \in S$ such that $\Delta \vdash \text{Id}_B O_A \varphi$, and $\Delta' \not\vdash \varphi$ for some $\Delta' \in \delta_L(\Delta, Id_B)$. So, there exists a model $M' = (S', I_{\Phi_1}', ..., I_{\Phi_n}', \delta'_L, \delta'_O)$ and $s^\varphi \in S'$ such that $(M', s^\varphi) \models \bigwedge \Delta'$ and $(M', s^\neg \varphi) \models \neg \varphi$. We construct a new model $M'' = (S'', I_{\Phi_1}'', ..., I_{\Phi_n}'', \delta''_L, \delta''_O)$ combining $M_\phi$ and $M'$ as follows:

- $S'' = S \cup S''$. We assume that $S$ and $S''$ are disjoint.
- The interpretation of predicates respects those in $M_\phi$ and $M'$
- $\delta''_L$ respects the accessibility relations $\delta_L$ and $\delta'_L$
- $\delta''_O$ respects the accessibility relations $\delta_O$ and $\delta'_O$, except that:

$$\delta''_O(\Delta', A) = \delta_O(\Delta', A) \cup \{s^\neg \varphi\}$$

The satisfaction of the constraints C3-C6 is immediate from the construction, as the only modification is to $\delta''_O(\Delta', A)$. The following are established easily by induction:

1. For all $s \in S'$, $(M'', s) \models \psi$ iff $(M', s) \models \psi$
2. For all $\Delta \in S$ and $\psi \in \text{sub}(\bigwedge U)$, $(M'', \Delta) \models \psi$ iff $\psi \in \Delta$

We can now reason as follows:

3. $(M'', \Delta) \models \bigwedge \Delta$ (using (2))
4. $(M'', s^\neg \varphi) \not\models \varphi$ (using (1))
(5) \((M'', \Delta') \not\models O_A \varphi\) (from (4) since \(s^\varphi \in \delta''_O(\Delta', A)\))

(6) \((M'', \Delta) \not\models \text{says}_{Id_B} O_A \varphi\) (from (5) since \(\Delta' \in \delta''_L(\Delta, Id_B)\))

(7) \(\Delta \not\models \text{says}_{Id_B} O_A \varphi\) (from (3) and (6), by soundness)

Item (7) contradicts the assumption that \(\Delta \models \text{says}_{Id_B} O_A \varphi\).

We are now ready to show that conformance checking is decidable:

**Theorem 4.5** (Decidability of Conformance). Given a state \(S\), a body of regulation \(\text{Reg}\), a fixed point \((U, U')\) where \(U = U'\) and \(|U|\) is finite, principals \(\{A, B\} \subseteq O\), and identifiers \(Id_B \subseteq l(B)\), there is a procedure to decide whether \(A\) conforms to \(B\) w.r.t. the laws \(Id_B\).

**Proof.** First, we observe that for all \(\varphi \in L_{\varphi_A}\), \((s, \text{Reg}) \models_{(U, U')} \text{says}_{Id_B} O_A \varphi\) iff \(U \models \text{says}_{Id_B} O_A \varphi\) (by definition). So, it suffices to check that for all \(\varphi \in L_{\varphi_A}\), if \(U \models \text{says}_{Id_B} O_A \varphi\), then \((s, \text{Reg}) \models_{(U, U')} \varphi\).

The key idea is to show that there is a formula \(\varphi_U \in L_{\varphi_A}\) such that:

(P1) \(U \models \text{says}_{Id_B} O_A \varphi_U\), and

(P2) For all \(\varphi \in L_{\varphi_A}\) such that \(U \models \text{says}_{Id_B} O_A \varphi\), we have \(\models \varphi_U \Rightarrow \varphi\).

Assuming that such a \(\varphi_U\) exists, we can show the following:

- \(A\) conforms to \(B\) w.r.t. \(Id_B\) iff \((s, \text{Reg}) \models_{(U, U')} \varphi_U\).

If \(A\) conforms to \(B\) w.r.t. \(Id_B\), since \(U \models \text{says}_{Id_B} O_A \varphi_U\), we have \((s, \text{Reg}) \models_{(U, U')} \varphi_U\). For the other direction, we need the observation that for all \(\phi \in L_{\varphi_A}\), if \(\models \phi\), then \((s, \text{Reg}) \models_{(U, U')} \phi\). Note that this claim does not hold when \(U \neq U'\). When \(U = U'\), the claim is easily verified by showing that the axioms \(A1-A3, A5,\) and \(A6\) are valid at \(s\) w.r.t. \((U, U')\), and that the rules \(R1\) and \(R2\) preserved validity. Instances of axiom schema \(A4\) are not in \(L_{\varphi_A}\).
Now suppose that \((s, \text{Reg}) \models_{(U,U')} \varphi_U\). For all \(\varphi \in L_{\varphi_A}\) such that \(U \vdash \text{says}_{Id_B} O_A \varphi\), we have \((s, \text{Reg}) \models_{(U,U')} \varphi_U \Rightarrow \varphi\) (using (B)). If \((s, \text{Reg}) \models_{(U,U')} \varphi_U\), then it follows that \((s, \text{Reg}) \models_{(U,U')} \varphi_U\). Thus, if \((s, \text{Reg}) \models_{(U,U')} \varphi_U\), then \(A\) conforms to \(B\) w.r.t. \(Id_B\). Since checking whether \((s, \text{Reg}) \models_{(U,U')} \varphi_U\) is decidable, conformance checking is decidable, provided that such a \(\varphi_U\) exists.

We now turn to the construction of \(\varphi_U\). Let \(M_U = (S, I_{\Phi_1}, \ldots, I_{\Phi_n}, \delta_L, \delta_O)\) be the canonical model for \(\bigwedge U\). Let \(S_U = \{\Delta_1 | \Delta_1 \in S and U \subseteq \Delta_1\}\). We will now define a formula \(\varphi_\Delta\) for each \(\Delta \in S_U\), and define \(\varphi_U\) as their disjunction:

\[
\varphi_\Delta = \bigvee_{\Delta' \in \delta_L(\Delta, Id_B)} \bigwedge_{A} \varphi_\Delta \\
\varphi_U = \bigvee_{\Delta \in S_U} \varphi_\Delta
\]

We claim the following for all \(\Delta \in S\):

(P3) \(\Delta \vdash \text{says}_{Id_B} O_A \varphi_\Delta\)

(P4) For all \(\varphi \in L\), if \(\Delta \vdash \text{says}_{Id_B} O_A \varphi\), then \(\vdash \varphi_\Delta \Rightarrow \varphi\)

**Proof of (P3):** Using propositional reasoning, for all \(\Delta' \in \delta_L(\Delta, Id_B)\), for all \(\Delta'' \in \delta_O(\Delta', A)\), \(\Delta'' \vdash \varphi_\Delta\). Hence, for all \(\Delta' \in \delta_L(\Delta, Id_B)\), by Proposition B.12 Item 1, we have \(\Delta' \vdash O_A \varphi_\Delta\). And using, Proposition B.12 Item 2, \(\Delta \vdash \text{says}_{Id_B} O_A \varphi_\Delta\).

**Proof of (P4):** Suppose \(\Delta \vdash \text{says}_{Id_B} O_A \varphi\). By Proposition B.12 Item 3, for all \(\Delta' \in \delta_L(\Delta, Id_B)\), \(\Delta'_A \vdash \varphi\). And, using propositional reasoning, \(\vdash \varphi_\Delta \Rightarrow \varphi\)

**Proof of (P1):** Using (P3), for all \(\Delta \in S_U\), we have \(\Delta \vdash \text{says}_{Id_B} O_A \varphi_\Delta\). And, by propositional reasoning, \(U \vdash \text{says}_{Id_B} O_A \varphi_\Delta\) (since \(S_U\) is the set of all maximal consistent sets containing \(U\)).

**Proof of (P2):** Using (P4), for all \(\varphi \in L\), if \(U \vdash \text{says}_{Id_B} O_A \varphi\), then \(\Delta \vdash \text{says}_{Id_B} O_A \varphi\) for all \(\Delta \in S_U\). Hence, \(\vdash \varphi_\Delta \Rightarrow \varphi\) for all \(\Delta \in S_U\), and by propositional reasoning, \(\vdash \varphi_U \Rightarrow \varphi\). 

\[\square\]
B.7 Completeness of the Decision Procedure for Chain Formulas

To establish completeness, it suffices to show the following:

Lemma B.3 (Completeness). Given a closed set $\Delta$ and a formula $\psi$, if $\Delta \vdash \psi$, then $\Delta \vDash \psi$.

The rest of this section gives the proof. As before, we will show the contrapositive, i.e., if $\Delta \nvdash \psi$, then $\Delta \not\vDash \psi$. We begin with some notation. Given a set of formulas $\Delta$, let $\Delta_{l(A)}$ be the set such that $\phi \in \Delta_{l(A)}$ iff $\text{says}_{l(A)} \phi \in \Delta$. Similarly, let $\Delta_A$ be the set such that $\phi \in \Delta_A$ iff $\text{O}_A \phi \in \Delta$. For all $P_A \phi \in \Delta$, let $\Delta^\phi_A = \Delta_A \cup \{ \phi \}$. And, finally, $\mathcal{X}(\Delta, A)$ is the class of sets such that $\Delta' \in \mathcal{X}(\Delta, A)$ iff $\Delta' = \Delta_A$ or $\Delta' = \Delta^\phi_A$ for some $P_A \phi \in \Delta$.

We mention some properties of closed sets, which are useful in establishing inductive properties:

Proposition B.13. Given a closed set $\Delta$:

- $\Delta_{l(A)}$ is closed.
- For all $\Delta' \in \mathcal{X}(\Delta, A)$, $\Delta'$ is closed.

Proof. $\Delta_{l(A)}$ contains only formulas that are atomic, $\bot$, or of the form $O_B \phi$. Similarly, for all $\Delta' \in \mathcal{X}(\Delta, A)$, $\Delta'$ contains only formulas that are atomic, $\bot$, or of the form $\text{says}_{l(A)} \phi$. In both cases Definition 4.16 is vacuously satisfied. □

Completeness is a consequence of the following proposition:

Proposition B.14. Given a closed set $\Delta$, if $\Delta \not\vdash \psi$, there exists a model $M = (S, \pi, \delta_C, \delta_O)$ and state $\Delta \in S$ such that:

$$ (M, \Delta) \models \bigwedge\Delta \text{ and } (M, \Delta) \not\models \psi $$

We say that $B$ is permitted to speak in $\Delta$, denoted $\text{pspeak}(B, \Delta)$, if for all $\mathcal{X}_B \text{ says}_{l(B)} \phi \in \Delta$, we have $\phi \in \Delta$. We require that for all $B \in O_P$:
M1 $\Delta \notin \delta_O(\Delta, B)$

M2 If $\text{pspeak}(B, \Delta)$, then for all $s' \in \delta_O(\Delta, B)$ and $C \in O_P$, $\Delta \in \delta_L(s', l(C))$.

Proof. The proof proceeds by induction on the structure of $\psi$.

Case 1: If $\psi = q$ is atomic (or false), we construct $M = (S, \pi, \delta_L, \delta_O)$ as follows:

- $S = \{\Delta\} \cup \{A \in O_P, X(\Delta, A)\}$

- For all atomic propositions $p \in \Phi$, $\Delta \in \pi(p)$ iff $p \in \Delta$. In addition, for all $p \in \Phi$, $A \in O_P$ and $\Delta' \in X(\Delta, A)$, we have $\Delta' \in \pi(p)$ iff $p \in \Delta'$.

- For all $B \in O_P$, $\delta_L(\Delta, l(B)) = \emptyset$. For all $B \in O_P$, $\Delta' \in X(\Delta, B)$:
  - If $\text{pspeak}(B, \Delta)$, then $\delta_L(\Delta', l(C)) = \{\Delta\}$ (for all $C \in O_P$)
  - Otherwise, $\delta_L(\Delta', l(C)) = \emptyset$ (for all $C \in O_P$)

- For all $B \in O_P$, $\delta_O(\Delta, B) = X(\Delta, B)$. For all $B \in O_P$ and $\Delta' \in X(\Delta, B)$, we have $\delta_O(\Delta', C) = S$ (for all $C \in O_P$).

It is easy to show that $(M, \Delta) \models \bigwedge \Delta$ and $(M, \Delta) \not\models q$. It may seem that the construction is complex for the satisfaction of an atomic formula. However, this is needed to ensure the additional properties M1 and M2. These properties are crucial to satisfy the constraints C5 and C6 when $\psi = \text{says}_{l(A)}(A)$ (case 4 below).

Case 2: $\psi = P_A \psi'$. Let $X(\Delta, A) = \{\Delta_1, \ldots, \Delta_n\}$. Since $\Delta \not\models \psi$, we have $\Delta_i \not\models \psi'$ for all $1 \leq i \leq n$. In addition, since $\Delta_i$ is closed, by induction, there exists a model $M^i = (S^i, \pi^i, \delta_L^i, \delta_O^i)$ such that $(M, \Delta_i) \models \bigwedge \Delta_i$ and $(M, \Delta_i) \not\models \psi'$. We construct a new model $M' = (S', \pi', \delta_L', \delta_O')$ as follows:

- $S' = \bigcup_{1 \leq i \leq n} S^i \cup \{\Delta\} \cup \{\Delta_B | B \in O_P, B \neq A\}$.

- $\pi'$ respects the assignment in each of the models. For all $p \in \Phi$, $\Delta \in \pi(p)$ iff $p \in \Delta$. And, for all $p \in \Phi$ and $B \in O_P$ such that $B \neq A$, we have $\Delta_B \in \pi(p)$ iff $p \in \Delta'$.
δ′_L respects the relation in each of the models, except for ∆_i \in S^i (1 \leq i \leq n), for which we proceed as follows:

- If pspeak(A, ∆), then δ′_L(∆_i, l(C)) = δ_L(∆_i, l(C)) ∪ {∆} (for all C \in O_P)
- Otherwise, δ′_L(∆_i, l(C)) = δ_L(∆_i, l(C)) (for all C \in O_P)

For the other states we proceed as in Case 1. For all B \in O_P, δ′_L(∆, l(B)) = ∅. For all B \in O_P such that B \neq A, ∆' \in X(∆, B):

- If pspeak(B, ∆), then δ′_L(∆', l(C)) = {∆} (for all C \in O_P)
- Otherwise, δ′_L(∆', l(C)) = ∅ (for all C \in O_P)

δ′_O respects the relation in each of the models. For all B \in O_P, δ′_O(∆, B) = X(∆, B). For all B \in O_P such that B \neq A and ∆' \in X(∆, B), we have δ′_O(∆', C) = S' (for all C \in O_P).

The following can be shown by simultaneous induction on the structure of formulas:

- For all ϕ \in ∆, (M', ∆) \models ϕ.
- For all B \in O_P, ∆' \in X(∆, B) and ϕ \in ∆', (M, ∆') \models ϕ.
- For all s^i \in S^i and all chain formulas φ, if (M^i, s^i) \not\models φ, then (M', s^i) \not\models φ.

It follows that (M', ∆) \models \bigwedge ∆. In addition, (M', ∆) \models O_A \neg ψ' since for all ∆' \in δ′_O(∆, A), we have (M', ∆') \not\models ψ'. Hence, (M', ∆) \not\models P_A ψ'.

**Case 3:** The case for ψ = O_A ψ' is similar to the one above, and we leave the details to the reader.

**Case 4:** The final and most difficult case is for ψ = says_(A_i) ψ'. We begin with some notation, followed by an example.

Let A = \{A_1, ..., A_n\} be the set of principals such that A_i \in A iff A_i = A or there is some subformula says_(A_i) \varphi \in ∆. A^1 = A and A^n = A^{n-1} \cup (A \times A^{n-1}) for n ≥ 2, where A^n is the set of chains of principals of length less than or equal to n. The set of chains A^+ = \bigcup_{i\geq1} A^i.
Given a chain \((A_n, ..., A_1) \in A^+\), we define the schema \(\phi(A_n...A_1)\) as follows:

\[
\phi(A_n...A_1) = \mathcal{P}_{A_n} \text{says}_{l(A_n)} ... \mathcal{P}_{A_1} \text{says}_{l(A_1)} \phi
\]

Note that \(\phi(A) = \mathcal{P}_A \text{says}_{l(A)} \phi\). The set of chains of \(\phi\) is given by:

\[
C(\phi) = \{\phi\} \bigcup_{\sigma \in A^+} \phi(\sigma)
\]

And, let \(L_\psi(\Delta)\) be set of all \(\psi' \in L_\psi\) such that \(\psi'\) is a subformula in \(\Delta\).

\[\text{Figure B.2: Model to show that } \{\text{says}_{l(A_4)} p(A_3, A_2, A_1), \text{says}_{l(A_3)} p(A_1), \text{says}_{l(A_1)} p\} \not\models \text{says}_{l(A_4)} p. \text{ The states } s_1, s_2 \text{ and } s_3 \text{ demonstrate the satisfaction of } C5 \text{ and } C6.\]

**Example:** We now discuss an example to provide intuition for the construction. Let \(\Delta\) consist of the following formulas:

- \(\text{says}_{l(A_4)} p(A_3, A_2, A_1)\)
- \(\text{says}_{l(A_3)} p(A_1)\)
- \(\text{says}_{l(A_1)} p\)
Suppose we wish to show that $\Delta \nvdash \text{say}_{(A_4)} p$. The closure of $\Delta$ is given by $\Delta^* = \Delta \cup \{\text{say}_{(A_3)} p\}$. We need to construct a model $M$ with state $s$ such that $(M, s) \models \bigwedge \Delta^*$ and $(M, s) \nvdash \text{say}_{(A_4)} p$. Hence, there needs to be a state $s_1 \in \delta_L(s, l(A_4))$ such that:

- $(M, s_1) \models p(A_3, A_2, A_1)$, and
- $(M, s_1) \nvdash p$

Figure B.2 shows such a model with a state $s_1$. The difficulty comes in enforcing the constraints $C_5$ and $C_6$. Intuitively, the constraint $C_5$ says that if $A$ permits $B$ to say $p$ and $A$ does not say $p$, then $A$ needs to demonstrate this by taking $B$’s statements as her own, and showing that $p$ does not follow. On the other hand, if the permission to $B$ does not entail $p$, then it suffices for $A$ to say what it is permitted to say. Thus, we unroll the chain $p(A_3, A_2, A_1)$ until a further unrolling would entail $p$. This results in:

- $(M, s_1) \models p(A_2, A_1)$, and
- $(M, s_1) \nvdash p(A_1)$

Unrolling $p(A_1)$ further would result in $p$, giving us a contradiction. The process of unrolling results in, for example, the loop from $s_1$ to $s' \in \delta_O(s_1, A_3)$ and back to $s_1 \in \delta_L(s', l(A_3))$. However, since $p(A_1)$ is not unrolled, there is no loop from $s_1$ to $s' \in \delta_O(s_1, A_1)$ and back to $s_1$. As a result, to satisfy $C_5$, we need to generate a sibling which permits $s_1$. For this we construct the state $s_2$ (in Figure B.2) such that:

- $(M, s_2) \models p(A_3, A_2, A_1)$, and
- $(M, s_2) \nvdash p(A_1)$

Thus we can unroll $p(A_3, A_2, A_1)$ just once, so that $(M, s_2) \models p(A_2, A_1)$, but no further. Hence, there is no loop from $s_2$ to $s' \in \delta_O(s_2, A_2)$ and back to $s_2$. However, there is a connection from $s_2$ to $s' \in \delta_O(s_2, A_1)$ to $s_1$, and thus $s_2$ permits $s_1$. Similarly, we need to generate a state $s_3$ which permits $s_2$. 

228
For a closed set $\Delta$ such that $\bot$ does not appear as a subformula, it can be shown that the number of witnesses (e.g., the states $s_2$ and $s_3$ in Figure B.2) needed are bounded by the length of the longest delegation chain. The presence of $\bot$ makes matters more difficult. For this case, we have not obtained a bound on the number of witnesses needed. We do not know if it is polynomial in the size of $\Delta$. The construction that we give below generates a potentially infinite number of witnesses. We are guaranteed (by decidability of the full logic) that there exists a finite model, but it may be exponential in the size of $\Delta$.

**Witnesses:** Given a closed set $\Delta$ and $\phi \in L_\psi$, a *witness* is a tuple $w = (\phi', B, \Delta', \lambda')$, where $\phi \in C(\phi)$, $B \in O_p$, $\Delta' \subseteq L_\psi(\Delta)$, and $\lambda' \subseteq O_p$, with the following restrictions. $\lambda'$ is the smallest set such that:

- $B \in \lambda'$
- If $C \in \lambda'$, $\Delta \triangleright \text{says}_{l(C)} \phi'(D)$, then $D \in \lambda'$

$\Delta'$ is defined as follows:

- If $\Delta \triangleright \text{says}_{l(C)} \phi'$ for some $C \in \lambda'$, then $\Delta' = \emptyset$.
- Otherwise, $\Delta' = \{ \psi'| \psi' \in L_\psi(\Delta) \text{ and } \psi' \not\triangleright \phi' \}$.

It is immediate that $\Delta'$ is closed. We restrict attention to the set of witnesses where $\Delta' \neq \emptyset$:

$$W(\phi) = \{ w | w = (\phi', B, \Delta', \lambda') \text{ and } \Delta' \neq \emptyset \}$$

We begin by establishing some properties of witnesses. For all $w \in W(\phi)$, such that $w = (\phi', D, \Delta', \lambda')$:

- **W1** For all $B \in \lambda'$, $\Delta_{l(B)} \subseteq \Delta'$
- **W2** For all $B \in O_P$, $\text{pspeak}(B, \Delta')$ iff $\psi \not\triangleright \phi'$ for all $X_B \text{says}_{l(B)} \psi \in \Delta'$.
- **W3** For all $B \not\in \lambda'$, if it is not the case that $\text{pspeak}(B, \Delta')$, then for all $C \in \lambda'$, there exists $w_1 \in W(\phi)$ such that:
W3.1 \( w_1 = (\phi'(B), C, \Delta'', \lambda'') \)

W3.2 For all \( A \in \lambda' \), if it is not the case that \( \phi'(B) \), then there exists \( w_1 \in \mathcal{W}(\phi) \) such that:

W4 For all \( B \in \lambda' \), if it is not the case that \( \phi'(B) \), then there exists \( w_1 \in \mathcal{W}(\phi) \) such that:

W4.1 \( w_1 = (\phi'(B), B, \Delta'', \lambda'') \)

W4.2 For all \( A \in \lambda' \), if it is not the case that \( \phi'(B) \), then there exists \( w_1 \in \mathcal{W}(\phi) \) such that:

W1 and W2 follow easily from the definition of witnesses. For W3, we proceed by contradiction. Suppose there is no \( w_1 \in \mathcal{W}(\phi) \) such that W3.1 and W3.2. Let \( \lambda'' \) be the smallest set such that:

- \( C \in \lambda'' \)
- If \( E \in \lambda'', \Delta \rhd \text{says}_{\text{s}(E)} \phi'(F, B) \), then \( F \in \lambda'' \)

It follows that \( \Delta \rhd \text{says}_{\text{s}(E)} \phi'(B) \) for some \( E \in \lambda' \) (otherwise \( w_1 \in \mathcal{W}(\phi) \) giving us a contradiction). Since \( \Delta \) is closed, a simple induction can be used to establish that \( \Delta \rhd \text{says}_{\text{s}(C)} \phi'(B) \). This suffices to conclude that \( B \in \lambda' \) giving us a contradiction. The proof of W4 is similar.

Construction: We are now ready to construct the model. We are given a closed set \( \Delta \) and \( \text{says}_{\text{s}(A)} \phi \) such that \( \Delta \nvdash \text{says}_{\text{s}(A)} \phi \). Let \( w^\phi = (\phi, A, \Delta^\phi, \lambda^\phi) \). Since \( \Delta^\phi \) is closed, by induction, there is a model \( M^{w^\phi} = (S^{w^\phi}, \pi^{w^\phi}, \delta^{w^\phi}_L, \delta^{w^\phi}_O) \) such that \( (M^{w^\phi}, w^\phi) \models \bigwedge \Delta^\phi \) and \( (M^{w^\phi}, w^\phi) \not\models \phi \).

Note that the induction hypothesis cannot be applied to longer chains. However, for all \( w \in \mathcal{W}(\phi) \) such that \( w = (\phi', B, \Delta', \lambda') \), we have \( \Delta' \not\models \perp \). Hence, by induction, we can construct \( M^w = (S^w, \pi^w, \delta^w_L, \delta^w_O) \) such that \( (M^w, w) \models \bigwedge \Delta' \).

We construct a new model \( M' = (S', \pi', \delta'_L, \delta'_O) \) as follows:

- \( S' = S^{w^\phi} \cup_{w \in \mathcal{W}(\phi) - \{w^\phi\}} S^w \cup \{\Delta\} \cup_{A \in O_P} \mathcal{X}(\Delta, A) \)
• $\pi'$ respects the assignment in each of the models. For all atomic propositions $p \in \Phi$, $\Delta \in \pi(p)$ iff $p \in \Delta$. In addition, for all $p \in \Phi$, $A \in O_P$ and $\Delta' \in \mathcal{X}(\Delta, A)$, we have $\Delta' \in \pi(p)$ iff $p \in \Delta'$.

• $\delta'_L$ respects the relation in each of the models, except for the following cases. Let $w = (\phi', C, \Delta', \lambda') \in \mathcal{W}(\phi) - w\phi$. For all $s \in \delta''_L(w, B)$ and $E \in O_P$, we have $s' \in \delta'_L(s, l(E))$ iff:

- $s' \in \delta''_L(s, l(E))$ or
- $s' = w_1$ for some $w_1 = (\phi'', D, \Delta'', \lambda'') \in \mathcal{W}(\phi)$ such that:
  * $\phi' = \phi''(B)$
  * It is not the case that $\text{pspeak}(B, \Delta'')$, and
  * $C \in \lambda''$

For the remaining states, we proceed as follows. For all $w \in \mathcal{W}(\phi)$, $B \in O_P$, $w \in \delta'_L(\Delta, l(B))$ iff $w = (\phi', C, \Delta', \lambda')$ and $B \in \lambda'$. For all $B \in O_P$ such that $B \neq A$, $\Delta' \in \mathcal{X}(\Delta, B)$:

- If $\text{pspeak}(B, \Delta)$, then $\delta'_L(\Delta', l(C)) = \{\Delta\}$ (for all $C \in O_P$)
- Otherwise, $\delta'_L(\Delta', l(C)) = \emptyset$ (for all $C \in O_P$)

• $\delta'_O$ respects the relation in each of the models. For all $B \in O_P$, $\delta'_O(\Delta, B) = \mathcal{X}(\Delta, B)$. For all $B \in O_P$ such that $B \neq A$ and $\Delta' \in \mathcal{X}(\Delta, B)$, we have $\delta'_O(\Delta', C) = S$ (for all $C \in O_P$).

We need to verify that the frame constraints hold in $M'$. The only non-trivial case is to show that $\textbf{C5}$ and $\textbf{C6}$ hold at $\Delta$. Consider $w = (\phi', D, \Delta', \lambda') \in \mathcal{W}(\phi)$ with $C \in \lambda'$. By construction, $w \in \delta'_L(\Delta, l(C))$. For all $B \in O_P$, there are three cases:

• If $B \in \lambda'$, then $w \in \delta'_L(\Delta, l(C))$ and $\textbf{C5}$ is satisfied.

• If $\text{pspeak}(B, \Delta')$, then $\textbf{C5}$ is satisfied because M2 holds at $w$ in $M^w$. 231
• Otherwise, we proceed as follows. By W3, there exists \( w_1 = (\phi'(B), C, \Delta'', \lambda'') \in W(\phi) \) which satisfies W3.1 and W3.2. Since \( C \in \lambda'' \), by construction, we have \( w_1 \in \delta_L'(\Delta, l(C)) \). In addition, by construction, for all \( s \in \delta^\prime_\phi(w_1, B) \), \( w \in \delta^\prime_L(s, l(B)) \). Hence, C5 is satisfied.

The proof of C6 is essentially identical, except that we use W4 instead of W3 in the last case. A straightforward inductive argument can be used to show that \( (M', \Delta) \models \land \Delta \) and \( (M', \Delta) \not\models \text{says}_{l(A)} \phi \). We omit the details. \qed
Appendix C

Computing ASTs: Proof

In this section, we discuss the NP-hardness of ordering or ranking siblings during the computation of the AST. We assume the notation developed in Section 5.5.1. The proof relies on the NP-completeness of the **Acyclic Subgraph** problem:

**Theorem C.1** (Karp [80]). The following problem is NP-complete:

**Input:** A graph $G = (V, E)$ and a constant $k$

**Output:** Yes, if there exists $E' \subseteq E$ such that $|E'| \geq k$ and $G' = (V, E')$ is acyclic

We now show that the decision version of the ranking problem is NP-complete:

**Theorem 5.1.** The following problem is NP-complete:

**Input:** A PPT $\tau$, a preterminal $p \in \tau$, probabilities $P(o^i \gg o^j|\tau)$, and $c \in [0, 1]$

**Output:** Yes, if there is an ordering $v$ such that $P(v(p)|\tau) \geq c$

**Proof.** The proof proceeds by reduction from **Acyclic Subgraph**. We are given a graph $G = (V, E)$ and a constant $k$. We construct a ppt $\tau$ with a single preterminal $p = V$, i.e., the set of operators are the vertices in the graph. The probabilities are defined as follows:

$$
P(u \gg v|\tau) = \begin{cases} 
0.5 & \text{if } (u, v) \in E, (v, u) \in E \\
0.6 & \text{if } (u, v) \in E, (v, u) \notin E \\
0.4 & \text{if } (u, v) \notin E, (v, u) \in E \\
0.5 & \text{otherwise}
\end{cases}
$$
We will now build up to the definition of the constant $c$. Let $E_1 \subseteq E$ be a set such that $(u,v) \in E_1$ iff $(u,v) \in E$ and $(v,u) \in E$. $E_1$ is symmetric, i.e., $(u,v) \in E_1$ iff $(v,u) \in E_1$. Suppose $G' = (V,E')$ is an acyclic subgraph of $G$. We claim that there is an acyclic subgraph $G'' = (V,E'')$ such that $E' \subseteq E''$ and for each edge $(u,v) \in E_1$, either $(u,v) \in E''$ or $(v,u) \in E''$. The construction is straightforward. Given $G' = (V,E')$, let $(v_1,\ldots,v_{|V|})$ be a topological sort of $G$. We construct $E''$ such that $(v_i,v_j) \in E''$ iff $(v_i,v_j) \in E'$ or $(v_i,v_j) \in E_1$ and $i < j$.

Thus, without loss of generality, we can consider acyclic subgraphs $G' = (V,E')$ such that $|E' \cap E_1| = \frac{|E_1|}{2}$. In other words, subgraphs which contain $(u,v) \in E'$ or $(v,u) \in E'$, for each edge $(u,v) \in E_1$. The pairwise orderings imposed by a ranking will correspond to the following – (1) the number of edges in $E' \cap E_1$, (2) the number of edges in $E' - E_1$, and (3) the number of non-edges (corresponding to the last case in the definition of the probabilities). We define three constants $k_1$, $k_2$, and $k_3$ for each of these cases:

$$k_1 = \frac{|E_1|}{2}$$
$$k_2 = k - k_1$$
$$k_3 = \binom{|V|}{2} - k$$

Note that $|E'| = k_1 + k_2 = k$. Finally, we define the constant $c$:

$$c = 0.5^{k_1} \times 0.6^{k_2} \times 0.5^{k_3}$$

We claim that $G$ has an acyclic subgraph of size $k$ iff there is a ranking $\tau$ such that $P(\tau(p) | \tau) \geq c$. Suppose $G' = (V,E')$ is an acyclic subgraph of $G$, and $|E'| \geq k$. Let $(v_1,\ldots,v_{|V|})$ be a topological sort of $G'$. The ranking $\tau$ which respects the topological sort order has probability $c$. Next, suppose $(v_1,\ldots,v_{|V|})$ is a ranking with probability $c$. We define the set $E' \subseteq E$ such that $(v_i,v_j) \in E'$ iff $i < j$, i.e., only edges from a vertex of lower rank to higher rank. It is immediate from the definition of $c$ that $|E'| = k$, i.e., $k_1$ edges with probability 0.5 and $k_2 = (k - k_1)$ edges with probability 0.6. □
Bibliography


236


ing of the Association for Computational Linguistics (ACL), pages 115–124, 2005.


