Topic:

Towards Automated Ethical Reasoning: Representing the AI Act in Higher-Order-Logic

Master's thesis

in Computing in the Humanities at the Faculty of Information Systems and Applied Computer Sciences at the Otto-Friedrichs-University Bamberg

By: Lara Lawniczak
Supervisor: Prof. Dr. Christoph Benzmüller and Prof Dr. Andrea Vestrucci, Chair for AI Systems Engineering
Deadline for Submission: 29.07.2024

URN: urn:nbn:de:bvb:473-irb-108631x
DOI: https://doi.org/10.20378/irb-108631

Bamberg 2025

Abstract

This work investigates the feasibility and methodology of representing the European Union's AI Act in Higher-Order Logic to facilitate automated ethical reasoning. The AI Act and its modalities are thoroughly analyzed, and existing embeddings of Standard Deontic Logic and Dyadic Deontic Logic are employed to represent a selection of these modalities, specifically Obligations and Contrary-to-Duty Obligations, in Isabelle/HOL. To capture Agency and Agentive Obligations, embeddings of Temporal Deontic STIT logic and variants of DDL are introduced. The effectiveness and limitations of different embeddings are evaluated by formalizing example sections of the AI Act in the embedded logics and utilizing the automated theorem provers, satisfiability modulo theories solvers, and model finders available in Isabelle/HOL. The results provide valuable insights into the challenges of automated legal reasoning in HOL in a complex context, highlighting the strengths and limitations of current theorem-proving tools.

Keywords: Legal Reasoning, Deontic Logic, STIT Logic, Higher-Order Logic, Semantical Embeddings, Automated Theorem-Proving

Contents

List of Figures

List of Abbreviations

AIA AI Act. 1

Akoma Ntoso Architecture for Knowledge-Oriented Management of African Normative Texts using Open Standards and Ontologies. 11

ASP Answer Set Programming. 12

ATP Automated Theorem Provers. 2

BT+AC Branching Time and Agent Choices. 49

CTD Contrary-to-Duty Obligations. 20

DDL Dyadic Deontic Logic. 7, 21, 52

EU European Union. 1

HOL Higher Order Logic. 2, 5, 7

IM it is impermissible that. 27

ITPs Interactive Theorem Provers. 7

LogiKEy Logic and Knowledge Engineering Framework and Methodology. 2, 5

NO it is non-optional that. 27

O obligation. 27

OB it is obligatory that. 27

OM it is omissible that. 27

OP it is optional that. 27

P permission. 27

PE it is permissible that. 27

RND Rule of Deontic Necessitation. 27

SDL Standard Deontic Logic. 6, 10, 17

SMT Satisfiability Modulo Theories. 2

SSE Shallow Semantic Embedding. 2, 6, 9

STIT Seeing-to-it-That. 20

T-STIT Temporal STIT. 49

TDS The Traditional Definitional Scheme. 27

TDS Temporal Deontic STIT. 49, 88

TL Tense Logic. 19

TMDL Term-modal Deontic Logic. 96

XML Extensible Markup Language. 11

Chapter 1

Introduction

1.1 Context

The integration of AI systems into society has spurred a pressing need for robust ethical guidelines and legal frameworks to govern their usage. In contemporary times, most people have come in close contact with AI systems via AI filters on social media, customer service bots, or Open AI's newest language model, ChatGPT. The latter is currently a matter of controversial debate. Newspapers, podcasts, and other media channels publish discussions on the benefits and dangers of the new technology. Concerns range from issues of data privacy (Gillmann et al., 2023) to the implications of ChatGPT in educational settings (Kammerl & Kniess, 2023), as well as the reliability of information generated by the model (Moini, 2023).

In response to the deep intertwining of technology and society, governments worldwide are challenged with finding suitable regulations for new technologies. The so-called AI Act (AIA) is the first stride in governing AI systems within the European Union (EU) (European Parliament, 2023). After months of discussions, the European Council approved it on the 21st of May 2024 (Uuk, 2023, 2024b). Once the Council and the European Parliament have signed the document, it will enter into force after twenty days (Uuk, 2024a). Member states will have a two-year transitional period to implement the new law, with some provisions holding earlier (Uuk, 2024b).

This complex legislation is expected to impact the companies mandated to adhere to it,

the individuals who must be aware of it, and the institutions tasked with its enforcement. It also raises difficult questions: How can the AIA be implemented in practice? How can lawmakers check whether AI systems conform to the regulations and ensure companies fulfill their new duties?

The thesis seeks to explore solutions to these challenges, aiming to contribute valuable insights into the practical implementation of the AIA. One approach to ensuring compliance with the AIA involves leveraging AI itself, particularly symbolic AI, to represent the AIA's rules and restrictions in a logical language. By encoding the AIA in a formal logic framework, AI systems could potentially self-regulate. Ideally, verification of conformity with the AIA could then occur automatically, based on various inputs such as text, questionnaires, or even real-time AI applications.

1.2 Goals and Methodology

This thesis builds upon the Logic and Knowledge Engineering Framework and Methodology (LogiKEy) framework, which aims to develop computational tools for normative reasoning based on formal methods (Benzmüller et al., 2020). LogiKEy employs the technique of Shallow Semantic Embedding (SSE) to integrate diverse logics into Higher Order Logic (HOL), leveraging HOL as a meta-logic to represent the syntax and semantics of various logics. This integration is facilitated within *Isabelle*—specifically, its variant for HOL, *Isabelle/HOL*—a proof assistant tool equipped with state-of-the-art higher-order Automated Theorem Provers (ATP), Satisfiability Modulo Theories (SMT) solvers, and model finders (Benzmüller et al., 2020).

The ultimate goal is to represent the AIA in Isabelle/HOL, enabling automated reasoning about systems' and companies' compliance with AIA regulations. Since achieving this entirely within this thesis is unrealistic, the goal is to identify relevant modalities and discuss logics suitable for their representation. A number of these modalities will be studied in more

detail, using existing embeddings or newly created embeddings of such logics for the representation of example parts from the AIA, and testing how well the available theorem provers can automatically reason with the formalized examples. This will yield insights into the practical challenges of automated legal reasoning in complex environments, specifically by identifying the strengths and weaknesses of current theorem-proving tools and assessing the suitability of different logics and their embeddings in Isabelle/HOL for representing specific modalities within the AIA.

The focus of this thesis is practical. While background and details on the utilized logics and their embeddings are provided, the thesis primarily explores the practical limitations of representing legal information and automatic reasoning in the legal domain.

1.3 Research Questions

Three research questions will guide the project:

- **R1:** What modalities (alethic, deontic, epistemic, temporal, etc.) are relevant within the AIA?
- **R2:** How can concrete examples from the AIA, employing these modalities, be represented in Isabelle/HOL, and which logics need to be embedded for this purpose?
- **R3:** Are the theorem provers available in Isabelle/HOL effective in reasoning with these representations?

1.4 Structure of the Thesis

The subsequent chapters are structured as follows: Chapter 2 introduces the LogiKEy approach, discusses important related work, and provides a brief overview of alternative

frameworks for legal representation. Following this, Chapter 3 outlines the methodology used in the thesis, while Chapter 4 presents an analysis of the AIA and its modalities. Chapter 5 then focuses on a selection of these modalities, formalizing concrete examples from the AIA in Isabelle/HOL and analyzing whether the available theorem provers can reason with them. To create a representations of example cases, existing embeddings of suitable logics will be used, or new embeddings will be created and tested. Experimenting with example cases will provide insights into the suitability and effectiveness of the logical embeddings and the available theorem provers. Finally, Chapter 6 concludes the thesis by summarizing key findings and implications, discussing limitations, and suggesting directions for future research.

Chapter 2

Background and Related Work

The following chapter starts by explaining the LogiKEy approach. Sections Two and Three continue to introduce Higher-Order-Logic (HOL), the formal framework used in LogiKEy, as well as in this thesis, and the tool Isabelle/HOL. Section Four explains the process of Shallow Semantic Embedding (SSE), which enables the embedding of other logics in HOL. Finally, Section Four provides an overview of alternative approaches to formalizing legal documents.

2.1 LogiKEy

The LogiKEy framework is designed to facilitate the creation of ethical, intelligent reasoners through the use of theorem provers and formal methods. Specifically, LogiKEy is interested in enhancing possibilities to regulate and control autonomous intelligent systems. (Benzmüller et al., 2020).

The founders of LogiKEy envision an ethical reasoner capable of evaluating the (proposed) behaviors of AI systems against formalized ethico-legal theories. Due to the complexity of modeling these theories and normative concepts, LogiKEy employs HOL as its formal framework. It utilizes HOL as a meta-logic, benefiting from its expressive power and flexibility that allows for the embedding of a wide range of logical systems within a unified

framework (Benzmüller et al., 2020).

Although HOL is undecidable, the SSE approach (Chapter 2.4) can map decidable proof problems, for example, in Standard Deontic Logic (SDL), onto decidable fragments of HOL. Consequently, systems like the proof assistant Isabelle/HOL (University of Cambridge & Technische Universität München, 2023) can handle the resulting formulas and employ integrated Automated Theorem Provers (ATP) and Satisfiability Modulo Theories (SMT) solvers, along with model finders that provide robust support for effective automated reasoning (Benzmüller et al., 2020).

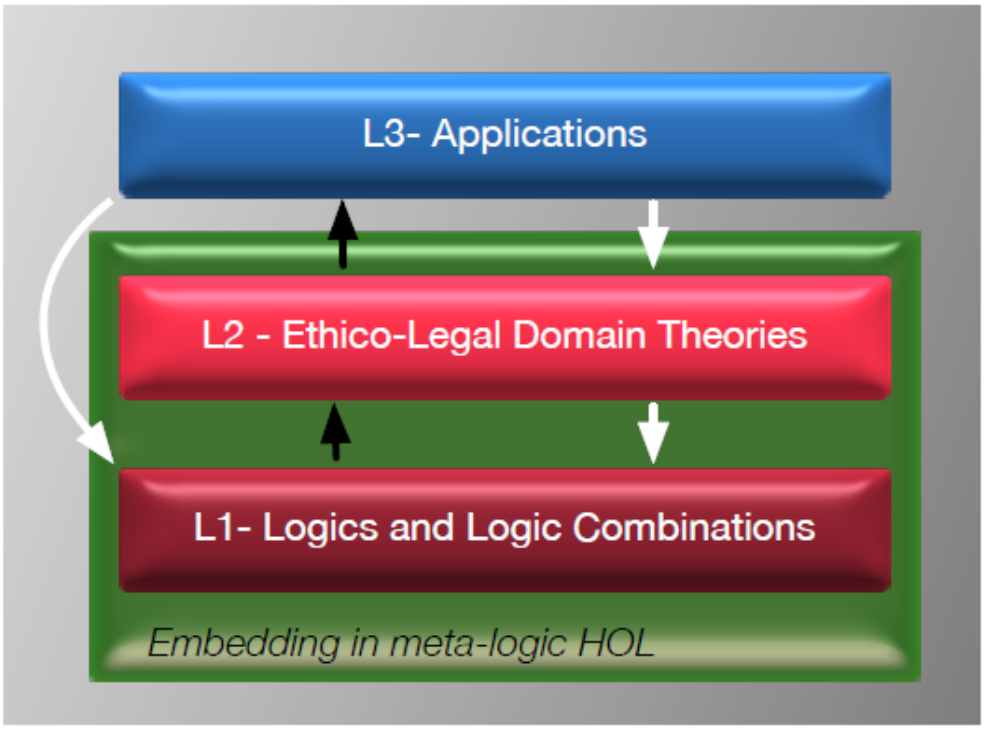


Figure 2.1: LogiKEy layers (Benzmüller et al., 2020)

LogiKEy is composed of three layers, stacked as displayed in Figure 2.1. Layer one contains the necessary logics and logic combinations embedded in HOL. Most of these logics will be deontic since deontic systems are suitable for normative and legal reasoning. In layer

two, ethico-legal theories can be formalized using the logics and logic combinations from layer one. Finally, the application reasoning about concrete examples using the information from the lower-level layers is situated in layer three (Benzmüller et al., 2020).

Up until now, several logics for layer one, such as SDL (Benzmüller & Parent, 2018), Dyadic Deontic Logic (DDL) (Benzmüller et al., 2022), and Åqvist's system E (Benzmüller et al., 2018) have already been embedded in HOL using the SSE approach and can be accessed in the LogiKEy repository (Benzmüller et al., 2023). Researchers have also used such embeddings to model example use cases, such as a lawyer's argumentation using Modal Preference Logic (Benzmüller & Fuenmayor, 2021).

To better understand the LogiKEy framework and the work attempted in this thesis, the following three Sections will provide a more detailed discussion of HOL, the tool Isabelle/HOL, and the SSE approach, respectively.

2.2 Higher Order Logic

HOL, also known as type theory, was first introduced and formalized in 1908 by Russell (Russell, 1908). Its current version is strongly influenced by the simple typed lambda calculus that was put forward in Alonzo Church's Simple Type Theory (Church, 1940).

Being a foundational framework for mathematical reasoning, HOL finds extensive use in many applications, such as formalizing mathematics, verifying hardware and software, and studying the formal semantics of natural language (Benzmüller & Andrews, 2022; Benzmüller et al., 2020). Reasons for the popularity of HOL include the fact that it avoids many known inconsistencies and that it employs λ-notation as a strategy to represent unnamed functions, predicates, and sets (Benzmüller et al., 2020). Additionally, HOL is well studied in the scientific community, and several powerful ATPs and Interactive Theorem Provers (ITPs) for HOL exist, whose performance is verified (Benzmüller, 2019).

In HOL, all terms have types. The set of simple types T is created freely from the set of

basic types σ, ι. The type σ is used for the boolean truth values, whereas ι denotes the type of individuals. Then if α and β are in T, so is the function type $\alpha \to \beta$ (Benzmüller et al., 2004).

Using C_a for typed constants and x_a for typed variables different to C_a, the terms of HOL can be defined as follows:

$$C_a, x_a, (\lambda x_\alpha . s_\beta)_{\alpha\to\beta}(\text{abstraction}), (s_{\alpha\to\beta} t_\alpha)_\beta(\text{application})$$

Via *abstraction*, a variable a of type α is mapped to a term s of type β. Via *application*, a term of type β can be constructed by applying the function s $\alpha \to \beta$ to the argument t of type β (Benzmüller et al., 2004, 2020).

The signature is assumed to also contain negation ($\neg_{\sigma\to\sigma}$), disjunction ($\vee_{\sigma\to\sigma\to\sigma}$), and universal quantification ($\prod_{(\alpha\to\sigma)\to\sigma}$) for each type α as primitive logical connectives. Based on these primitives, other logical connectives (conjunction, implication, existential quantification, etc.) can be defined. For universal quantification, the shorter binder notation ($\forall x_\alpha . s_\sigma$) can be used to express $\prod_{(\alpha\to\sigma)\to\sigma} \lambda x_\alpha . s_\sigma$ (Benzmüller et al., 2004, 2020).

Moreover, constants for logical equality ($=_{\alpha\to\alpha\to\sigma}$) and description or choice ($\in_{(\alpha\to\sigma)\to\alpha}$) for each type α can be included. Terms of type σ are referred to as *formulas*, and type information can be left out if it is clear from the context (Benzmüller et al., 2020).

For more in-depth information on HOL, the reader may refer to the literature (Benzmüller & Andrews, 2022; Benzmüller et al., 2004; Church, 1940).

2.3 Isabelle/HOL

The tool that is used to work with HOL within the LogiKEy framework is the interactive theorem prover *Isabelle*. While it supports a variety of logical frameworks, only its specialization for HOL, *Isabelle/HOL*, is relevant for this thesis.

Programming in Isabelle/HOL means creating theories composed of types, functions, and

theorems. Isabelle/HOL provides several base types, e.g., for truth values (*bool*) and natural numbers (*nat*), and type constructors, e.g., for lists, function types, and type variables. However, users can also create custom types (Nipkow et al., 2022).

More complex HOL terms are then constructed via function application. Additionally, terms may include lambda abstractions (Nipkow et al., 2022). Lambda abstractions are statements of the following form: $\lambda x.B$, with λ marking the start of the abstraction, x representing a variable, and B representing the body, usually a function or expression containing x. For example, the following is a simple lambda abstraction representing the identity function: $\lambda x.x$ (Nipkow, 2013).

Isabelle/HOL terms of type bool, such as the basic constants True and False and the logical connectives, are called formulas (Nipkow, 2013).

Even though Isabelle/HOL employs type inference, it is often required to annotate terms to avoid ambiguity. This is achieved via the following notation: $t :: \Gamma$, which declares t to be of type Γ (Nipkow, 2013).

Particularly relevant for this thesis is the integration of Isabelle/HOL with the *Sledgehammer* tool, which provides access to powerful ATPs and SMT solvers (J. C. Blanchette & Paulson, 2013) and the counterexample generator *Nitpick* (J. Blanchette, 2023). These tools are useful for testing the embeddings of selected logics and for automated reasoning with example use cases.

2.4 Shallow Semantic Embeddings

As discussed earlier, the SSE approach is used to represent a chosen logic in HOL. Its general idea is "to provide a lean and elegant equational theory which interprets the syntactical constituents of [a chosen] logic (...) as lambda-terms of the meta-logic HOL" (Benzmüller, 2019, p. 3).

Different from deep embedding strategies, the SSE approach allows HOL and the logic to

embed to share what they have in common, addressing only their relevant differences. Since several modal logics have already been successfully embedded in HOL (Benzmüller, 2019; Benzmüller & Parent, 2018; Benzmüller et al., 2015), we will explain the process using the embedding of SDL as an example.

An important difference between SDL and HOL is the possible world semantics of SDL, which must be modeled in HOL (Benzmüller & Parent, 2018). To achieve this, boolean-valued SDL formulas are identified with world-predicates of type $o = i \to bool$ in HOL, with i being the type for worlds. This way, a formula can be evaluated relative to a given world in HOL. To represent the logical connectives of SDL, they must be lifted to relations on worlds in HOL. For example, the negation operator of SDL takes an input of type o and returns an output of type o. It is identified with the following lambda term in HOL: $\neg_{o\to o} \equiv \lambda\phi_o.\lambda w_i.\neg(\phi w)$ (Benzmüller & Parent, 2018).

The connectives $\wedge$, $\vee$, $\to$, and $\leftrightarrow$ all have the type $o \to o \to o$, meaning they take in two terms of type o and a world w of type i, and then return the conjunction, disjunction, implication, or biconditional of these terms at the given world. The lambda term identified with $\wedge$ in HOL looks as follows: $\wedge_{o\to o\to o} \equiv \lambda\phi_o.\lambda\upsilon_o.\lambda w_i.(\phi\, w \wedge \upsilon\, w)$. Lambda terms for the other connectives in HOL are constructed accordingly (Benzmüller & Parent, 2018).

To define the obligation operator of SDL, we use the same approach. It is identified with the following lambda term: $OB_{o\to o} \equiv \lambda\phi_o.\lambda w_i.\forall v_i.(w\, r\, v \longrightarrow \phi\, v)$, with r being the accessibility relation denoting the ideal worlds from the perspective of a given world (Benzmüller & Parent, 2018).

Lastly, the notions of global and local validity of SDL formulas must be embedded in HOL. They both have the type $o \to$ bool and are identified with the following lambda terms: $\lfloor\rfloor_{o\to bool} \equiv \lambda\phi_o.\forall w_i.(\phi\, w)$ (global validity) and $\lfloor\rfloor_{l\ o\to bool} \equiv \lambda\phi_o.(\phi\, cw)$ with cw referring to the current world (local validity) (Benzmüller & Parent, 2018).

2.5 Other Frameworks for Legal Representation

The LogiKEy is neither the first nor the only approach towards representing and reasoning with legal or normative theories. The following two Sections will introduce the Architecture for Knowledge-Oriented Management of African Normative Texts using Open Standards and Ontologies (Akoma Ntoso) standard (OASIS Open, 2018) and LegalRuleML (Athan et al., 2015; Palmirani et al., 2011).

2.5.1 Akoma Ntoso

Akoma Ntoso is a standardized framework designed to represent the structure of legislative, parliamentary, and judicial documents in a machine-readable format using Extensible Markup Language (XML). It was developed under the "Africa i-Parliament Action Plan" by UN/DESA and has since grown into a globally recognized standard. Akoma Ntoso translates to "linked hearts" in the language of West Africa, representing its goal of fostering connectedness and accessibility in the domain of legal information (Cornell Law School, n.d.; OASIS Open, 2018).

The standard defines a comprehensive schema that includes metadata and content components, facilitating the detailed and precise representation of legal texts. A complete introduction to the standard and its vocabulary can be accessed online (OASIS Open, 2018).
Akoma Ntoso is helpful for various applications. For example, it can assist in drafting legal documents, help legal researchers with advanced search and indexing functionalities, and promote transparency by making legal documents more easily accessible to the public. Even the European Parliament has adopted Akoma Ntoso to manage its legislative documents (OASIS Open, 2018).

While Akoma Ntoso is primarily concerned with the structure and metadata of legal documents and keeping them organized, Legal RuleML focuses on extracting their logical content and enabling automated reasoning.

2.5.2 LegalRuleML

LegalRuleML was created under the guidance of the OASIS LegalRuleML Technical Committee (OASIS Open, 2020). It is an extension of RuleML, a family of XML-based languages designed for expressing rules and logical statements in a standardized, machine-readable format (Boley et al., 2010), tailored to the needs of the legal domain. These include the possibility of modeling both *constitutive rules*, which define legal concepts, relations, activities, institutions, etc., and *prescriptive rules* governing actions or outcomes by declaring them obligatory, permitted or prohibited (Athan et al., 2015). Other features of LegalRuleML that are specific to legal reasoning are the implementation of isomorphism between rules and normative provisions, the management of rule reification, the representation of normative effects and values, and the modeling of legal procedural rules (Athan et al., 2015).

Finally, a key requirement of legal reasoning is the implementation of defeasibility: "When the antecedent of a rule is satisfied by the facts of a case (or via other rules), the conclusion of the rule presumably holds, but is not necessarily true" (Palmirani et al., 2011, p. 301). Defeasibility is essential for reasoning in the legal domain since it allows for the expression of exceptions from rules and for rules conflicting with each other without causing inconsistencies and contradictions (Athan et al., 2015). The task of managing defeasible legal reasoning has been extensively studied by many researchers. Among the most influential ones are Governatori and colleagues, whose research is essential in the field of automated legal reasoning, and whose key contributions include the integration of Answer Set Programming (ASP) in legal reasoning, especially the implementation of defeasible deontic logic in ASP (Governatori, 2023; Governatori & Wong, 2023; Governatori et al., 2004).

For further details on LegalRuleML and its XML schemas, the reader may refer to the LegalRuleML Core specification from OASIS (OASIS Open, 2020), and the literature (Athan et al., 2015; Palmirani et al., 2011).

Chapter 3

Methodology

This chapter presents the methodological approach of the thesis in detail.

In the first step, the AIA will be carefully analyzed (Chapter 4). Specifically, pages 38 to 88 of the AIA document are considered since they contain the requirements for AI systems and for the agents involved in their production and usage (*Artificial Intelligence Act*, 2021).

To ensure a thorough understanding, the AIA will be read multiple times, and results will be visualized using mind maps and tables. In the analysis, special attention is directed towards the logical modalities (obligations, permission, etc.) used to express rules for AI systems and responsibilities of actors rather than the concrete rules and regulations themselves. For each identified modality, the logics that seem suitable to represent them will be described briefly. Additionally, the importance of representing each modality in a logic system intended to formalize the AIA will be assessed.

Using the insights gained from the analysis, the next step is to represent the identified modalities in Isabelle/HOL. Due to the scope of this thesis, the discussion in Chapter 5 will be limited to those modalities identified as most crucial. For these modalities, appropriate logics and their shallow semantic embeddings in Isabelle/HOL will be introduced. The embeddings will then be employed to formalize and test selected excerpts from the AIA.

During the tests, Isabelle's Sledgehammer tool, a powerful feature that automates aspects of the theorem-proving process by leveraging external ATPs and SMT solvers, will be used

(J. C. Blanchette & Paulson, 2013). In Isabelle, Sledgehammer is called by typing "sledgehammer" after a lemma/theorem. If the tool finds a proof, the proof and the employed ATPs or SMTs are shown in the output console. Moreover, the model finder Nitpick will be used to generate models or disprove claims with counterexamples (J. Blanchette, 2023). To use Nitpick in Isabelle, "nitpick" must be typed after a lemma/theorem. Additional arguments are provided by listing them in square brackets ([ ]) after the nitpick command. The most important ones are *satisfy*, which tells Nitpick to search for a model that satisfies all the axioms and definitions in the theory; *user_axioms*, which asks Nitpick to take into account all user-defined axioms when searching for a model, *show_all*, which tells Nitpick to display detailed information about the found model; and *card 'type' = n*, which can be used to specify the cardinality of a specific type in the model (J. Blanchette, 2023).

The experiments attempt to find a logic or logic combination (embedded in Isabelle/HOL) that can reason with the selected modalities as well as possible. This logic or logic combination will be built step by step in Chapter 5. First, a logic apt to formalize statements containing the first selected modality is introduced and used, and then it is altered, exchanged, or combined with another logic apt for the second selected modality. This process will be continued until all selected modalities have been discussed.

The success or failure of this attempt will help discover the strengths and limitations of the embedded logics and the available theorem provers. Moreover, it should offer valuable insights into the feasibility of using automated logical reasoning in complex practical contexts like the AIA.

Finally, all conclusions, limitations, learnings, and open questions will be summarized in Chapter 6.

Chapter 4

Analysis of the AI Act

In the following chapter, the results of the analysis are discussed one modality at a time. A visual summary of the whole analysis can be found in Figure 4.3[1]. Additional tables and mind maps created during the analysis can be found in Appendix One.

[1] All figures without a source have been created by the author.

Tiles/Chapters	Modalities
Tile 2 Prohibited AI Practices	• Obligations/Prohibitions/Permissions • Agentive Obligations/Agency • Fuzziness • Exceptions • Temporality
Tile 3, Chapter 1 High-risk AI Practices	• Obligations/Prohibitions/Permissions • Agentive Obligations/Agency • Temporality
Tile 3, Chapter 2 High-risk AI Practices	• Obligations/Prohibitions/Permissions • Agentive Obligations/Agency • Temporality • Fuzziness
Tile 3, Chapter 3 High-risk AI Practices	• Agentive Obligations/Agency • Contrary-to-Duty Obligations (CTDs), involving Agents • Fuzzy CTDs • Fuzziness • Beliefs/Knowledge of Agents
Tile 3, Chapter 4 High-risk AI Practices	• Agentive Obligations/Agency • CTDs, involving (multiple) Agents, with Temporal Implications • CTDs involving multiple Agents • Beliefs/Knowledge of Agents
Tile 3, Chapter 5 High-risk AI Practices	• Agentive Obligations/Agency • CTDs, involving (multiple) Agents, with Temporal Implications • Temporality • Exceptions • Fuzziness • Beliefs/Knowledge of Agents
Tile 4 Transparency Obligations	• Agentive Obligations/Agency • Exceptions

Figure 4.1: Part 1

Tile 5 Measures in Support of Innovation	• Obligations/Prohibitions/Permissions • Agentive Obligations/Agency • Exceptions • Temporality
Tile 6, Chapter 1 Governance	• Obligations/Prohibitions/Permissions • Agentive Obligations/Agency • Fuzziness • Exceptions
Tile 6, Chapter 2 Governance	• Agentive Obligations/Agency • Fuzziness
Tile 7 EU Database for stand-alone high-risk AI-Systems	• Agentive Obligations/Agency • Fuzziness
Tile 8 Post-market Monitoring, Information Sharing, Market Surveillance	• Obligations/Prohibitions/Permissions • Temporality • Agentive Obligations/Agency • CTDs, involving Agents • Fuzziness • Beliefs/Knowledge of Agents
Tile 9 Codes of Conduct	• Agentive Obligations/Agency
Tile 10 Confidentiality and Penalties	• Agentive Obligations/Agency • Temporality
Tile 11 Delegation of Power and Committee Procedure	• Agentive Obligations/Agency • Temporality
Tile 12 Final Provisions	• Agentive Obligations/Agency • Temporality

Figure 4.2: Part 2

Figure 4.3: AIA modality analysis

4.1 Obligations, Prohibitions, and Permissions

Given that the AIA is a legislation designed to regulate the use of AI systems, it is unsurprising that it contains many obligations, prohibitions, and permissions.

Obligations in the AIA are expressed with the word "shall" such as in this sentence from Article eight: "High-risk AI systems shall comply with the requirements established in this Chapter." (*Artificial Intelligence Act*, 2021, p. 46). The negated shall, "shall not" is em-

ployed to express *prohibitions* in the AIA. An example prohibition can be found in Article 26: "Where an importer considers or has reason to consider that a high-risk AI system is not in conformity with this Regulation, it shall not place that system on the market..."(*Artificial Intelligence Act*, 2021, p. 56). Finally, the AIA has two strategies for the expression of *permissions*: It either uses "may" or "is/are empowered to" (*Artificial Intelligence Act*, 2021).

SDL, the most studied system of deontic logic, can express and reason with all the described modalities. It is a modal logic that builds upon classical propositional logic and introduces a monadic deontic operator, which can express that a formula is obligatory ($O(A)$). Other deontic modalities, such as prohibitions and permissions, can be defined based on this operator (McNamara & Van De Putte, 2022).

Since the AIA primarily consists of obligations for certain kinds of AI systems, it is essential to enable their representation in a logic system meant to formalize the AIA. This will be analyzed further and experimented with in Chapter 5.1.

4.2 Fuzziness

Fuzzy statements can be found in many places within the AIA. The following expressions are examples of such fuzziness: "unless and in as far as such use is strictly necessary for one of the following objectives" (*Artificial Intelligence Act*, 2021, p. 43), "to the extent to which..." (*Artificial Intelligence Act*, 2021, p. 46), "at a minimum" (*Artificial Intelligence Act*, 2021, p. 49). All these statements express vague notions or involve degrees of truth/necessity.

A logic that can do justice to this kind of vagueness is called fuzzy logic and is employed "to model logical reasoning with vague or imprecise statements" (Cintula et al., 2023). Typically, fuzzy logic assigns a degree of truth to a proposition expressed on a scale within the real unit interval [0, 1], with zero equal to total falsehood and one equal to total truth (Cintula et al., 2023).

Fuzziness does repeatedly occur within the AIA. However, due to the constraints of this

thesis, it will not be discussed, remaining open for future work.

4.3 Exceptions

Within the AIA, the concept of exceptions from general rules emerges as a recurring theme. Consider the following sentence from Article Five as an example: "The following artificial intelligence practices shall be prohibited: the use of 'real-time' remote biometric identification systems in publicly accessible spaces for the purpose of law enforcement, unless and in as far as such use is strictly necessary for one of the following objectives:..." (*Artificial Intelligence Act*, 2021, p. 43). This sentence outlines a general rule – the prohibition of certain AI practices – followed by exceptions based on specific objectives.

Non-monotonic logic can deal with such exceptions. It is designed to handle defeasible inference, enabling reasoners to retract conclusions when warranted by additional information, such as the presence of objectives allowing for an exception (Strasser & Antonelli, n.d.).

Exceptions from rules are stated in several articles of the AIA and should hence be expressed correctly in a formalization. Their expression in non-monotonic logic will not be analyzed in this thesis. However, an alternative approach to formalizing them is discussed in Chapter 5.1.3.

4.4 Temporality and Temporal Notions

Time is an important concept within the AIA, with many articles specifying obligations that must be fulfilled before certain events occur. For instance, Article 19 states: "Providers of high-risk AI systems shall ensure that their systems undergo the relevant conformity assessment procedure in accordance with Article 43, prior to their placing on the market or putting into service" (*Artificial Intelligence Act*, 2021, p. 54). This obligation for providers is tied to a specific point in time; it must be fulfilled **before** the system is placed on the

market or put into service.

Additionally, some sentences contain hidden temporal notions such as this phrase in Article Five: "When implementing the risk management system (...), x shall..." (*Artificial Intelligence Act*, 2021, p. 48). The sentence implies simultaneity, which becomes obvious if rephrased as follows: "While implementing the risk management system, x is obligated to."

Temporal logic is a class of modal logics that can formally represent and argue with temporality (Goranko & Rumberg, 2023). The most popular system is called Tense Logic (TL) and dates back to the work of Prior (Lejewski, 1959; Prior, 1959; Prior, 1967, 1968). TL treats propositions as tensed and introduces temporal operators into the language, thereby allowing for the representation of temporal relations like the one in the AIA (Goranko & Rumberg, 2023).

While temporal notions don't occur frequently, they are essential to some of the regulations of the AIA. Ideally, the logic system used to formalize the AIA should facilitate expressing them. However, due to the limitations of this thesis, temporality will not be discussed in depth here.

4.5 Agency and Agentive Obligations

Since the AIA is concerned with defining rules for the training, usage, and handling of AI systems, a huge part of it states obligations and prohibitions. However, many of these obligations are not general in nature but relate to agents with a specific position regarding the AI systems, e.g., to providers or importers. For instance, the sentence "Providers of high-risk AI systems shall (...) have a quality management system in place which complies with Article 17" (*Artificial Intelligence Act*, 2021, p. 52) is such an agentive obligation since it expresses what the provider ought to do.

To formalize agentive obligations, a modal logic of agency is needed. While there are many ideas for formulating such a logic (Anderson, 1962; Brown, 1988; Fitch, 1963; Kanger,

1972; Von Wright, 1963, 1981), the most prominent theory of agency is a branch called Seeing-to-it-That (STIT) logic that originated in the works of Belnap (Belnap, 1991) and Belnap and Perloff (Belnap & Perloff, 1988). STIT theory introduces a STIT operator of the form

$$a \text{ stit } F$$

which expresses that an agent a sees to it that F holds. Given that the AIA states not only the actions of agents but also how they ought to behave, a suitable agency logic must also be able to formulate obligations for specific agents. Several authors have researched on this issue and presented their approaches, which could be apt to help represent the AIA (J. Horty, 2001; J. F. Horty & Belnap, 1995; Murakami, 1998; van Berkel & Lyon, 2019; Xu, 2015).

Different agents and their obligations are essential in the AIA, especially in Tile Three (Figure 4.3). Consequently, a logic system attempting to formalize the AIA should be able to express agentive modalities. Achieving this is the topic of Chapter 5.3.

4.6 Contrary-to-Duty-Obligations

Obligations have already been discussed in Chapter 4.6. However, another type of obligation that deserves special attention within the AIA is the so-called Contrary-to-Duty Obligations (CTD). A CTD is an obligation that arises only if a primary obligation is not fulfilled, meaning that it is "conditional on [...] violating [a] primary obligation" (McNamara & Van De Putte, 2022).

An example of a CTD in the AIA Act can be found in Article 16: "Providers of high-risk AI systems shall (...) take the necessary corrective actions if the high-risk AI system is not in conformity with the requirements set out in Chapter 2 of this Title" (*Artificial Intelligence Act*, 2021, p. 52). It has been stated before that high-risk systems must comply with the requirements in Chapter two (*Artificial Intelligence Act*, 2021, p. 46). This is the primary

obligation in the given context. However, obligations are not always fulfilled; they may also be violated. In the event of such a violation, the obligation to take corrective action becomes relevant. This is a CTD, applicable only when the primary obligation has been violated. CTD situations are often represented in a typical structure following Chisholm (1963). The discussed example then looks as follows:

> 1: It ought to be that high-risk AI systems comply with the requirements in Chapter 2.
> 2: It ought to be that if a high-risk AI system does not comply with the requirements in Chapter 2, providers take corrective actions.
> 3: If a system complies with the requirements in Chapter 2, the provider must not take any corrective actions.
> Concrete Situation: The system does not comply with the requirements in chapter two.

Note that some obligations in this example are agentive for the provider. As agency has already been discussed in the previous section, it will not be elaborated on here.

DDL by Carmo and Jones is a suitable logic to express adequately and reason with CTDs (Benzmüller et al., 2022; Carmo & Jones, 2002a, 2022). It differentiates between ideal and actual obligations and introduces a conditional obligation operator, thereby enabling the expression of CTDs that arise when ideal obligations have been violated (Benzmüller et al., 2022; Carmo & Jones, 2002a).

Next, two special kinds of CTDs will be discussed.

4.6.1 Contrary-to-Duty-Obligations Involving Multiple Agents

Some CTDs within the AIA relate to multiple agents, making the matter more complex. The following example from Article 36 will be discussed: "If the notifying authority comes to the conclusion that the notified body (...) is failing to fulfill its obligations, it shall restrict,

suspend or withdraw the notification as appropriate, depending on the seriousness of the failure" (*Artificial Intelligence Act*, 2021, p. 62). For simplicity, the notion of an agent belief expressed in the word *ascertains* is ignored since beliefs are the subject of Chapter 4.7. Also, we disregard the temporal notion contained in *no longer* since CTDs involving temporality will be discussed in the next section.

The focus here lies on the involvement of two distinct agents: The notifying authority and the notified body. The situation is as follows: The notified body has several primary obligations that are specified within the AIA (*Artificial Intelligence Act*, 2021, 60f.). If it violates these obligations, another obligation arises. However, this obligation is agentive for the notifying authority, not for the notified body that violated the primary obligation.

4.6.2 Contrary-to-Duty-Obligations Involving Temporality

Moreover, there are cases in the AIA where CTDs occur in combination with temporality, stating that a certain property used to be fulfilled but *no longer* is. An example of this can be found in Article 36: "Where a notifying authority has suspicions or has been informed that a notified body no longer meets the requirements laid down in Article 33 (...) that authority shall without delay investigate the matter with the utmost diligence" (*Artificial Intelligence Act*, 2021, p. 62). To understand the involved CTD, Article 32 must be taken into account. Here, it is stated that the notifying authority "may notify only conformity assessment bodies which have satisfied the requirements laid down in Article 33" (*Artificial Intelligence Act*, 2021, p. 59).

Unfortunately, this example does involve not only temporality but also multiple agents, a matter discussed in the previous section. To focus only on the temporality here, we will reformulate it as follows:

> Only conformity assessment bodies that fulfill the requirements in Article 33 before the notification may be notified (adapted from Article 32). If a conformity

assessment body that was notified (= notified body) no longer meets the requirements laid down in Article 33, the matter shall be investigated further with the utmost diligence (adapted from Article 36).

Now it becomes visible how this example is different from a typical CTD: Technically speaking, the notified body is only obligated to fulfill the requirements in Article 33 at one point in time *before* being notified. Hence, the fact that a notified body *no longer* fulfills the requirements in Article 33 does not equal a violation of the previous obligation. The obligation to further investigate the matter then is just a normal obligation, not a CTD, and could be expressed as such. However, in this case, the logic representing this situation must not only contain obligation operators but also be able to express temporality to capture the *before* notion (as discussed in Chapter 4.4).

Another possibility is disregarding the temporality here and translating the example to match the usual CTD structure:

1: It ought to be that notified bodies fulfill the requirements in Article 33.
2: It ought to be that if a notified body does not comply with the requirements in Article 33, further investigations are started.
3: If a notified body does comply with the requirements in Article 33, no further investigation must be started.
Concrete Situation: A notified body does not fulfill the requirements in Article 33.

Reformulated as above, the example can be treated like a typical CTD.

CTDs, in general, are relevant to many of the rules within the AIA, making them an essential modality that a logic trying to formalize the AIA should be able to reason with. This will be analyzed in depth in Chapter 5.2.

4.7 Beliefs of Agents

Agency and agentive obligations have already been discussed, but it is crucial to acknowledge another related modality: Agent beliefs. For example, Article 21 states:

> "Providers of high-risk AI systems which consider or have reason to consider that a high-risk AI system which they have placed on the market or put into service is not in conformity with this Regulation shall immediately take the necessary corrective actions to bring that system into conformity, to withdraw it or to recall it, as appropriate" (*Artificial Intelligence Act*, 2021, p. 55).

Here, the provider's belief that a high-risk AI system is not in conformity is relevant, not whether the system is, in fact, not conforming. Beliefs can be either correct or incorrect, and a logic suitable to handle them must distinguish between the facts and agent beliefs.

Fortunately, there is a field in logic equipped to deal with this called epistemic logic that allows for the exploration of different ideas of knowledge and beliefs of agents and their logical relations (Rendsvig et al., 2024). The most popular epistemic logic is a modal logic using possible world semantics that extends propositional logic with modal operators representing what agents *know* and *believe* (Rendsvig et al., 2024).

Beliefs and knowledge of agents do play a role in many parts of the AIA. However, it will not be possible to discuss epistemic logic in this thesis. The representation of agent beliefs within the AIA should be analyzed in future work.

4.8 Interactions and Combinations of Different Modalities

Obviously, the modalities identified up until now do not appear in isolation. Some of the examples discussed (e.g., Chapter 4.6) have already shown that they often are combined within a single sentence. While this has been ignored in the previous discussion for simplicity, it can not be denied when constructing a logical system meant to formalize the AIA.

Ultimately, such a system must not only accurately represent all the relevant modalities individually but also capture what happens when they are combined. In this thesis, the interplay of all modalities selected for further experimentation will be examined.

Chapter 5

Representing the AIA in Isabelle/HOL

In the following chapter, parts of the AIA containing the modalities identified in the previous chapter will be formalized in Isabelle/HOL using embeddings of suitable logic. As explained before, the constraints of this thesis do not allow for a detailed study and experimentation with all the modalities. Instead, the focus will lie on the most relevant ones within the AIA: obligations, CTDs, and agency/agentive obligations. These modalities occur in most articles of the AIA, and without them, a realistic representation of the AIA would be difficult to imagine. Studying the remaining modalities is a topic for future work.

The first Section discusses the representation of obligations, the second considers CTDs, and the third discusses agency and agentive obligations.

5.1 Representing Obligations

To start with, an Isabelle representation of Article 5, Point 1 on prohibited AI systems (*Artificial Intelligence Act*, 2021, pp. 43–45) will be created. This part is chosen because it contains almost exclusively obligations. The only other involved modality - exceptions - will be briefly covered in Chapter 5.1.3.

Before discussing the concrete representation of the chosen part in Chapter 5.1.2., SDL and its semantic embedding in Isabelle/HOL will be introduced shortly.

5.1.1 SDL and its Embedding in Isabelle/HOL

SDL is the most studied system of deontic logic and stems from the work of von Wright in 1951 (Dov Gabbay et al., 2013). It recognizes the usual six normative statuses of deontic logic: "it is obligatory that (OB), it is permissible that (PE), it is impermissible that (IM), it is omissible that (OM), it is optional that (OP), it is non-optional that (NO)" (McNamara & Van De Putte, 2022). Note here that these abbreviations are not standard, and different ones will be employed in this thesis, mainly obligation (O) and permission (P). SDL takes the obligation operator O as primitive so that the other operators can be defined according to The Traditional Definitional Scheme (TDS):

"$P(p) := \neg O(\neg p)$

IM(p) := $O(\neg p)$

OM(p) := $\neg O(p)$

OP(p) := $(\neg O(p) \wedge \neg O(\neg p))$

NO(p) := $(O(p) \vee O(\neg p)$" (McNamara & Van De Putte, 2022).

SDL uses classical propositional logic with an infinite set of propositional variables and the truth-functional operators ($\neg$, $\rightarrow$, $\wedge$, etc.) and extends it with the obligation operator $O(p)$, stating that the proposition p is obligatory (McNamara & Van De Putte, 2022). It is axiomatized by adding the Rule of Deontic Necessitation (RND) and the modal axiom schemata KD and DD to classical propositional logic (Dov Gabbay et al., 2013, p. 37). The RND states that if p is a theorem, so is $O(p)$. Schemata KD and DD, on the other hand, hold the following:

"(KD) $O(p \longrightarrow q) \longrightarrow (O(p) \longrightarrow O(q)$

(DD) $O(p) \longrightarrow \neg O(\neg p)$"

(Dov Gabbay et al., 2013, p. 37).

SDL is a normal modal logic since the RND is a version of the Necessitation rule K (Garson, 2023). The following theorems hold in SDL:

$O(p \wedge q) \longrightarrow (O(p) \wedge O(q))$ *(Conjunctive Distributivity of O)*

$O(p) \wedge O(q) \longrightarrow O(p \wedge q)$ *(Aggregation for O)*

$O(p) \longrightarrow O(p \vee q)$ *(Weakening)*

$O(p \longrightarrow q) \longrightarrow (P(p) \longleftarrow P(q))$

$P(p) \longrightarrow P(p \vee q)$

$P(p \vee q) \longrightarrow (P(p) \vee P(q))$ *(Disjunctive Distributivity of P)*

$P(p \wedge q) \longrightarrow (P(p)$

$O(\top)$ *(ON)*

$\neg O(\bot)$ (OD)

$O(p) \longrightarrow (P(p)$ *(DD')*

$O(p \wedge P(q)) \longrightarrow (P(p \wedge q)$

$O(p) \vee (P(p) \wedge P(\neg p)) \vee O(\neg p)$ *(Exhaustion)*

$\neg(O(p) \wedge (P(p) \wedge P(\neg p))) \wedge \neg(O(\neg p) \wedge (P(p) \wedge P(\neg p))) \wedge \neg(O(p) \wedge O(\neg p))$

(Dov Gabbay et al., 2013, p. 38).

In addition to these theorems, two rules of inference can be derived, namely the inheritance principle (1.) and the deontic equivalence rule (2.):

"1. If $p \longrightarrow q$ is a theorem, then $O(p) \longrightarrow O(q)$ is a theorem.

2. If $p \longleftrightarrow q$ is a theorem, then $O(p) \longleftrightarrow O(q)$ is a theorem"

(Dov Gabbay et al., 2013, p. 38).

Fortunately, an embedding of SDL has already been created, verified, and applied to example cases (Figure 5.1). To learn details about this embedding and explore case studies, the reader is advised to refer to the sources (Benzmüller & Parent, 2018; Benzmüller et al., 2020, 2023).

```
theory SDL (* SDL: Standard Deontic Logic. C. Benzmüller & X. Parent, 2019 *)
  imports Main types
begin

 typedecl i (*Type for possible worlds.*)
 type_synonym σ = "(i⇒bool)"
 type_synonym γ = "σ⇒σ"
 type_synonym ϱ = "σ⇒σ⇒σ"

 consts R::"i⇒i⇒bool" (infixr "R" 70) (*Accessibility relation.*)
        aw::i (*Actual world.*)

 abbreviation SDLtop::σ ("⊤") where "⊤ ≡ λw. True"
 abbreviation SDLbot::σ ("⊥") where "⊥ ≡ λw. False"
 abbreviation SDLnot::γ ("¬_"[52]53) where "¬φ ≡ λw. ¬φ(w)"
 abbreviation SDLand::ϱ (infixr"∧"51) where "φ∧ψ ≡ λw. φ(w) ∧ ψ(w)"
 abbreviation SDLor::ϱ (infixr"∨"50) where "φ∨ψ ≡ λw. φ(w) ∨ ψ(w)"
 abbreviation SDLimp::ϱ (infixr"→"49) where "φ→ψ ≡ λw. φ(w) ⟶ ψ(w)"
 abbreviation SDLequ::ϱ (infixr"↔"48) where "φ↔ψ ≡ λw. φ(w) ⟷ ψ(w)"

 abbreviation SDLobligatory::γ ("OB") where "OB φ ≡ λw. ∀v.  w R v ⟶ φ(v)"
 abbreviation SDLpermissible::γ ("PE") where "PE φ ≡ ¬(OB(¬φ))"
 abbreviation SDLimpermissible::γ ("IM") where "IM φ ≡ OB(¬φ)"
 abbreviation SDLomissible::γ ("OM") where "OM φ ≡ ¬(OB φ)"
 abbreviation SDLoptional::γ ("OP") where "OP φ ≡ (¬(OB φ) ∧  ¬(OB(¬φ)))"

 abbreviation SDLvalid::"σ⇒bool" ("⌊_⌋"[7]105)  where "⌊A⌋ ≡ ∀w. A w"       (*Global validity.*)
 abbreviation SDLvalidcw::"σ⇒bool" ("⌊_⌋ι"[7]105)    where "⌊A⌋ι ≡ A aw"   (*Validity in actual world.*)

(*Possibilist Quantification.*)
 abbreviation SDLforall ("∀") where "∀Φ ≡ λw.∀x. (Φ x w)"
 abbreviation SDLforallB (binder"∀"[8]9) where "∀x. φ(x) ≡ ∀φ"
 abbreviation SDLexists ("∃") where "∃Φ ≡ λw.∃x. (Φ x w)"
 abbreviation SDLexistsB (binder"∃"[8]9) where "∃x. φ(x) ≡ ∃φ"

axiomatization where
  D: "⌊¬ ((OB φ) ∧ (OB (¬ φ)))⌋" (*Axiom D: seriality of R.*)
 lemma seriality: "(∀w. ∃v. w R v)" using D by blast

 abbreviation SDLobl::γ ("○<_>") where "○<φ> ≡  OB φ"  (*New syntax: A is obligatory.*)

(*Consistency confirmed by model finder Nitpick.*)
lemma True nitpick[satisfy,user_axioms,expect=genuine] oops

assumes seriality
shows D
  sledgehammer oops

lemma "⌊(OB φ) → (OB (OB φ))⌋"
  nitpick[falsify,user_axioms,expect=genuine] oops

(*Barcan formulas.*)
lemma Barcan:         "⌊(∀d. ○<φ(d)>) → (○<∀d. φ(d)>)⌋" by simp
lemma ConverseBarcan: "⌊(○<∀d. φ(d)>) → (∀d. ○<φ(d)>)⌋" by simp

end
```

Figure 5.1: SDL embedding in Isabelle/HOL

5.1.2 Representing Example Paragraphs

Now, let's use this SDL embedding to represent the selected part from the AIA: Article 5, Point 1). To achieve this, additional types are needed (Figure 5.2). Most importantly, *aiSys* is the type for AI systems. The types file also introduces the constants *prohibited* and *high-risk*, which indicate the classification of an AI system.

```
(*prohibited*)
typedecl aiSys (*Type for AI-systems*)
datatype quality_person = age | physcial_disability | mental_disability (*quality of a person*)
datatype consequence = harm | harm_physical | harm_psychological | detri_treatment_unrelated_context |
detri_treatment_unjustified_disprop | affect_personal_rights | affect_personal_freedom (*type for consequences caused by AI-systems*)
datatype purpose = distort_behavior | exploit_groups | eval_trustworthiness_over_time | targeted_search |
prevention | detection (*type for purposes of AI-systems*)

consts
   prohibited::"aiSys⇒σ" (*system is declared prohibited*)
   high_risk::"aiSys⇒σ" (*system is declared a high-risk system*)
```

Figure 5.2: Types Article 5

Next, constant symbols mirroring the passage's content are declared (figure 5.3). For example, the constant *subliminal_technique* takes an AI system and evaluates to true in a given world if the AI System deploys a subliminal technique beyond a person's consciousness as written in Article 5, Point 1 a (*Artificial Intelligence Act*, 2021, p. 43). The constant *has_consequence* takes an AI System and a consequence and evaluates to true in a world if the AI System has the specified consequence, for example, *physical harm* or *psychological harm* (abbreviated here as harm_physical and harm_psychological). As visible in figure 5.3, more constant symbols are defined. Short explanations can be found in the comments, which should be sufficient to explain their intentions.

```
theory prohibited
  imports
   SDL
   types
begin

consts (*Predicates/relations*)
  subliminal_technique::"aiSys⇒σ" (*deploys a subliminal technique beyond a persons consciousness*)
  has_consequence::"aiSys⇒consequence⇒σ" (*system has or may have a consequence*)
  has_purpose::"aiSys⇒purpose⇒σ" (*system has a purpose*)
  exploits_vulnerabilities_group::"aiSys⇒quality_person⇒σ"
  exploits_vulnerable_group::"aiSys⇒σ" (*exploits any of the vulnerabilities of a specific group due to a certain quality*)
  employed_by_pauthorities::"aiSys⇒σ" (*employed by public authorities or on their behalf*)
```

Figure 5.3: Constants Article 5, point 1

In the following step, the constants are used to express the concrete rules given in the selected part of the AIA (Figure 5.4). Abbreviations *H1* and *H2* are helpers since they do not themselves state a rule but serve to make the expression of the rules easier. For example, the abbreviation *H1* holds that if an AI System x has either the consequence *harm_physical* or *harm_psychological*, it has the consequence harm.

The abbreviations *A1* to *C1* represent Point 1 of Article 5, except for Point 1 d. They all follow a similar pattern and state conditions under which AI Systems should be prohibited according to the AIA (*Artificial Intelligence Act*, 2021, p. 43). Point 1 d is left out here because it defines a prohibition that allows exceptions. Exceptions are a different modality, which will be discussed briefly in Chapter 5.1.3.

```
abbreviation "H1 ≡ ⌊∀x::aiSys. ((has_consequence x harm_physical) ∨
(has_consequence x harm_psychological)) ↔ has_consequence x harm⌋"
abbreviation "H2 ≡ ⌊∀x::aiSys. ((exploits_vulnerabilities_group x age) ∨ (exploits_vulnerabilities_group x physcial_disability)
∨ (exploits_vulnerabilities_group x mental_disability)) ↔ exploits_vulnerable_group x⌋"
abbreviation "A1 ≡ ⌊∀x::aiSys. subliminal_technique x  ∧  has_consequence x harm ∧
has_purpose x distort_behavior → ○<prohibited x>⌋"
abbreviation "B1 ≡ ⌊∀x::aiSys. exploits_vulnerable_group x ∧ has_purpose x distort_behavior ∧
has_consequence x harm → ○<prohibited x>⌋"
abbreviation "C1 ≡ ⌊∀x::aiSys. employed_by_pauthorities x ∧ has_purpose x eval_trustworthiness_over_time ∧
(has_consequence x detri_treatment_unrelated_context ∨ has_consequence x detri_treatment_unjustified_disprop)
→ ○<prohibited x>⌋"
```

Figure 5.4: Abbreviations Article 5, point 1 a-c

To work with the abbreviations and check whether Isabelle can reason with them, a situation containing information on a particular AI System must be constructed. This situation is described in the abbreviations *F1* to *F3*, where an AI Systems x is introduced and characterized (Figure 5.5).

```
consts
  x::aiSys

abbreviation "F1 w ≡ (subliminal_technique x) w"
abbreviation "F2 w ≡ (has_consequence x harm_physical) w"
abbreviation "F3 w ≡ (has_purpose x distort_behavior) w"
```

Figure 5.5: Facts Article 5 point 1

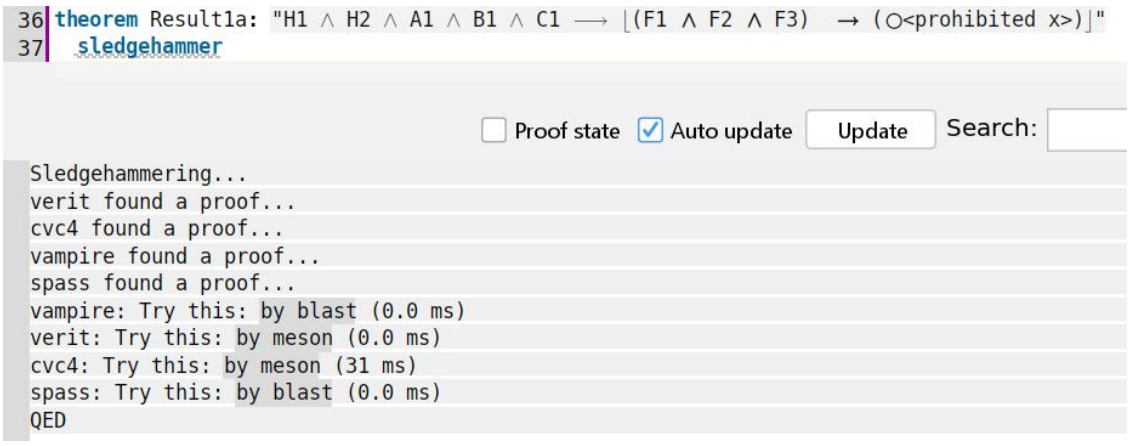


Figure 5.6: Sledgehammer proof Result1a

The facts about system x can now be used to prove theorems. Using *F1*, *F2*, and *F3*, it should be possible to conclude that system x is obligated to be prohibited according to Article 5, Point 1 a (*Artificial Intelligence Act*, 2021, p. 43). Sledgehammer confirms this since the tool finds proof for the statement (Figure 5.6). As expected, no proof is found if only facts *F1* and *F2* are used to draw the same conclusion. Instead, the model finder Nitpick presents a counterexample when called (Figure 5.7). Also, in line 42, Nitpick finds a model confirming the consistency of the representation of this part of the AIA with SDL (Figure 5.8).

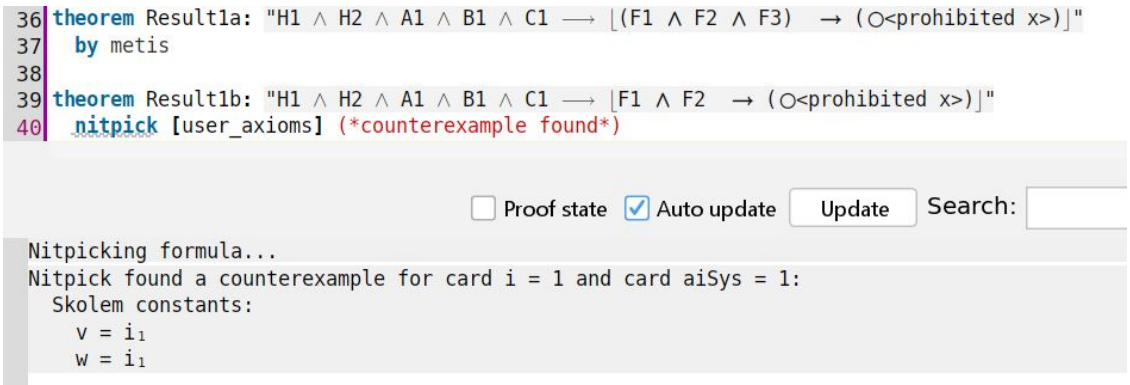


Figure 5.7: Nitpick counterexample Result1b

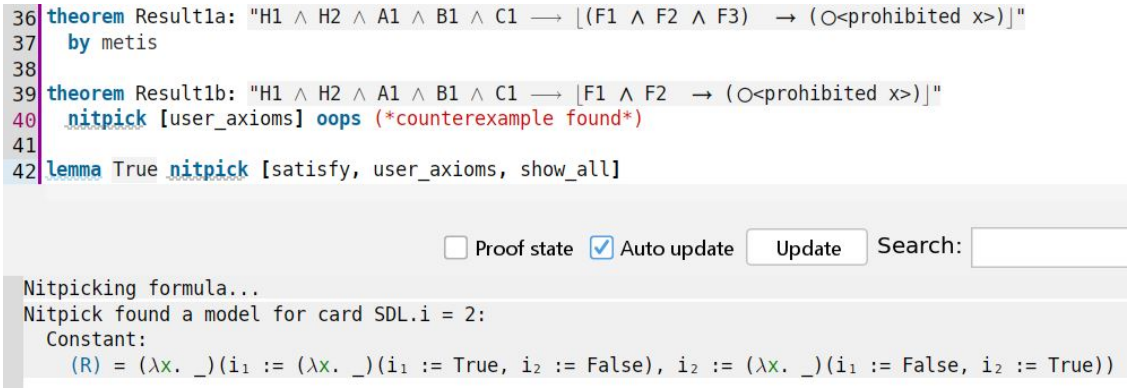


Figure 5.8: Consistency confirmed by Nitpick

To summarize, a representation of the chosen part of the AIA in Isabelle HOL using the embedding of SDL has been achieved. As the tests show, Sledgehammer can reason with the created representation without problems and supports logically correct claims. Even though no prohibitions and permissions appear within the chosen excerpt, these modalities could also be expressed using the SDL embedding (Figure 5.1, lines 22-23).

5.1.3 Remarks on Representing Exceptions

In Chapter 4.3, it was discussed that the ideal solution for formalizing exceptions is using non-monotonic logic, e.g., an Isabelle/HOL embedding of such a logic. However, an alternative solution that uses only the SDL embedding was discovered during this thesis and will be introduced below.

The example from Chapter 4.3 will be used:

> "The following artificial intelligence practices shall be prohibited: the use of 'real-time' remote biometric identification systems in publicly accessible spaces for the purpose of law enforcement, unless and in as far as such use is strictly necessary for one of the following objectives:..." (*Artificial Intelligence Act*, 2021, p. 43).

One can also look at this part as expressing two different implications: One represents the general rule for all systems but the ones covered by the exception and one for those affected

by the exception. To represent this in Isabelle/HOL, the same types and predicates defined previously (Chapter 5.1.2), as well as some additional constants (Figure 5.9) are needed. They can now be utilized to express the exception as explained above (Figure 5.10).

```
consts
  real_time_bioid::"aiSys⇒σ" (*system is a real time bio identification system*)
  use_public_spaces::"aiSys⇒σ" (*system is planned to be used in public spaces*)
  use_law_enforcement::"aiSys⇒σ" (*system is used for law enforcement*)
  strictly_necessary_for::"aiSys⇒purpose⇒σ" (*use of system is strictly necessary for a purpose*)
  consider_consequence::"aiSys⇒consequence⇒σ" (*consider specific consequence of the use of a system*)
  consider_consequence_no_use::"aiSys⇒consequence⇒σ" (*consider specific consequence of not using the system*)
  consider_context::"aiSys⇒σ" (*consider the context in which the system is used*)
  complies_with_bioid_rules::"aiSys⇒σ" (*does system comply or not?*)
```

Figure 5.9: Predicates used to represent Exception

```
abbreviation "D1 ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
(((has_purpose x targeted_search) ∧ (¬(strictly_necessary_for x targeted_search))) ∨
((has_purpose x detection) ∧ (¬(strictly_necessary_for x detection))) ∨ ((has_purpose x prevention) ∧
(¬(strictly_necessary_for x prevention)))) → ○<prohibited x>⌋"
(*implicit: not prohibited*)
abbreviation "D1b ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
(((has_purpose x targeted_search) ∧ (strictly_necessary_for x targeted_search)) ∨
((has_purpose x detection) ∧ (strictly_necessary_for x detection)) ∨ ((has_purpose x prevention) ∧
(strictly_necessary_for x prevention))) → (¬(○<prohibited x>) ∧ (○<high_risk x>))⌋"
abbreviation "A2a ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
  (○<high_risk x>) → ○<consider_consequence_no_use x harm_psychological> ∧
  ○<consider_consequence_no_use x harm_physical>⌋"
abbreviation "A2b ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
  (○<high_risk x>) → ○<consider_consequence x affect_personal_rights> ∧ ○<consider_consequence x affect_personal_freedom>⌋"
abbreviation "A2c ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
  (○<high_risk x>) → ○<complies_with_bioid_rules x>⌋"
```

Figure 5.10: Representation of Exception

The abbreviation *D1a* states the prohibition holding for systems not covered by the exception, whereas *D1b* holds that those falling under the exception should not be prohibited but considered high-risk systems. Abbreviations *A2a, b,* and *c* then summarize the rules that the allowed systems must comply with. To check whether Sledgehammer can reason correctly with this representation of an exception, a test case is constructed (Figure 5.11).

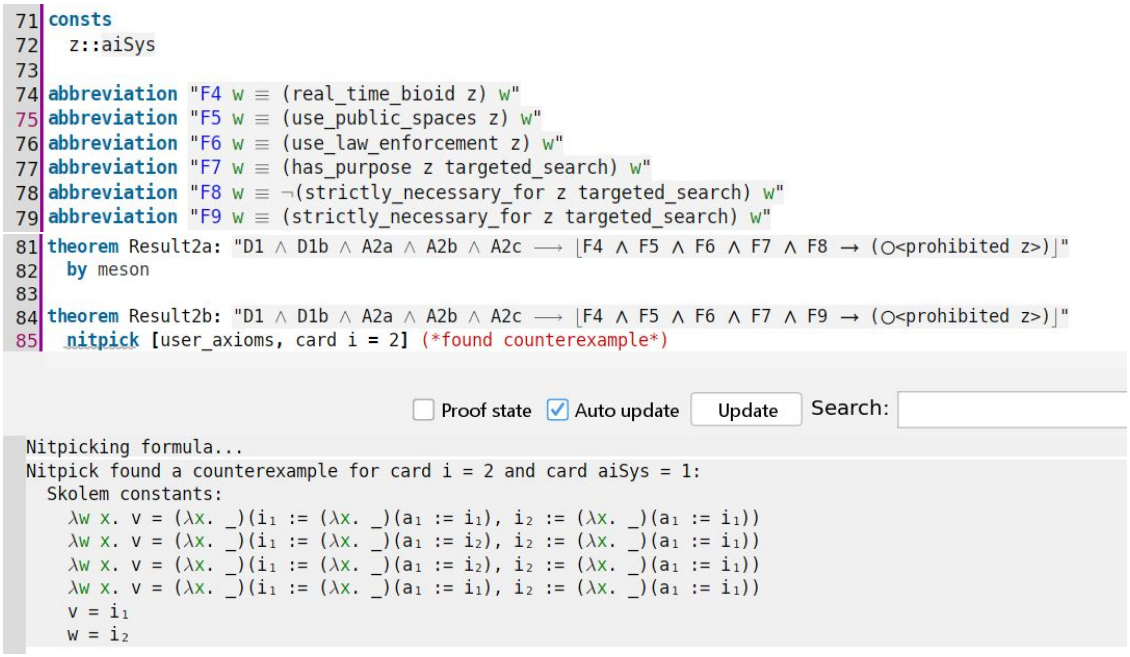


Figure 5.11: Test case Exception

As visible in Figure 5.11, there is one case in which the system z should be prohibited according to the AIA (result2a): If it is a real-time remote biometric identification system used in publicly accessible spaces for law enforcement, has the purpose targeted search and is **not strictly necessary** for this objective. We construct a second case in which everything is as in case one, except for the fact that this time, the system is **strictly necessary** for targeted search, thereby falling under the exception (result2b). For the first lemma, Sledgehammer finds proof, whereas, for the second one, no proof is found, and instead, Nitpick presents a counterexample. This mirrors the desired behavior, showing that the strategy effectively represents the exception. Also, Nitpick finds a model, confirming the representation's consistency with SDL (Figure 5.12).

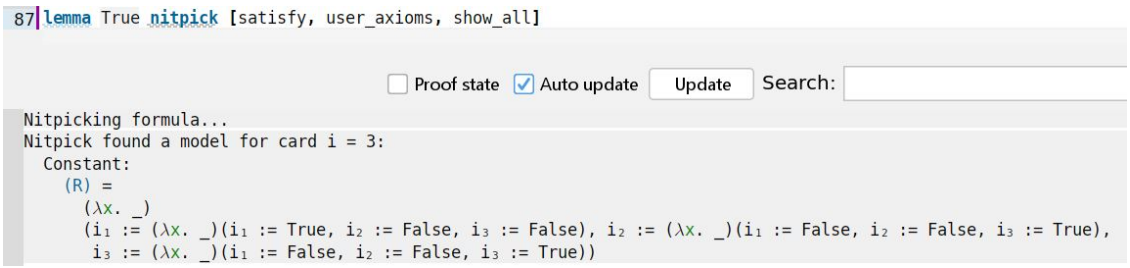


Figure 5.12: Nitpick model found for Exception example

The introduced strategy effectively preserves the relevant information from the AIA and allows for consistent reasoning within Isabelle. However, all exceptions must be explicitly stated. Using non-monotonic logic instead would allow for a way to deal with exceptions without the need to list all of them (Strasser & Antonelli, n.d.), and should be considered for the representation of the AIA in the future.

5.2 Representing Contrary-to-Duty-Obligations

Next, selected CTDs from the AIA will be represented. After Chapter 4.6 has already identified DDL as a logic qualified to express CTDs, this logic and its embedding in Isabelle/HOL will now be introduced briefly.

5.2.1 DDL and its Embedding in Isabelle/HOL

Dyadic Deontic Logic (DDL), as conceptualized by Carmo and Jones, offers a formal system allowing for the expression of conditional obligations, particularly contrary-to-duty scenarios (Carmo & Jones, 2002b). DDL employs dyadic obligations of the form *it ought to be that... if...* to handle such complexities, effectively sidestepping paradoxes like Chisholm's (Carmo & Jones, 2002b).

DDL formulas are constructed in the following way: Each $p^j \in P$ is a DDL formula.

Given two arbitrary DDL formulas i and k, the classical negation $\neg i$ and the classical disjunction $i \vee k$ are DDL formulas. As usual, other logical connectives can be defined using negation and disjunction as primitives (Carmo & Jones, 2002b).

Additionally, there are a few special DDL formulas, namely: $\Box$ a P: P holds in all actual versions of the current world, $\Box$ p P: P holds in all potential versions of the current world, $Oa(P)$: P is an actual obligation, $Op(P)$: P is an ideal/potential obligation, and $O(P/Q)$: P is obligated given that Q (Carmo & Jones, 2002b).

Diving into the semantic, the dyadic operator $O(P/Q)$ is used to represent that in any context (set of worlds) in which Q is true, it is obligated that P (Carmo & Jones, 2002b, p. 282). A function *ob: P(W)* $\rightarrow$ *P(P(W))* is defined, with *P(W)* denoting the powerset of *W*. This function then "picks out for each context the propositions which represent that which is obligatory in that context" (Carmo & Jones, 2002b, p. 282). A sentence of the form $O(P/Q)$ then is true in a model iff in any context X where Q is true and P is possible, P is obligatory (Carmo & Jones, 2002b).

To derive ideal and actual obligations from this, we must differentiate two notions of necessity. First, we will use $\Box$ a to represent actual necessity, meaning that which is unalterable in a concrete context. This meaning is captured by adding a function *av(w)* to our model that picks out for each world w the set of worlds that are actual versions of w, meaning that they make up the relevant context for deriving actual obligations at w. A sentence of the form $\Box$ a P then is true at a world w iff P is true in all actual versions of w (Carmo & Jones, 2002b, p. 283).

Actual obligations are defined as follows: The set of propositions *ob(av(w))* are all propositions representing that which is obligatory in the context *av(w)*. A sentence $Oa(P)$ then is true at a world w if the proposition P is in *ob(av(w))*. Furthermore, it is required that there exists one world in *av(w)* in which P is false since actual obligations may be violated (Carmo & Jones, 2002b, p. 283).

The second notion of necessity $\Box$ p P is used to capture features that "could not have been avoided by the agents concerned, no matter what they had done" (Carmo & Jones,

2002b, p. 283). Based on this, potential or ideal obligations can be defined by introducing another function *pv(w)*. This function selects for a given world w the set of worlds which are potential versions of w, e.g., the relevant context to decide which ideal obligations are relevant at w. A sentence $\Box$ p P then is true at a world w iff P is true at all worlds contained in *pv(w)*. Moreover, for all propositions picked out by *ob(pv(w))*, we can say $Op(P)$, meaning that it is ideally obligatory that P. Again, the truth of $Op(P)$ demands there to be at least one world in *pv(w)* in which P is false, since ideal obligations can also be violated (Carmo & Jones, 2002b, 283f.).

Using these ideas, a DDL model can be defined as a structure M = <W, av, pv, ob, V>, with:

- W being a non-empty set containing all possible worlds,
- V being an evaluation function that assigns a truth set (a set of worlds) to each atomic sentence,
- av: W $\rightarrow$ P(W) being a function that takes a world and maps it to a set of worlds that contains the actual versions of a world, and av(w) $\neq \varnothing$, and
- pv: W $\rightarrow$ P(W) being a function that takes a world and maps it to a set of worlds that contains the potential versions of a world, such that av(w) $\subseteq$ pv(w) and w $\in$ pv(w), and
- ob: P(W) $\rightarrow$ P(P(W)) being a function connecting sets of worlds to sets of sets of worlds which represent the set of propositions that are obligated in context X (Benzmüller et al., 2022; Carmo & Jones, 2002b).

Several conditions must hold for *ob* (where *X, Y, Z* designate arbitrary subsets of *W*):

– $\varnothing \notin \text{ob}(X)$.

– If $Y \cap X = Z \cap X$, then ($Y \in \text{ob}(X)$ if and only if $Z \in \text{ob}(X)$).

– Let $\beta \subseteq \text{ob}(X)$ and $\beta \neq \varnothing$. If $(\cap\beta) \cap X \neq \varnothing$
(where $\cap\beta = \{\text{s} \in \text{S}$; for all $Z \in \beta$, we have $\text{s} \in \text{Z}\}$),
then $(\cap\beta) \in \text{ob}(X)$.

– If $\text{Y} \subseteq \text{X}$ and $\text{Y} \in \text{ob(X)}$ and $\text{X} \subseteq \text{Z}$, then $(\text{Z} \setminus \text{X}) \cup \text{Y} \in \text{ob}(Z)$.

– If $\text{Y} \subseteq \text{X}$ and $\text{Z} \in \text{ob(X)}$ and $\text{Y} \cap \text{Z} \neq \varnothing$, then $\text{Z} \in \text{ob}(Y)$
(Benzmüller et al., 2022; Carmo & Jones, 2002b).

Truth of a formula A relative to a world w in a model M then is determined in the following way:

M, w $\models p^j$	if and only if w $\in \text{V}(p^j)$
M, w $\models \neg i$	if and only if M, w $\not\models i$ (meaning not M, w $\models i$)
M, w $\models i \vee v$	if and only if M, w $\models i$ or M, w $\models v$
M, w $\models \Box i$	if and only if $\text{V}(i) = W$
M, w $\models \Box$ a i	if and only if $\text{av}(w) \subseteq \text{V}(i)$
M, w $\models \Box$ p i	if and only if $\text{pv}(w) \subseteq \text{V}(i)$
M, w $\models O(v \,/\, i)$	if and only if $\text{V}(v) \in \text{ob}(\text{V}(i))$
M, w $\models Oa(i)$	if and only if $\text{V}(i) \in \text{ob}(\text{av}(w))$ and $\text{av}(w) \cap \text{V}(\neg i) \neq \varnothing$
M, w $\models Op(i)$	if and only if $\text{V}(i) \in \text{ob}(\text{pv}(w))$ and $\text{pv}(w) \cap \text{V}(\neg i) \neq \varnothing$

(Benzmüller et al., 2022; Carmo & Jones, 2002b).

A DDL formula i is valid in a model M ($M \models^{DDL} i$), if and only if for all worlds $w \in W$, we have $M, w \models \phi$. It is valid ($\models^{DDL}$ i), if and only if it is valid in every DDL model (Benzmüller et al., 2022; Carmo & Jones, 2002b).

For more detailed investigations into the specifics of DDL, the interested reader may consult the literature (Carmo & Jones, 2002a, 2002b, 2013, 2022).

An embedding of DDL in Isabelle/HOL has already been created and tested by Benzmüller et al. (2022) (Figure 5.13). For further explanations, please refer to the literature (Benzmüller et al., 2022).

```
theory DDL (* DDL: Dyadic Deontic Logic by Carmo and Jones, Benzmüller, Parent Farjami, 2018 *)
  imports types Main
begin

 consts av::"i⇒σ" pv::"i⇒σ" ob::"σ⇒(σ⇒bool)" (*accessibility relations*)
           cw::i (*current world*)

 axiomatization where
  ax_3a: "∀w.∃x. av(w)(x)" and ax_4a: "∀w x. av(w)(x) ⟶ pv(w)(x)" and ax_4b: "∀w. pv(w)(w)" and
  ax_5a: "∀X.¬ob(X)(λx. False)" and
  ax_5b: "∀X Y Z. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (ob(X)(Y) ⟷ ob(X)(Z))" and
  ax_5ca: "∀X β. ((∀Z. β(Z) ⟶ ob(X)(Z)) ∧ (∃Z. β(Z))) ⟶
      (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ ob(X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
  ax_5c: "∀X Y Z. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  ob(X)(Y)  ∧  ob(X)(Z))  ⟶  ob(X)(λw. Y(w) ∧ Z(w)))" and
  ax_5d: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ ob(X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                    ⟶ ob(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
  ax_5e: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ ob(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ ob(Y)(Z)"

 abbreviation ddlneg::γ ("¬_"[52]53) where "¬A ≡ λw. ¬A(w)"
 abbreviation ddland::ϱ (infixr"∧"51) where "A∧B ≡ λw. A(w)∧B(w)"
 abbreviation ddlor::ϱ (infixr"∨"50) where "A∨B ≡ λw. A(w)∨B(w)"
 abbreviation ddlimp::ϱ (infixr"→"49) where "A→B ≡ λw. A(w)⟶B(w)"
 abbreviation ddlequiv::ϱ (infixr"↔"48) where "A↔B ≡ λw. A(w)⟷B(w)"
 abbreviation ddlbox::γ ("□") where "□A ≡ λw.∀v. A(v)"  (*A = (λw. True)*)

 abbreviation ddlboxa::γ ("□ₐ") where "□ₐA ≡ λw. (∀x. av(w)(x) ⟶ A(x))"  (*in all actual worlds*)
 abbreviation ddlboxp::γ ("□ₚ") where "□ₚA ≡ λw. (∀x. pv(w)(x) ⟶ A(x))" (*in all potential worlds*)
 abbreviation ddldia::γ ("◇") where "◇A ≡ ¬□(¬A)"
 abbreviation ddldiaa::γ ("◇ₐ") where "◇ₐA ≡ ¬□ₐ(¬A)"
 abbreviation ddldiap::γ ("◇ₚ") where "◇ₚA ≡ ¬□ₚ(¬A)"

 abbreviation ddlo::ϱ ("O⟨_|_⟩"[52]53) where "O⟨B|A⟩ ≡ λw. ob(A)(B)"  (*it ought to be ψ, given φ *)
 abbreviation ddloa::γ  ("Oₐ") where "OₐA ≡ λw. ob(av(w))(A) ∧ (∃x. av(w)(x) ∧ ¬A(x))" (*actual obligation*)
 abbreviation ddlop::γ  ("Oₚ") where "OₚA ≡ λw. ob(pv(w))(A) ∧ (∃x. pv(w)(x) ∧ ¬A(x))"  (*primary obligation*)
 abbreviation ddltop::σ ("⊤") where "⊤ ≡ λw. True"
 abbreviation ddlbot::σ ("⊥") where "⊥ ≡ λw. False"

(*Possibilist Quantification.*)
 abbreviation ddlforall ("∀") where "∀Φ ≡ λw.∀x. (Φ x w)"
 abbreviation ddlforallB (binder"∀"[8]9) where "∀x. φ(x) ≡ ∀φ"
 abbreviation ddlexists ("∃") where "∃Φ ≡ λw.∃x. (Φ x w)"
 abbreviation ddlexistsB (binder"∃"[8]9) where "∃x. φ(x) ≡ ∃φ"

 abbreviation ddlvalid::"σ ⇒ bool" ("⌊_⌋"[7]105) where "⌊A⌋ ≡ ∀w. A w"   (*Global validity*)
 abbreviation ddlvalidcw::"σ ⇒ bool" ("⌊_⌋ι"[7]105) where "⌊A⌋ι ≡ A cw" (*Local validity (in cw)*)

(* A is obliagtory *)
 abbreviation ddlobl::γ ("O<_>") where "O<A> ≡  O⟨A|⊤⟩"  (*New syntax: A is obligatory.*)

(* Consistency *)
lemma True nitpick [satisfy,user_axioms,show_all] oops

end
```

Figure 5.13: DDL Embedding in Isabelle/HOL

5.2.2 Representing Example CTDs

As explained before, the goal of this section is to represent CTDs from the AIA in DDL. Additionally, it will be shown why SDL is unable to reason with CTDs correctly.

The running example in this chapter appears in Article 16 of the AIA:

> "Providers of high-risk AI systems shall:
> (a) ensure that their high-risk AI systems are compliant with the requirements set out in Chapter 2 of this Tile;
> (...)
> (g) take the necessary corrective actions if the high-risk AI system is not in conformity with the requirements set out in Chapter 2 of this Tile;
> (h) inform the national competent authorities of the Member States in which they made the AI system available or put it into service and, where applicable, the notified body of the non-compliance and any corrective actions taken;
> (...)" (*Artificial Intelligence Act*, 2021, 52f.).

To easier recognize the CTD here, the sentences will be reformulated to suit the typical CTD structure (Chapter 4.6). The fact that the obligations in this article are agentive for the provider is disregarded for now. Additionally, it is focused on the obligation to inform national competent authorities, meaning that the duty to take corrective actions is ignored. Finally, point three in the reformulation is not mentioned explicitly but can be inferred from the given passage. The resulting CTD then looks as follows:

> 1. High-risk AI systems shall be compliant with the requirements set out in Chapter 2 of this tile.
> 2. If a high-risk AI system is not in conformity, the national competent authorities shall be informed of the non-compliance.
> 3. If a high-risk AI system is in conformity, the national competent authorities shall not be informed.
> 4. A high-risk system x is not in conformity with the requirements.

In the following, the DDL representation of this CTD will be discussed (Figure 5.14).

```
theory "16_DDLonly"
  imports
  DDL
begin

consts
  compliance_req_chap2::"aiSys⇒σ"
  inform_com_auth::"aiSys⇒σ"
  kill_everyone::"aiSys⇒σ"

(*CTD example DDL:*)
consts
  l::aiSys

(*interesting part: CTD; Trying to create the typical structure*)
axiomatization where
A0: "⌊(high_risk l)⌋" and
A1: "⌊∀x::aiSys.(high_risk x) ⟶ ○<(compliance_req_chap2 x)>⌋" and
A2: "⌊∀x::aiSys. ¬(compliance_req_chap2 x) ∧ (high_risk x) ⟶ ○<(inform_com_auth x)>⌋" and
(*implicit: If the compliance with the requirements is a given, the provider is obligated to not inform authorities
of non-compliance (since that would make no sense*)
A3: "⌊∀x::aiSys. ○<(compliance_req_chap2 x) ∧ (high_risk x) ⟶ (¬(inform_com_auth x))>⌋" and
Situation: "⌊¬ (compliance_req_chap2 l)⌋ι"

(***Some Experiments***)
lemma True nitpick [satisfy, user_axioms, show_all] oops(*Consistency-check: Nitpick finds a model.*)



lemma "⌊○<(inform_com_auth l)>⌋ι" using A0 A2 Situation by auto
lemma "⌊○<¬(inform_com_auth l)>⌋ι" nitpick [user_axioms] oops (*counterexample found*)
lemma "⌊○<(kill_everyone l)>⌋ι" try oops

end
```

Figure 5.14: Isabelle representation Article 16 in DDL

In lines 2-3, the types file and the embedding of DDL are imported. The only relevant import from the types file is the type definition *aiSys* for AI systems and the constant *high_risk* that evaluates to true if a system is classified as high-risk. Lines 7-8 then define two more constants: One to indicate whether high-risk AI systems conform with the requirements, and one to show whether the responsible national competent authorities are informed about an AI system. The constant declared in line 9 can be ignored for now; it is used for experimentation later.

Afterward, a constant l denoting a particular AI system is declared in line 13. Line 17 simply states that the given AI system l is indeed classified as high-risk. The next few lines contain the actual CTD: *A1* matches phrase one, *A2* phrase two, *A3* phrase three, and what is called *Situation* in Isabelle expresses phrase four (lines 18-23). To express the current situation, we use the notion of local validity, meaning that a statement is valid in the current world (Benzmüller et al., 2022).

Some experiments will test this representation. A first consistency check with Nitpick in line 26 succeeds (Figure 5.15).

Figure 5.15: Consistency confirmed via Nitpick

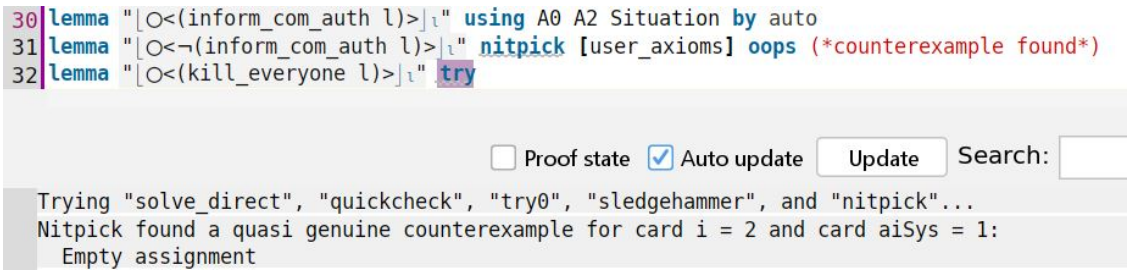


Figure 5.16: No false statements proven

Next, different lemmas are constructed to control whether Sledgehammer can reason with the formalized CTD (Figure 5.16). Fortunately, this leads to the desired results: Only the lemma in line 30 can be proven, which states that it is an obligation in the current world to inform the authorities about system l. This aligns with the rules specified in the AIA for the given situation (*Artificial Intelligence Act*, 2021, 52f.). On the other hand, no proof is found for the lemma stating the opposite in line 31, and instead, Nitpick finds a counterexample. Additionally, no proof is found for the abstruse claim that the AI system l should kill everyone since the axioms do not lead to inconsistencies when formalized in DDL.

To display the contrast between SDL and DDL, the same file is created again, but this time, the SDL embedding is imported (Figure 5.17). This easy switch of the two logics is possible due to line 48 in the DDL embedding (Figure 5.13), where a syntax is introduced that matches the syntax of SDL.

Using SDL produces different results: The consistency check in line 26 fails, and Nitpick does not find a model. In line 27, the axioms even allow us to infer Falsum using Sledgehammer (Figure 5.18). This is a serious problem because, from inconsistency, everything can be proven. We can see that in the lemmas in lines 30 and 31. Here, Sledgehammer finds proof supporting the claim that the national competent authorities should be informed about system l, as well as for the opposite claim. Line 32 is even more concerning. Here, Sledgehammer proves that it is obligatory for the AI system l to kill everyone (Figure 5.19).

```
theory "16_SDLonly"
  imports
  SDL
begin

consts
  compliance_req_chap2::"aiSys⇒σ"
  inform_com_auth::"aiSys⇒σ"
  kill_everyone::"aiSys⇒σ"

(*CTD example SDL:*)
consts
  l::aiSys

(*interesting part: CTD; Trying to create the typical structure*)
axiomatization where
A0: "⌊(high_risk l)⌋" and
A1: "⌊∀x::aiSys.(high_risk x) → ○<(compliance_req_chap2 x)>⌋" and
A2: "⌊∀x::aiSys. ¬(compliance_req_chap2 x) ∧ (high_risk x) → ○<(inform_com_auth x)>⌋" and
(*implicit: If the compliance with the requirements is a given, the provider is obligated to not inform authorities
of non-compliance (since that would make no sense*)
A3: "⌊∀x::aiSys. ○<(compliance_req_chap2 x) ∧ (high_risk x) → (¬(inform_com_auth x))>⌋" and
Situation: "⌊¬ (compliance_req_chap2 l)⌋ι"

(***Some Experiments***)
lemma True nitpick [satisfy, user_axioms] oops (*Consistency-check: Nitpick finds no model!*)
lemma False sledgehammer oops


lemma "⌊○<(inform_com_auth l)>⌋ι" using A0 A2 Situation by auto
lemma "⌊○<¬(inform_com_auth l)>⌋ι" by (simp add: A0 A1 A3)
lemma "⌊○<(kill_everyone l)>⌋ι" by (metis A0 A1 A2 A3 Situation)

end
```

Figure 5.17: Isabelle representation Article 16 in SDL

This behavior is problematic and not desirable in a faithful representation. It clearly outlines why DDL is needed to represent and consistently reason with AIA paragraphs containing CTDs.

Fortunately, anything that can be expressed in SDL can also be expressed in DDL (Dov Gabbay et al., 2013), meaning that all parts of the AIA containing the modalities discussed so far (obligations, permissions, prohibitions, and CTDs) can faithfully be represented using the introduced DDL embedding. A representation of the examples from Chapter 5.1 in DDL can be found in Appendix Eight.

```
25 (***Some Experiments***)
26 lemma True nitpick [satisfy, user_axioms] oops (*Consistency-check: Nitpick finds no model!*)
27 lemma False sledgehammer
```

Proof state ☑ Auto update | Update | Search:

```
Sledgehammering...
cvc4 found a proof...
verit found a proof...
cvc4: Try this: by (meson A0 A1 A2 A3 Situation seriality) (15 ms)
cvc4 found a proof...
cvc4 found a proof...
cvc4: Try this: by (meson A0 A1 A2 A3 Situation seriality) (46 ms)
verit: Try this: by (metis A0 A1 A2 A3 D Situation) (0.0 ms)
cvc4: Try this: by (meson A0 A1 A2 A3 Situation seriality) (0.0 ms)
QED
```

Figure 5.18: Proving Falsum using Sledgehammer

```
30 lemma "⌊○<(inform_com_auth l)>⌋ι" using A0 A2 Situation by auto
31 lemma "⌊○<¬(inform_com_auth l)>⌋ι" by (simp add: A0 A1 A3)
32 lemma "⌊○<(kill_everyone l)>⌋ι" try
```

Proof state ☑ Auto update | Update | Search:

```
Trying "solve_direct", "quickcheck", "try0", "sledgehammer", and "nitpick"...
cvc4: Try this: by (meson A0 A1 A2 A3 Situation) (0.0 ms)
verit: Try this: by (meson A0 A1 A2 A3 Situation) (15 ms)
cvc4: Try this: by (meson A0 A1 A2 A3 Situation) (0.0 ms)
spass: Try this: using A0 A1 A2 A3 Situation by blast (46 ms)
cvc4: Try this: by (meson A0 A1 A2 A3 Situation) (0.0 ms)
cvc4: Try this: using A0 A1 A2 A3 Situation by blast (0.0 ms)
cvc4: Try this: using A0 A1 A2 A3 Situation by blast (31 ms)
cvc4: Try this: using A0 A1 A2 A3 Situation by blast (31 ms)
vampire: Try this: using A0 A1 A2 A3 Situation by blast (31 ms)
spass: Try this: by (metis A0 A1 A2 A3 Situation) (0.0 ms)
z3: Try this: by (meson A0 A1 A2 A3 Situation) (15 ms)
QED
```

Figure 5.19: Consequences of inconsistency

5.3 Representing Agency and Agent-Based Obligations

In this section, the representation of agency will be discussed. As indicated in Chapter 4.5, the branch of STIT logics is a good choice for this purpose. More precisely, a deontic version of STIT theory is needed since the AIA contains not only actions but also agentive obligations that must be formalized. Temporal Deontic STIT (TDS) Logic as introduced by van Berkel and Lyon in 2019 extends STIT logic with an *ought-to-do* operator $\circ_i\phi$, making it a suitable candidate (van Berkel & Lyon, 2019).

5.3.1 Temporal Deontic STIT Logic

TDS Logic combines Lorini's Temporal STIT (T-STIT) Logic (2013) with atemporal deontic STIT Logic (Murakami, 2005). Its language consists of a countably set of propositional variables *Var*, a finite set of agents *Ag*, and the following formulas ϕ:

$$p \text{ ranging over Var, } \neg\phi,\ \phi \wedge \phi,\ [i]\phi \text{ with } i \text{ ranging over Ag, } [Ag]\phi,\ \Box\phi,\ [G]\phi,\ [H]\phi,\ \circ_i \phi$$

As usual, the other boolean constructions $(\vee, \rightarrow, \leftrightarrow)$ can be defined based on negation and conjunction, whereas $\top$ and $\bot$ are defined as $p \vee \neg p$ and $p \wedge \neg p$ respectively (van Berkel & Lyon, 2019).

The individual STIT operator $[i]$ translates to *agent i ought to see to it that*, and the group STIT operator $[Ag]$ to *the group of all agents Ag ought to see to it that*. To express that *agent i ought to see to it that*, the ought-to-do operator $\circ_i$ can be used. The Box operator $\Box\phi$ expresses that ϕ is always true, independent of how all agents act. G and H are temporal operators that can be used to make statements about the strict future and the strict past, with $[G]\phi$ meaning that ϕ will always be true in the future, and $[H]\phi$ meaning that ϕ has always been true in the past (van Berkel & Lyon, 2019).

Originally, STIT theory is set against a semantic framework of indeterminism, particularly the theory of Branching Time and Agent Choices (BT+AC). BT structures are a

tree-like ordering over a set of moments. The AC function then maps each moment into a history consisting of linearly ordered moments so that the equivalence classes of the partition compose the possible choices for an agent at each moment (Lorini, 2013). Since BT+AC semantics can be simulated in possible world semantics, they will not be elaborated here. For a detailed account of BT+AC, the reader may refer to Belnap et al. (2001).

Interpreted in Kripke-style semantics, TDS Logic uses multiple accessibility relations governed by specific constraints:

- $\mathbf{R}_{\Box}$(w), which returns the set of all worlds that are alternatives to w,
- $\mathbf{R}_{i}$(w), which returns the set of worlds that are forced by the choice of agent i at world w,
- $\mathbf{R}_{Ag}$(w), which returns the set of worlds that are forced by the choices of all agents in the set Ag at w,
- $\mathbf{R}_{\circ_i}$**(w)**, which returns the set of ideal worlds that an agent i ought to enforce with their choice at w, and
- $\mathbf{R}_{G}$(w) and $\mathbf{R}_{H}$(w), which return all worlds that are in the strict future [past] of w (van Berkel & Lyon, 2019).

A TDS frame can then be described as a tuple F = (W, $R_{\Box}$, R_i | i ∈ Ag, R_{Ag}, R_G, R_H, $R_{\circ_i}$ | i ∈ Ag) where W is a non-empty set of possible worlds, and

- "for all i ∈ Ag, $R_{\Box}$, R_i, R_{Ag} ⊂ W x W are equivalence relations such that:
  (C1) $R_i \subseteq R_{\Box}$;
  (C2) for all u_1, ..., u_n ∈ W: if (u_i, u_j) ∈ $R_{\Box}$ for all i, j ∈ 1, ..., n then $\bigcap_{1 \leq i \leq n} R_i(u_i) \neq \emptyset$;
  (C3) for all w ∈ W: $R_{Ag}(w) = \bigcap_{i \in Ag} R_i(w)$;
- R_G is a transitive and serial binary relation and R_H is the converse of R_G, such that
  (T4) for all w, v, u ∈ W: if v, u, ∈ R_G(w) then u ∈ R_G(v) or v ∈ R_G(u) or

u = v
(T5) for all w, v, u ∈ W: if v, u ∈ R_H(w) then u ∈ R_H(v) or v ∈ R_H(u) or u = v;
(T6) $R_G \circ R_\Box \subseteq R_{Ag} \circ R_G$;
(T7) for all w ∈ W: if v ∈ $R_\Box$(w) then v ≠ R_G(w);

- For all i ∈ Ag, $R_{\circ_i} \subset$ W x W are binary relations such that:
(D8) $R_{\circ_i} \subseteq R_\Box$.
(D9) For all $w \in W$, there exists a $v \in W$ such that $v \in R_{\Box(w)}$ and for all $u \in W$, if $u \in R_i(v)$, then $u \in R_{\circ_i}(w)$.
(D10) For all $w, v, u, z \in W$, if $v, u \in R_\Box(w)$ and $z \in R_{\circ_i}(u)$, then $z \in R_{\circ_i}(v)$.
(D11) For all $w, v \in W$, if $v \in R_{\circ_i}(w)$ then there exists $u \in W$ such that $u \in R_\Box(w)$and $v \in R_i(u)$, and for all $z \in W$, if $z \in R_i(u)$ then $z \in R_{\circ_i}(w)$" (van Berkel & Lyon, 2019, p. 3)

A TDS model is a tuple M = (F,V), with F being a TDS frame as explained above, and V being a valuation function for propositional variables such that V: Var $\longrightarrow$ P(W).

The truth of a formula ϕ in a TDS model M is determined based on the following conditions:

$$
\begin{aligned}
&\text{"1. } M, w \models p \text{ iff } w \in V(p)\\
&\text{2. } M, w \models \neg\phi \text{ iff } M, w \not\models \phi\\
&\text{3. } M, w \models \phi \wedge \upsilon \text{ iff } M, w \models \phi \text{ and } M, w \models \upsilon\\
&\text{4. } M, w \models \Box\phi\\
&\text{iff } \forall v \in R_\Box(w) : M, v \models \phi\\
&\text{5. } M, w \models [i]\phi \text{ iff } \forall v \in R_i(w) : M, v \models \phi\\
&\text{6. } M, w \models \circ_i\phi \text{ iff } \forall u \in R_{\circ_i}(w), M, u \models \phi\\
&\text{7. } M, w \models [Ag]\phi \text{ iff } \forall v \in R_{Ag}(w) : M, v \models \phi\\
&\text{8. } M, w \models [G]\phi \text{ iff } \forall v \in R_G(w) : M, v \models \phi\\
&\text{9. } M, w \models [H]\phi \text{ iff } \forall v \in R_H(w) : M, v \models \phi\text{"}\\
&\text{(van Berkel \& Lyon, 2019, p. 4).}
\end{aligned}
$$

As visible in the truth conditions, both the individual and the group STIT operators employ Chellas notion of a STIT operator (Chellas, 1992; van Berkel & Lyon, 2019). Based on this, it is possible to define the deliberative STIT operator as follows: $[i]_d\phi$ if and only if

$[i]\phi \wedge \Diamond\neg\phi$ (J. Horty, 1989; van Berkel & Lyon, 2019; Von Kutschera, 1986). Similarly, the deliberative ought operator can be defined as: $\circ_{id}\phi$ if and only if $\circ_i\phi \wedge \Diamond\neg\phi$ (van Berkel & Lyon, 2019). For more details on the different notions of STIT, please refer to the literature, for example, J. F. Horty and Belnap (1995).

A TDS formula ϕ is TDS valid ($\models_{TDS} \phi$), if and only if for every TDS model M and for every world w in M we have $M, w \models \phi$. ϕ is satisfiable in TDS iff $\neg\phi$ is not TDS valid (Blackburn et al., 2001; van Berkel & Lyon, 2019). For more details on the relations in TDS Logic and the constraints on them, please refer to van Berkel and Lyon (2019).

5.3.1.1 Embedding TDS Logic in Isabelle/HOL

The following section will introduce an embedding of TDS Logic in Isabelle/HOL. The work by Benzmüller et al. on embedding DDL in Isabelle/HOL serves as an orientation since DDL, like TDS Logic, employs a Kripke-style possible world semantics that must be sustained in the HOL embedding (Benzmüller et al., 2022).

The full TDS embedding can be found in Figures 5.20, 5.21 and 5.22. Explanations are provided below.

```
1  theory Tstit_Deontic_clean (*TDS logic*)
2    imports Main
3  begin
4
5  declare [[show_types]]
6  nitpick_params[user_axioms, show_all, format=2] (*global parameter setting for nitpick*)
7
8  typedecl i (*possible world*)
9  type_synonym σ = "(i⇒bool)"
10 type_synonym γ = "σ⇒σ"
11 type_synonym ϱ = "σ⇒σ⇒σ"
12 type_synonym δ = "i⇒i⇒bool" (* type of accessibility relations between worlds *)
13
14 datatype ag = a1 | a2  (* datatype of mutually different agents; we provide 2, more can be added as needed *)
15
16 type_synonym ω = "ag⇒i⇒i⇒bool"   (* type of agent dependent accessibility relations between worlds *)
17 type_synonym ν = "(ag⇒bool)⇒i⇒i⇒bool" (* type of set-of-agents dependent accessibility relations between worlds *)
18
19 consts
20   cw::i (*current world*)
21
22   RBox::δ (*worlds that are alternatives to each other: if (w, w1) then w1 is an alternative to w*)
23   R_ag::ω (*worlds that are actual choices for agent a, set of alternatives that are forced by agents i choice or action at world w*)
24   R_set::ν (*worlds that are actual choices for the set of agents Ag*)
25   R_ag_ought::ω (*set of alternatives that agent a ought to chose at moment m*)
26
27   RG::δ (*all worlds that are the strict future of world w: (w, w1) means that w1 is the strict future of w*)
28   RH::δ (*all worlds that are the strict past of world w: (w, w1) means that w1 is the strict past of w*)
29
30   Ag::"ag⇒bool" (*set of all agents*)
31
32 definition Inv::"δ ⇒ δ" ("Inv _") where
33   "Inv R ≡ λy x. R x y"
34
35 lemma True nitpick [satisfy, user_axioms, show_all] oops (*empty assignment*)
36
37 axiomatization where
38   a1Set: "Ag a1" and
39   a2Set: "Ag a2"
40
```

Figure 5.20: Embedding TDS Logic in Isabelle/HOL part 1

First, a type i for possible worlds is introduced (line 8). This is important since TDS formulas (as usual in Kripke-style semantics) are always evaluated based on a given world. Consequently, the type for TDS formulas is $i \rightarrow bool$, abbreviated as σ (line 9). Next, the types γ, ρ, and δ needed for embedding the logical connectives are declared (lines 10-12). Line 14 declares the datatype for agents *ag* and names two mutually different agents *a1* and *a2* (more can be added as needed). Finally, two more types ω for agent-dependent accessibility relations between worlds and ν for accessibility relations depending on the set of agents are defined (lines 16-17).

```
axiomatization where
 (*reflexivity, symmetry, and transitivity for all equivalence relations*)
  accReR_ag:  "∀a w. (R_ag a) w w" and
  accSymR_ag: "∀a w v. (R_ag a) w v ⟶ (R_ag a) v w" and
  accTraR_ag: "∀a w v u. ((R_ag a) w v ∧ (R_ag a) v u) ⟶ (R_ag a) w u" and

  accReRBox:  "∀w. RBox w w" and
  accSymRBox: "∀w v. RBox w v ⟶ RBox v w" and
  accTraRBox: "∀w v u. (RBox w v ∧ RBox v u) ⟶ RBox w u" and

  accReR_set:  "∀s w. (R_set s) w w" and
  accSymR_set: "∀s w v. (R_set s) w v ⟶ (R_set s) v w" and
  accTraR_set: "∀s w v u. ((R_set s) w v ∧ (R_set s) v u) ⟶ (R_set s) w u" and

  RG_serial: "(∀x. (∃y. (RG x y)))" and (*seriality of RG*)
  RG_trans: "(∀x y z. (RG x y) ∧ (RG y z) ⟶ (RG x z))" and (*transitivity of RG*)
  Inv: "(Inv RG) = RH" and (*RH is the inverse of RG*)

  C1: "∀a w1 w2. (R_ag a) w1 w2 ⟶ RBox w1 w2" and (*agents can only choose between alternatives*)
  (*independence of agents/choices → if w1 and w2 are alternatives to each other, there exists a world w which is
  part of the actual choice of all agents → see tests*)
  C2: "∀w1 w2. (RBox w1 w2) ⟶ (∃w. ∀a. (R_ag a) w1 w)" and
  (*independence of agents/choices → if w1 and w2 are alternatives to each other, there exists a world w which is
  part of the actual choice of all agents*)
  C3: "∀S w1 w2. ((R_set S) w1 w2) ⟶ (∀a. S a ⟶ (R_ag a) w1 w2)" and (*choices of agents in group Agt are
  made up of the choices of the intersection of choices of each individual agent*)

  T4: "∀u v w. ((RG w u) ∧ (RG w v)) ⟶ ((RG v u) ∨ (RG u v) ∨ u = v)" and (*future*)
  T5: "∀u v w. ((RH w u) ∧ (RH w v)) ⟶ ((RH v u) ∨ (RH u v) ∨ u = v)" and (*past*)
  (* If v is in the future of w and u and v are in the same moment,  then there exists an alternative z
  in the collective choice of all agents at w such that u is in the future of z.*)
  T6: "∀v w u S. (RG w v) ∧ (RBox v u) ⟶ (∃z. ((R_set S) w z) ∧ (RG z u))" and
  T7: "∀w v. ((RBox w v)) ⟶ ¬(RG w v)" and (*if worlds are in the same moment, they can't be in each others future*)

  (*ideal worlds accessible at a moment are alternatives to the current world*)
  D8: "∀a. ∀w v. ((R_ag_ought a) w v) ⟶ (RBox w v)" and
  (*at every moment for each agent there is a choice available that is an ideal choice*)
  D9: "∀a. ∀w. (∃v. (RBox w v) ∧ (∀u. ((R_ag a) v u) ⟶ ((R_ag_ought a) w u)))" and
  (*for each agent, if a world is ideal from the perspective of a particular world at a moment, that world is ideal from
  the perspective of any world at that same moment, ideal worlds are settled upon moments*)
  D10: "∀a. ∀w v u z. (RBox w v) ∧ (RBox w u) ∧ ((R_ag_ought a) u z) ⟶ ((R_ag_ought a) v z)" and
  (*Every ideal world extends to a complete ideal choice, no choice contains both ideal and non-ideal worlds*)
  D11: "∀a. ∀w v. ((R_ag_ought a) w v) ⟶ (∃u. (RBox w u) ∧ ((R_ag a) u v) ∧ (∀z. ((R_ag a) u z) ⟶ ((R_ag_ought a) w z)))"

  (*Logical connectives lifted*)
  abbreviation tdsNot::γ ("¬_") where "¬A ≡ λw. ¬A(w)"
  abbreviation tdsAnd::ϱ ("_∧_") where "A∧B ≡ λw. A(w)∧B(w)"
  abbreviation tdsOr::ϱ ("_ ∨ _") where "A∨B ≡ λw. A(w)∨B(w)"
  abbreviation tdsImp::ϱ ("_⟶_") where "A⟶B ≡ λw. A(w)⟶B(w)"
  abbreviation tdsEquiv::ϱ ("_⟷_") where "A⟷B ≡ λw. A(w)⟷B(w)"
  abbreviation tdsBox::γ ("□") where "□A ≡ λw. ∀v. A(v)"
  abbreviation tdsDia::γ ("◇") where "◇A ≡ ¬□(¬A)"
  abbreviation tdsTop::σ ("⊤") where "⊤ ≡ λw. True"
  abbreviation tdsBot::σ ("⊥") where "⊥ ≡ λw. False"

  (*Operators*)
  abbreviation tdsCstit::"ag⇒γ" ("[_] _") where "[i] A ≡ λw. (∀y. ((R_ag i) w y) ⟶ (A y))"  (*Chellas Stit*)
  abbreviation tdsCstitPoss::"ag⇒γ" ("<_>_") where "<i> A ≡ ¬([i] (¬ A))" (*Possibility Group Chellas stit*)
  abbreviation tdsCstitGr::"γ" ("[Ag]_") where "[Ag] A ≡ λw. (∀v. ((R_set Ag) w v) ⟶ (A v))" (*Chellas stit group*)
  abbreviation tdsCstitGrPoss::"γ" ("<Ag>_") where "<Ag> A ≡ ¬([Ag] (¬ A))"
  abbreviation tdsDstit::"ag⇒γ" ("[_]d _") where "[i]d A ≡ ([i]A) ∧ ¬(□A)" (*Dstit*)
  abbreviation tdsDstitPoss::"ag⇒γ" ("<_>d _") where "<i>d A ≡ ¬ ([i]d (¬ A))" (*Dstit Poss*)
  abbreviation tdsDstitGr::γ ("[Ag]d_") where "[Ag]d A ≡ ([Ag] A) ∧ (¬(□A))" (*Dstit group*)
  abbreviation tdsDstitGrPoss::γ ("<Ag>d_") where "<Ag>d A ≡ ¬([Ag]d(¬ A))"
  abbreviation tdsOughtToDo::"ag⇒γ" ("⊗ _ _") where "⊗ i A ≡ λw. (∀v. ((R_ag_ought i) w v) ⟶ (A v))" (*OughtToDo Operator*)
  abbreviation tdsOughtToDoD::"ag⇒γ" ("⊗d _ _") where "⊗d i A ≡ (⊗ i A) ∧ (◇ ¬ A)" (*OughtToDo Operator*)
```

Figure 5.21: Embedding TDS Logic in Isabelle/HOL part 2

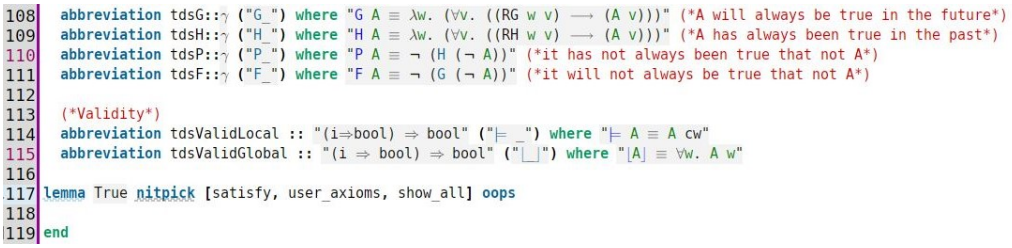


Figure 5.22: Embedding of TDS Logic in Isabelle/HOL part 3

Next, some constants are declared. The first one denotes the current world as *cw* (line 20). This is followed by the different accessibility relations that are defined as functions in Isabelle in lines 22-28: *RBox* (worlds that are alternatives to each other), *R_ag* (actual choices for the agent given as an argument), *R_set* (actual choices for the set of all agents Agt), *R_ag_ought* (set of ideal worlds that agent a ought to choose at moment m), *RG* (strict future), and *RH* (strict past). They are defined that way to avoid set notation, which makes automation inefficient.

Line 30 defines the set of agents *Ag*, which contains *a1* and *a2* as specified in lines 378 and 39. An inverse function, which is required for the axiomatization, is defined in lines 32 and 33.

Afterwards, the constraints for the accessibility relations as defined by van Berkel and Lyon (2019) are stated as axioms. First, the properties of equivalence relations, namely reflexivity, symmetry, and transitivity, are claimed for *R_ag, R_set*, and *RBox* (lines 42-53). Lines 55 and 56 define seriality, and transitivity is established for *RG*, whereas line 57 identifies the relation *RH* as the inverse of *RG*.

Next, the constraints governing the accessibility relations are defined in lines 59-83 (see Chapter 5.3.1, constraints C1 to D11). Note that the constraints in this Isabelle file differ from the original definition of van Berkel and Lyon (2019) since we have defined the accessibility relations as functions. However, the meaning is identical.

The type-lifted TDS logical connectives are declared as abbreviations in lines 85-94. To avoid confusion with the usual connectives of HOL, the TDS connectives are printed in **bold**.

Another set of abbreviations declares the TDS operators using the truth conditions for TDS formulas ϕ as described in the previous chapter: Lines 97-100 define the individual and group Chellas stit operators and their duals, while lines 101-104 define the individual and group deliberative stit operators and their duals. Lines 105-106 then declare the ought-to-do and deliberative ought-to-do operators.

The abbreviations for the lifted tense operators G and H and their duals P and F (van Berkel & Lyon, 2019, p. 2) are introduced in lines 108-111. Finally, local and global validity are defined in lines 114-115.

5.3.1.2 Testing the TDS Embedding

Definition 4 (Axiomatization of TDS). *For each $i \in Ag$ we have,*

A0 *All propositional tautologies.*
A1 $\Box(\phi \to \psi) \to (\Box\phi \to \Box\psi)$,
A2 $\Box\phi \to \phi$
A3 $\Diamond\phi \to \Box\Diamond\phi$
A4 $[i](\phi \to \psi) \to ([i]\phi \to [i]\psi)$
A5 $[i]\phi \to \phi$
A6 $\langle i\rangle\phi \to [i]\langle i\rangle\phi$
A7 $[Ag](\phi \to \psi) \to ([Ag]\phi \to [Ag]\psi)$
A8 $[Ag]\phi \to \phi$
A9 $\langle Ag\rangle\phi \to [Ag]\langle Ag\rangle\phi$
A10 $\bigwedge_{0\leq i\leq n} \Diamond[i]\phi_k \to \Diamond \bigwedge_{0\leq i\leq n}[i]\phi_k$
A11 $\bigwedge_{1\leq i\leq n}[i]\phi_i \to [Ag]\bigwedge_{1\leq i\leq n}\phi_i$
A12 $\otimes_i(\phi \to \psi) \to (\otimes_i\phi \to \otimes_i\psi)$
A13 $\Box\phi \to ([i]\phi \wedge \otimes_i\phi)$
A14 $\otimes_i\phi \to \Diamond[i]\phi$
A15 $\Diamond \otimes_i \phi \to \Box \otimes_i \phi$
A16 $\Box([i]\phi \to [i]\psi) \to (\otimes_i\phi \to \otimes_i\psi)$
A17 $\mathsf{G}(\phi \to \psi) \to (\mathsf{G}\phi \to \mathsf{G}\psi)$
A18 $\mathsf{G}\phi \to \mathsf{GG}\phi$
A19 $\mathsf{G}\phi \to \mathsf{F}\phi$
A20 $\mathsf{H}(\phi \to \psi) \to (\mathsf{H}\phi \to \mathsf{H}\psi)$
A21 $\phi \to \mathsf{GP}\phi$
A22 $\phi \to \mathsf{HF}\phi$
A23 $\mathsf{FP}\phi \to \mathsf{P}\phi \vee \phi \vee \mathsf{F}\phi$
A24 $\mathsf{PF}\phi \to \mathsf{P}\phi \vee \phi \vee \mathsf{F}\phi$
A25 $\mathsf{F}\Diamond\phi \to \langle Ag\rangle\mathsf{F}\phi$
R0 $\vdash_{\mathsf{TDS}}(\psi \to \phi)$ *and* $\vdash_{\mathsf{TDS}}\psi$ *implies* $\vdash_{\mathsf{TDS}}\phi$
R1 $\vdash_{\mathsf{TDS}}\phi$ *implies* $\vdash_{\mathsf{TDS}}[\alpha]\phi$, $[\alpha] \in \{\Box, \mathsf{G}, \mathsf{H}\}$
R2 $\vdash_{\mathsf{TDS}} (\Box\neg p \wedge \Box(\mathsf{G}p \wedge \mathsf{H}p)) \to \phi$ *implies* $\vdash_{\mathsf{TDS}}\phi$, *given* $p \notin \phi$

Figure 5.23: Axiomatisation of TDS (van Berkel & Lyon, 2019, p. 5)

Several axioms for TDS Logic (Figure 5.23) have been specified by van Berkel and Lyon, which will be tested in the following section. We will formulate the axioms as lemmas so that they can be proven using the Sledgehammer tool or disproven with counter models generated by Nitpick. To carry out the tests, another Isabelle file that imports the TDS embedding is

created (Figures 5.24).

```
1  theory Tstit_Deontic_test_clean  (*TDS*)
2    imports Main Tstit_Deontic_clean
3  begin
4
5  (*Tautologies of classical propositional calculus*)
6  lemma Identity: "⌊A⌋ ⟹ ⌊A⌋" by simp
7  lemma NonContradiction: "⌊¬ (A ∧ (¬A))⌋" by simp
8  lemma ExcludedMiddle: "⌊A ∨ (¬A)⌋" by simp
9  lemma DoubleNegation: "⌊(¬ (¬ A)) → A⌋" by simp
10 lemma Implication: "⌊A → A⌋" by simp
11 lemma DeMorgan1: "⌊(¬(A ∧ B)) ↔ ((¬ A) ∨ (¬ B))⌋" by simp
12 lemma DeMorgan2:  "⌊(¬(A ∨ B)) ↔ ((¬A) ∧ (¬B))⌋" by simp
13 lemma Contrapositive: "⌊(A → B) ↔ ((¬B) → (¬A))⌋" by blast
14
15 (*other axioms*)
16 lemma A1: "⌊(□(A→B)) → ((□A)→(□B))⌋" by simp
17 lemma A2: "⌊(□A) → A⌋" by simp
18 lemma A3: "⌊(◇A) → (□(◇A))⌋" by simp
19 lemma A4: "⌊([a1](A → B)) → (([a1]A) → ([a1]B))⌋" by simp
20 lemma A5: "⌊([a1]A) → A⌋" by (simp add: accReR_ag)
21 lemma A6: "⌊(<a1> A) → [a1](<a1>A)⌋" using accSymR_ag accTraR_ag by blast
22 lemma A7: "⌊([Ag](A → B)) → (([Ag]A) → ([Ag]B))⌋" by simp
23 lemma A8: "⌊([Ag] A) → A⌋" by (simp add: accReR_set)
24 lemma A9: "⌊(<Ag> A) → [Ag] (<Ag> A)⌋" using accSymR_set accTraR_set by blast
25 lemma A10: "⌊((◇ ([a1] A)) ∧ (◇ ([a2] B))) → (◇(([a1] A) ∧ ([a2] B)))⌋" nitpick [user_axioms] oops (*ran out of time*)
26 lemma A11: "⌊(([a1] A) ∧ ([a2] B)) → ([Ag](A ∧ B))⌋" nitpick [user_axioms] oops (*ran out of time*)
27 lemma A12: "⌊(⊗ a1 (A → B)) → ((⊗ a1 A) → (⊗ a1 B))⌋" by simp
28 lemma A13: "⌊(□ A) → (([a1] A) ∧ (⊗ a1 A))⌋" by simp
29 lemma A14: "⌊(⊗ a1 A) → (◇ ([a1] A))⌋" using D9 by blast
30 lemma A15: "⌊(◇ (⊗ a1 A)) → (□ (⊗ a1 A))⌋" nitpick [user_axioms] oops (*ran out of time*)
31 lemma A16: "⌊(□ (([a1] A) → ([a1] B))) → ((⊗ a1 A) → (⊗ a1 B))⌋" by (meson D11)
32 lemma A17: "⌊(G (A → B)) → ((G A) → (G B))⌋" by simp
33 lemma A18: "⌊(G A ) → (G (G A))⌋" using RG_trans by blast
34 lemma A19: "⌊(G A ) → (F A)⌋" by (simp add: RG_serial)
35 lemma A20: "⌊(H (A → B)) → ((H A) → (H B))⌋" by simp
36 lemma A21: "⌊A → (G (P A))⌋" by (metis Inv Inv_def)
37 lemma A22: "⌊A → (H (F A))⌋" by (metis Inv Inv_def)
38 lemma A23: "⌊(F (P A)) → (((P A) ∨ A) ∨ (F A))⌋" by (metis T5 Inv Inv_def)
39 lemma A24: "⌊(P (F A)) → (((P A) ∨ A) ∨ (F A))⌋" by (metis T4 Inv Inv_def)
40 lemma A25: "⌊(F (◇ A)) → (<Ag> (F A))⌋"  nitpick [user_axioms] oops (*ran out of time*)
41 lemma R0: "⟦⌊A⌋; ⌊A → B⌋⟧ ⟹ ⌊B⌋" by simp
42 lemma R1a: "⌊A⌋ ⟹ ⌊□ A⌋" by simp
43 lemma R1b: "⌊A⌋ ⟹ ⌊G A⌋" by simp
44 lemma R1c: "⌊A⌋ ⟹ ⌊H A⌋" by simp
45 lemma R2: "⌊((□(¬ A)) ∧ (□((G A) ∧ (H A)))) → B⌋ ⟹ ⌊B⌋" nitpick [user_axioms] oops (*ran out of time*)
46
47 end
```

Figure 5.24: TDS tests

The first block of lemmas (lines 7-14) contains the tautologies of classical propositional calculus that are expected to hold in TDS Logic (van Berkel & Lyon, 2019). All of them can be easily proven via Sledgehammer.

A1-A3, A4-A6, and A7-A9 assure that all of $\Box$, $[i]$, and $[Ag]$ are S5 operators (van Berkel & Lyon, 2019). Sledgehammer confirms this by finding proof for reflexivity, transitivity, and

symmetry for each operator in lines 16-24.

Axioms A12 and A13 combined with R1a, b and c check that $\circ_i$ is a normal modal operator (van Berkel & Lyon, 2019). This is confirmed via Sledgehammer in lines 27-28 and 42-44.

Moreover, several axioms consider the tense operators G and H: Operator G is a KD4 operator and should behave accordingly (van Berkel & Lyon, 2019). This is verified in axioms A17-A19, for all of which Sledgehammer finds proof using transitivity and seriality of the relevant RG relation (lines 32-34). For operator H, axiom K must hold, which is specified by axiom 20. Moreover, axioms A21 and A22 ensure that it is the converse of G (van Berkel & Lyon, 2019). All axioms relating to H are verified via Sledgehammer in lines 35-37.

The remaining axioms establish properties and principles relating to agents and choices. For example, axiom A10 verifies the *independence of agents*, whereas A14 expresses the *ought-implies-can* principle, and axiom A25 states the *no-choice-between-undivided-histories* property. For more details on all axioms, please refer to the literature (van Berkel & Lyon, 2019).

Sledgehammer can prove all but the following five axioms: A10, A11, A15, A25, and R2. Fortunately, Nitpick doesn't find a counterexample that would refute them. Instead, it runs out of time.

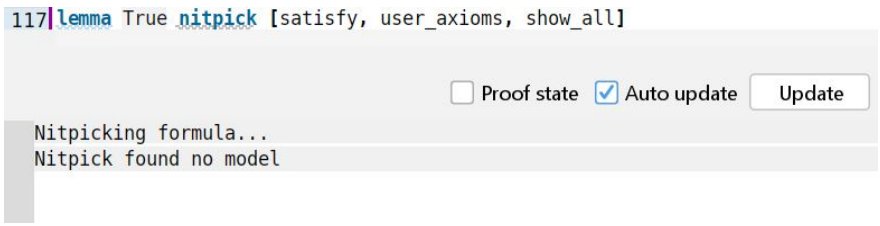


Figure 5.25: No model for TDS Embedding

This likely is connected to another finding made when testing the consistency of the embedding. In line 117 of the TDS embedding, Nitpick is called to provide a model satisfying

all the properties specified within the embedding. However, Nitpick fails to do so, as visible in Figure 5.25.

Several experiments have been run to clarify why, of which one has led to a hypothesis. It appears that constraint T7 makes a difference. The constraint states the following:

$$\forall w\ v.(RBox\ w\ v) \longrightarrow \neg(RG\ w\ v)$$

Spelled out, this means that two worlds that are alternatives to each other can not be in each other's future. Within an experiment, the axiom has been adapted to look like this:

$$\forall w\ v.(RBox\ w\ v) \wedge (w \neq v) \longrightarrow \neg(RG\ w\ v)$$

The only difference here is that this version asserts that w and v are not equal. If everything else is left as it is, Nitpick suddenly finds a model (Figure 5.26). However, with this addition, Nitpick finds counterexamples to the axioms A10, A15, A25, and R2 (Figure 5.27), which is undesirable.

```
117 lemma True nitpick [satisfy, user_axioms, show_all]
```

Proof state ☑ Auto update | Update | Search:

```
Nitpicking formula...
Nitpick found a model for card i = 1:
  Types:
    i × i [boxed] = {(i₁, i₁)}
    ag = {a1, a2}
  Constants:
    Ag = (λx::ag. _)(a1 := True, a2 := True)
    RBox = (λx::i × i. _)((i₁, i₁) := True)
    RG = (λx::i × i. _)((i₁, i₁) := True)
    RH = (λx::i × i. _)((i₁, i₁) := True)
    R_ag = (λx::ag. _)(a1 := (λx::i × i. _)((i₁, i₁) := True), a2 := (λx::i × i. _)((i₁, i₁) := True))
    R_ag_ought = (λx::ag. _)(a1 := (λx::i × i. _)((i₁, i₁) := True), a2 := (λx::i × i. _)((i₁, i₁) := True))
    R_set =
      (λx::ag ⇒ bool. _)
      ((λx::ag. _)(a1 := True, a2 := True) := (λx::i × i. _)((i₁, i₁) := True),
       (λx::ag. _)(a1 := True, a2 := False) := (λx::i × i. _)((i₁, i₁) := True),
       (λx::ag. _)(a1 := False, a2 := True) := (λx::i × i. _)((i₁, i₁) := True),
       (λx::ag. _)(a1 := False, a2 := False) := (λx::i × i. _)((i₁, i₁) := True))
```

Figure 5.26: Nitpick model found

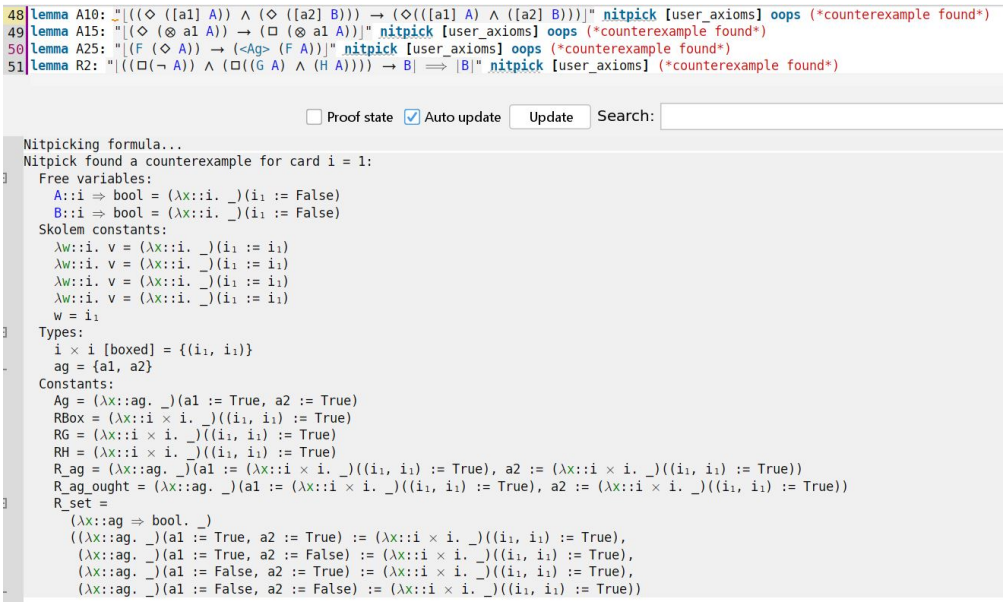


Figure 5.27: Counterexamples to axioms with changed constraint T7

Closer analysis has led to the theory that axiom T7, in combination with the seriality and transitivity of the relation RG, results in infinity. This can be seen in Figure 5.28. In line 62, an abbreviation for the logical concept of infinity is introduced. Then, two lemmas that claim to show infinity are constructed. The first one uses the seriality and transitivity of RG combined with the altered axiom C7; the second one combines them with the original axiom C7. For the lemma, one Nitpick finds a counterexample that refutes the infinity claim. However, Nitpick does not find a counterexample for lemma two. Instead, it runs out of time. These results strongly suggest that TDS Logic has only infinite models, e.g., models with an infinite number of possible worlds. A similar problem has already been found for an Isabelle/HOL embedding of the related T-STIT logic (Lorini, 2013) in the master thesis of Meder, supporting this suspicion (Meder, 2018).

Unfortunately, Nitpick can only reason with finite models (J. Blanchette, 2023). Consequently, the TDS embedding created is not suitable for the purpose of this thesis. An alternative solution is needed.

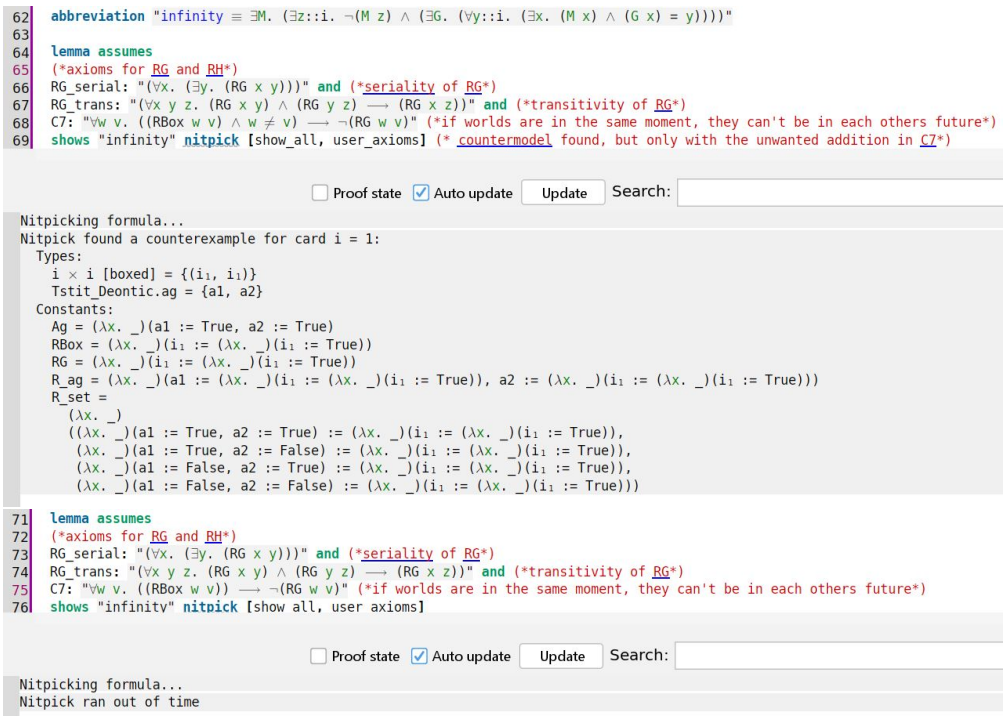


Figure 5.28: Infinity proof

The upcoming section will discuss the possibility of extending DDL, which was identified as being able to represent the modalities discussed in Chapter 5.1 and Chapter 5.2, to integrate a STIT operator and agentive obligations.

5.3.2 Extended DDL

This section will extend the existing DDL embedding by Benzmüller et al. to enable the representation of agency and agentive obligations.

5.3.2.1 Extending DDL with Agentive Obligations and a STIT Operator

To start with, a way to state agentive obligations must be included. In DDL, the conditional and normal obligation operators, as well as the □ a and □ p , use the relations *av* (actual worlds), *pv* (potential worlds), and *ob* (set of propositions that are obligatory in a context). To formulate obligations for a specific agent, additional accessibility relations must be introduced. We declare agents as constants of type *ag* in the types file (5.29), with each agent constant representing a particular type of agent. The new accessibility relations can be associated with specific agent types. For example, the new relations *avd*, *pvd*, and *obd* denote available worlds and potential worlds for agent *d*, and the propositions that are obligatory in a context for that agent *d*. They are displayed in Figure 5.30.

```
64 consts
65     (*identify agents:*)
66     a::ag (*a = type for judicial authorities or independent administrative authorities*)
67     b::ag (*b = type for importers*)
68     c::ag (*c = type for eu commission*)
69     d::ag (*d = type for providers*)
70     e::ag (*e = type for conformity assessment bodies*)
71     f::ag (*f = type for notifying authorities*)
72     g::ag (*g = type for notified bodies*)
73     h::ag (*h = type for members states*)
```

Figure 5.29: Agent constants

```
consts
cw::i (*current world*)
av::"i⇒σ" pv::"i⇒σ" ob::"σ⇒(σ⇒bool)" (*general accessibility relations*)

avd::"i⇒σ" pvd::"i⇒σ" obd::"σ⇒(σ⇒bool)" (*accessibility relations for agent d*)
```

Figure 5.30: Accessibility relations agent d

Next, what holds for the general accessibility relations must also hold for the agent-dependent accessibility relations. Therefore, the axioms stated for the general accessibility relations are added for the agent-dependent ones (Figure 5.31).

```
axiomatization where
  ax_3a: "∀w.∃x. av(w)(x)" and ax_4a: "∀w x. av(w)(x) ⟶ pv(w)(x)" and ax_4b: "∀w. pv(w)(w)" and
  ax_5a: "∀X.¬ob(X)(λx. False)" and
  ax_5b: "∀X Y Z. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (ob(X)(Y) ⟷ ob(X)(Z))" and
  ax_5ca: "∀X β. ((∀Z. β(Z) ⟶ ob(X)(Z)) ∧ (∃Z. β(Z))) ⟶
      (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ ob(X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
  ax_5c: "∀X Y Z. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  ob(X)(Y)  ∧  ob(X)(Z))  ⟶  ob(X)(λw. Y(w) ∧ Z(w)))" and
  ax_5d: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ ob(X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                    ⟶ ob(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
  ax_5e: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ ob(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ ob(Y)(Z)" and

  (*for agent d: providers*)
  axd_3a: "∀w.∃x. avd(w)(x)" and axd_4a: "∀w x. avd(w)(x) ⟶ pvd(w)(x)" and axa_4ba: "∀w. pvd(w)(w)" and
  axd_5a: "∀X.¬obd(X)(λx. False)" and
  axd_5b: "∀X Y Z. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (obd(X)(Y) ⟷ obd(X)(Z))" and
  axd_5ca: "∀X β. ((∀Z. β(Z) ⟶ obd(X)(Z)) ∧ (∃Z. β(Z))) ⟶
      (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ obd(X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
  axd_5c: "∀X Y Z. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  obd(X)(Y)  ∧  obd(X)(Z))  ⟶  obd(X)(λw. Y(w) ∧ Z(w)))" and
  axd_5d: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obd(X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                    ⟶ obd(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
  axd_5e: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obd(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ obd(Y)(Z)" and
```

Figure 5.31: Axioms for new accessibility relations

Finally, the necessity, possibility, and obligation operators must be adapted to allow the expression of both general and agent-dependent obligations, necessity, and possibility (Figures 5.32, 5.33). Therefore, the existing operators for actual and potential necessity, possibility, and obligation are replaced with more general ones that take as input not only a proposition but also a relation. That way, different relations can be passed in to the operator (agent-dependent or general ones), enabling expressions such as: *it is actually obligatory for agent a that...*, etc., in addition to *it is actually obligatory that*, etc. If one wants to formulate an agentive statement, the respective relation belonging to that agent (Figure 5.30, line 11) must be provided. If general statements are desired, the usual, non-agentive relation (Figure

5.30, line 9) must be passed in.

```
15 type_synonym ν = "(σ⇒(σ⇒bool))⇒σ⇒σ⇒σ"
16 type_synonym μ = "(σ⇒(σ⇒bool))⇒(i⇒i⇒bool)⇒σ⇒σ"
17 type_synonym ζ = "(i⇒σ)⇒σ⇒σ"
```

Figure 5.32: Types new operators

```
123 (*Necessity/possibility for agents*)
124 abbreviation ddlboxa_g::ζ ("□ₐ") where "□ₐ rel A ≡ λw. (∀x. (rel (w)(x) ⟶ A(x)))"  (*in all actual worlds*)
125 abbreviation ddlboxp_g::ζ ("□ₚ") where "□ₚ rel A ≡ λw. (∀x. (rel (w)(x) ⟶ A(x)))" (*in all potential worlds*)
126 abbreviation ddldia_g::ζ ("◇ₐ") where "◇ₐ rel A ≡ ¬□ₐ rel (¬A)"
127 abbreviation ddldiap_g::ζ ("◇ₚ") where "◇ₚ rel A ≡ ¬□ₚ rel (¬A)"
128
129 (*generalised obligation operators with relation as a parameter*)
130 abbreviation ddlo_g::ν ("O _ (_|_)") where "O rel ⟨B|A⟩ ≡ λw. rel (A)(B)"  (*it ought to be A, given B *)
131 abbreviation ddloa_g::μ  ("Oₐ ") where "Oₐ rel1 rel2 A ≡ λw. rel1(rel2(w))(A) ∧ (∃x. rel2(w)(x) ∧ ¬A(x))" (*actual obligation*)
132 abbreviation ddlop_g::μ  ("Oₚ") where "Oₚ rel1 rel2 A ≡ λw. rel1(rel2(w))(A) ∧ (∃x. rel2(w)(x) ∧ ¬A(x))"  (*primary obligation*)
```

Figure 5.33: New operators

An additional abbreviation is introduced to enable the easier expression of agentive obligations (Figure 5.34). While line 151 introduces the usual DDL obligation operator using the conditional obligation operator with the general *ob* relation, line 152 defines an obligation operator for agent *d* using the conditional obligation operator with the *obd* relation belonging to agent *d*. Now, the formula Od <P> can be used to express that it is *obligated for agent d that P*.

```
73 abbreviation ddlobl::γ ("○<_>") where "○<A> ≡  O ob ⟨A|T⟩"  (*New syntax: A is obligatory.*)
74 abbreviation ddlobld::γ ("○d<_>") where "○d<A> ≡  O obd ⟨A|T⟩"  (*New syntax: A is obligatory for agent d.*)
```

Figure 5.34: Agentive Obligation operator

Even though only one agent type has been introduced so far, further agent types can easily be added. For each agent type, a set of accessibility relations, axioms constraining them, and an agentive obligation operator must be introduced. With more than one agent type, it is also necessary to specify that the agents are distinct from each other. To achieve this, an axiom of the form $agent_1 \neq agent_2$ (and more axioms like this if more than two agent types are needed) must be added to the axiomatization.

In addition to agentive obligations, DDL will be extended with a STIT operator (Figure

5.35). It can be used to represent statements of the form d *stit* P which express that *agent type d sees to it that P.*

While this operator can now be used within a statement, it currently lacks semantic meaning. In TDS logic, the STIT operator is embedded in a rich semantic framework. That level of expressiveness will not be achieved here. Instead, the goal will be to equip the STIT operator with only the minimal semantics necessary in the context of the AIA.

```
13 (*stit operator*)
14 stit::"ag⇒σ⇒σ" (*ag sees to it that*)
```

Figure 5.35: STIT operator DDL

This is achieved by adding an axiom to the DDL embedding (Figure 5.36). With one simple line, it is ensured that whatever an agent a sees to actually holds.

```
stit1: "∀a F w. ((stit a F) w) ⟶ F w"
```

Figure 5.36: Axiomatization STIT operator

```
80 consts
81   l::aiSys
82   compliance_req_chap2::"aiSys⇒σ"
83
84 lemma assumes
85 (*"∀ a F w. ((stit a F) w) ⟶ F w"*)
86 "⌊stit d (compliance_req_chap2 l)⌋"
87 shows "⌊compliance_req_chap2 l⌋" try

Proof state  Auto update  Update  Search
Trying "solve_direct", "quickcheck", "try0", "sledgehammer", and "nitpick"...
Nitpick found a quasi genuine counterexample for card ag = 1, card aiSys = 1, and card i = 1:
```

Figure 5.37: Without added axiom example 1

Let's look at an example: Article 16 lists several obligations for providers of high-risk AI systems, for instance, to "ensure that their high-risk AI systems are compliant with the requirements set out in Chapter 2 of this Title" (*Artificial Intelligence Act*, 2021, p. 52).

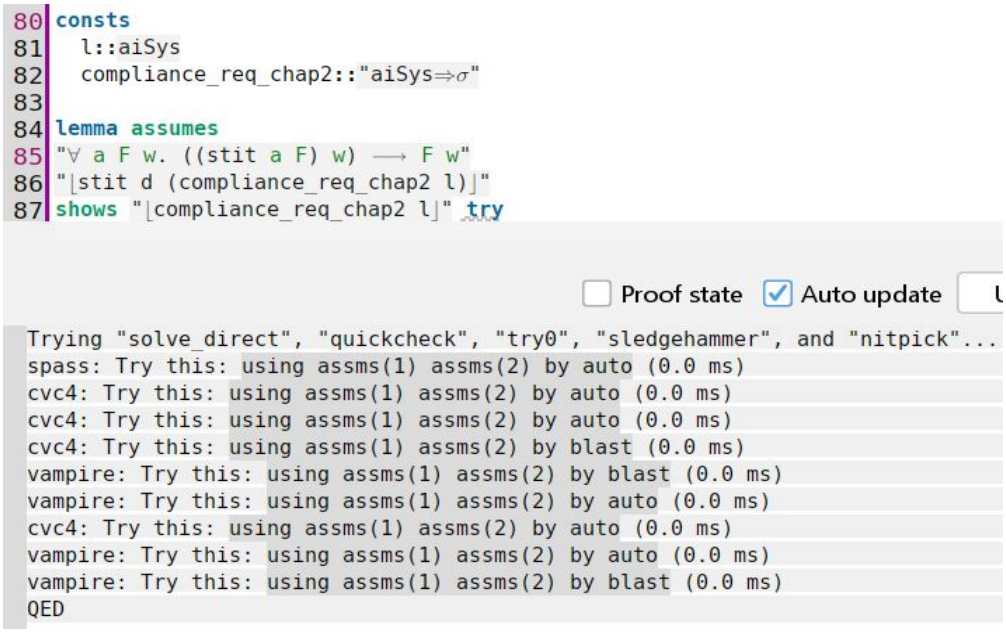


Figure 5.38: With added axiom example 1

Formalized, this sentence looks like this: *Od <stit d compliance_req_chap2 l>*, with *d* representing the provider, *l* representing a high-risk AI system, and *compliance_req_chap2 l* expressing that a quality management system exists for that system *l*. To then state that the provider fulfills their obligation, one can write: *stit d compliance_req_chap2 l*. Logically, from this statement, it must be possible to infer that *compliance_req_chap2 l* evaluates to true. However, without the added axioms, this inference does not hold (Figure 5.37). Only with the addition the statement can be proven (Figure 5.38).

In some places of the AIA, obligations for a first agent to see to it that another second agent acts in a certain way are specified. From this, it should be possible to infer that the second agent acts in the way the first agent desires. For example, let's look at this sentence from Article 26 (disregarding temporality): "(...) importers of such [high-risk] system shall ensure that: (a) the appropriate conformity assessment procedure [is] carried out by the provider of that AI system (...)" (*Artificial Intelligence Act*, 2021, p. 56). If the importer fulfills this obligation in a concrete situation, it should logically follow that the provider sees

to it that the appropriate conformity assessment procedure is carried out.

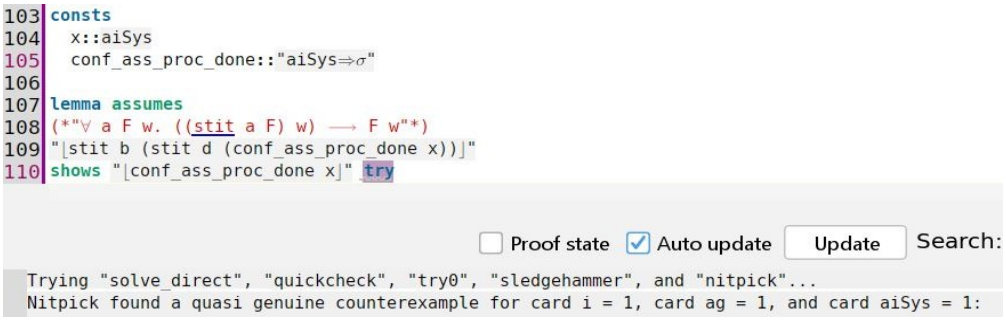


Figure 5.39: Without added axiom example 2

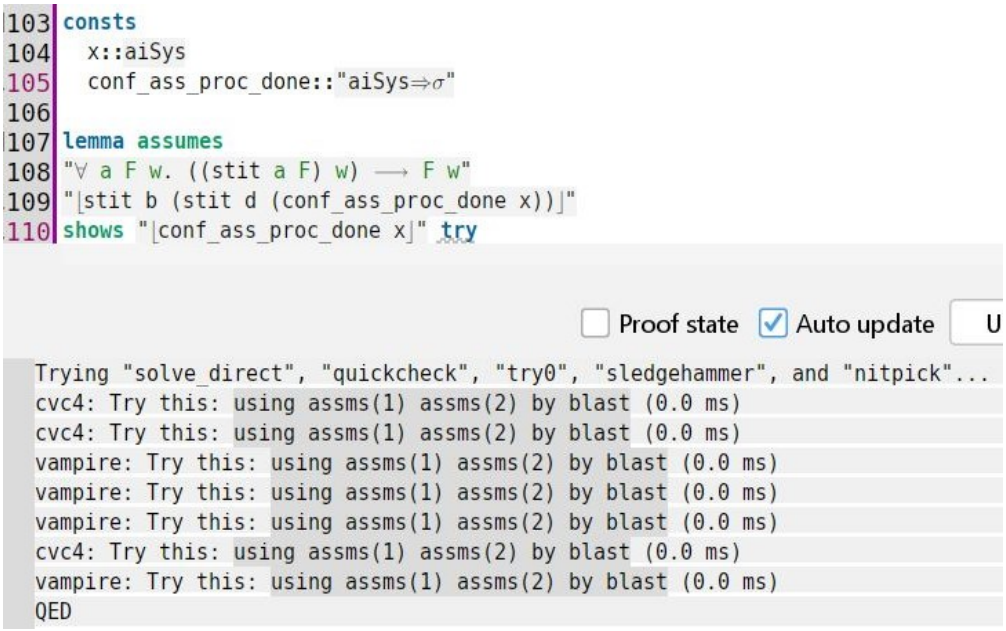


Figure 5.40: With added axiom example 2

As visible in Figures 5.39 and 5.40, the added axiom for the STIT operator ensures this. Here, agent b denotes the importer of a high-risk AI system, for which accessibility relations, affiliated axioms, an obligation operator, and an axiom to ensure that it is distinct from agent d have been added to the DDL embedding. If it is now stated that the importer fulfills their

obligation, the desired inference can be proven via Sledgehammer using this statement and axiom stit1.

Nitpick is used to confirm consistency, and a model is found for Extended DDL with two agents b and d (Figure 5.41). Unfortunately, this is guaranteed only up to a cardinality of i=2. For higher cardinalities (i > 2), Nitpick runs out of time, suggesting that the models soon become too complex.

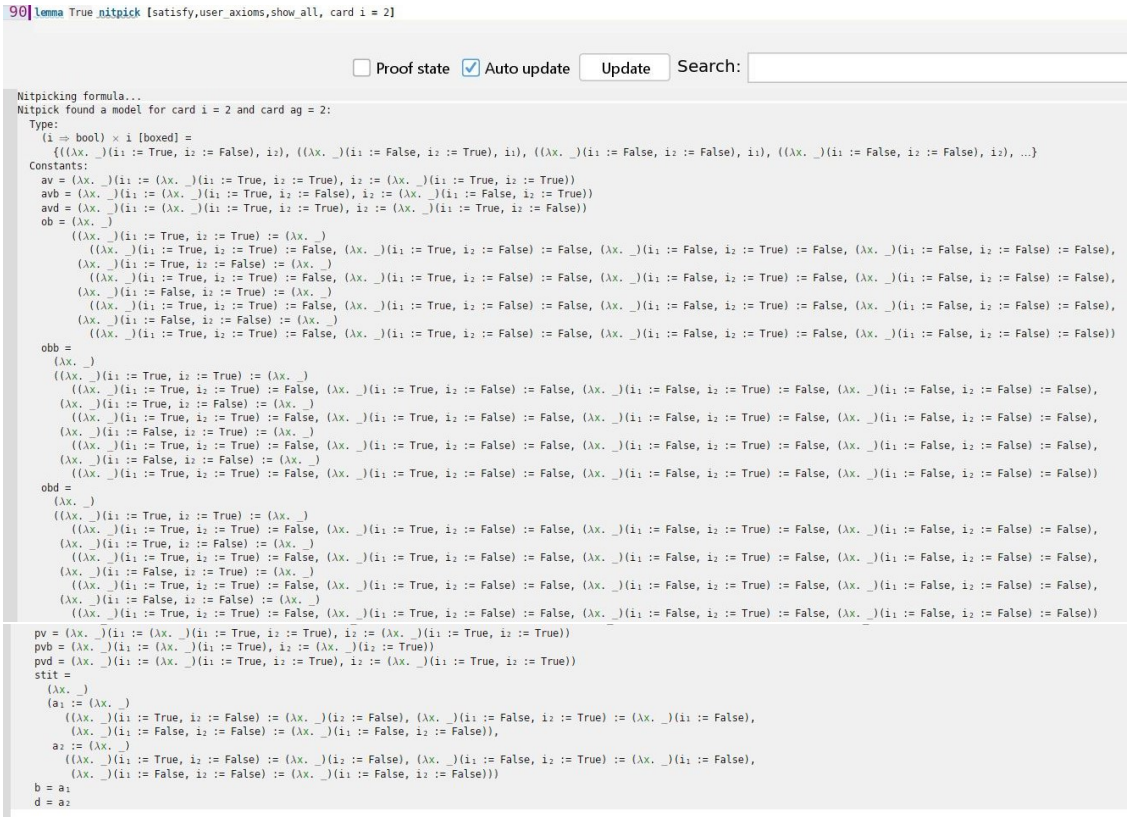


Figure 5.41: Nitpick model Extended DDL

To further check the correctness of Extended DDL, the tests that Benzmüller et al. have run on their DDL Embedding were run for Extended DDL. Expect for lemma C_8 and lemma obs_II_4_2_2 which is based on it, all lemmas that hold for O, Oa, Op, $\Diamond$, and $\Box$ in DDL, can also be proven for the agentive version for these operators in Extended DDL. It is not

possible here to analyze why the two lemmas do not hold, so this is a question for future work. The files containing the DDL embedding with two agents d and b, the tests for DDL provided by Benzmüller et al., as well as the adapted version testing the same properties for Extended DDL can be found in Appendix Two, Appendix Five, and Appendix Six, respectively.

The minimal semantics achieved by adding axiom stit1 is nowhere near as expressive as TDS semantics. However, this is not necessarily required to represent the AIA. The AIA's main purpose is to specify obligations for different agents in a broader context, which already seems possible with Extended DDL. The upcoming section will verify this claim by formalizing selected parts from the AIA in Extended DDL and running some tests. Whenever additional agents are needed, they will be added to the existing embedding, as explained earlier.

5.3.2.1.1 Using Extended DDL to Represent Parts from the AIA

In this Section, Extended DDL is used to re-formalize the CTD from Chapter 5.2. Having disregarded agency before, it will now be taken into account to represent the correct CTD in Extended DDL and test whether a faithful representation can be achieved. For the sake of completeness, a representation of the example discussed in Chapter 5.1 in Extended DDL can be found in Appendix 9.

The following CTD has already been formalized in (the usual version of) DDL in Chapter 5.2:

> "Providers of high-risk AI systems shall:
> (a) ensure that their high-risk AI systems are compliant with the requirements set out in Chapter 2 of this Tile;
> (...)
> (g) take the necessary corrective actions if the high-risk AI system is not in conformity with the requirements set out in Chapter 2 of this Tile;
> (h) inform the national competent authorities of the Member States in which they made the AI system available or put it into service and, where applicable,

the notified body of the non-compliance and of any corrective actions taken; (...)" (*Artificial Intelligence Act*, 2021, 52f.).

While the agency of the provider was ignored in Chapter 5.2, it will now be acknowledged and modeled using Extended DDL. The general structure of the CTD does not change from Chapter 5.2. Again, the focus lies on the obligation to inform authorities, whereas the second obligation to take corrective action is ignored. However, this time, the fact that the obligations are agentive for the provider will be modeled, too. It is assumed that the agent term d denotes providers. The result is displayed in Figure 5.42.

```
55 (*-----------------------------------------------------------------*)
56 (*CTD example Extended DDL:*)
57 consts
58   l::aiSys
59
60
61 (*interesting part: CTD; Trying to create the typical structure*)
62 axiomatization where
63 F1: "⌊(high_risk l)⌋" and
64 A1: "⌊∀x::aiSys.(high_risk x) ⟶ ⊙d<stit d (compliance_req_chap2 x)>⌋" and
65 A8: "⌊∀x::aiSys. ¬(compliance_req_chap2 x) ∧ (high_risk x) ⟶ ⊙d<(stit d (inform_com_auth x))>⌋" and
66 (*implicit: If the compliance with the requirements is a given, the provider is obligated to not inform authorities
67 of non-compliance (since that would make no sense*)
68 AX: "⌊∀x::aiSys. ⊙d<(compliance_req_chap2 x ∧ high_risk x) ⟶ ¬ stit d (inform_com_auth x)>⌋"and
69 Situation: "⌊¬ (compliance_req_chap2 l)⌋"
```

Figure 5.42: Modeling CTD with agentive obligations in Extended DDL

Line 58 introduces a constant l for an AI system, whereas line 59 defines a constant for specifying the relation between providers and AI systems. In the axiomatization, it is first claimed that system l is a high-risk system (line 63). Next, line 64 states that for all high-risk AI systems, the provider of the system is obligated to see to it that it complies with the requirements specified in Tile 2 of the AIA. Afterward, it is claimed that if an AI system is high-risk and does not comply with the requirements, the provider is obligated to see to it that the national competent authorities of the concerned Member States are informed about the system (line 65). Line 68 then specifies the implicit obligation: If the system is high-risk and does comply with the requirements, the provider is not obligated see to it that the authorities are informed. Finally, a situation where the high-risk AI system l does not fulfill the requirements in the current world is expressed in line 69.

The lemma in line 72 confirms consistency since Nitpick finds a model (figure 5.34.).

The two other lemmas also yield the expected results (Figure 5.44): Whereas Sledgehammer provides proof for the statement that, in the current world, the provider is obligated to see to it that the authorities are informed (line 74), a counterexample is found for the opposite claim (line 75). Consequently, the CTD and its agentive obligations can be successfully formalized in Extended DDL.

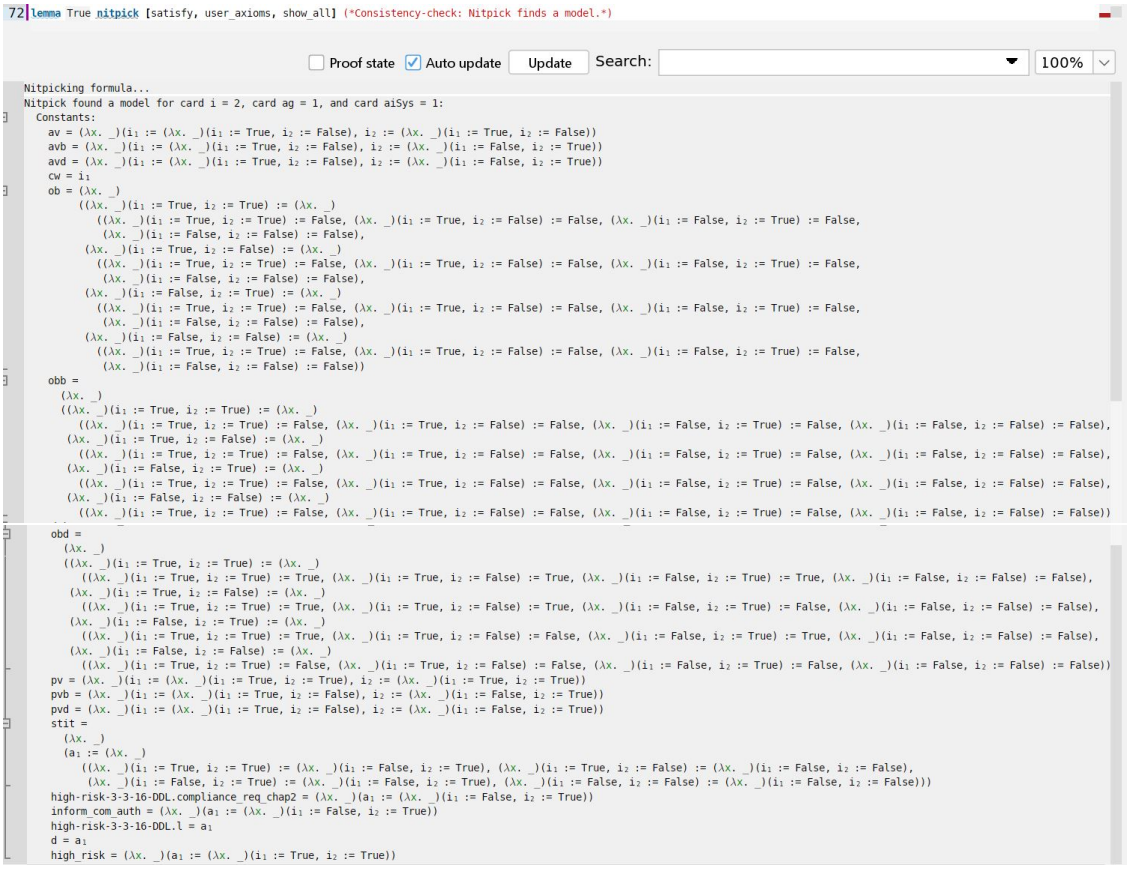


Figure 5.43: Nitpick model for Extended DDL CTD

However, this representation is not without problems. A major flaw of Extended DDL is the restricted options in modeling types of agents and their relations to one another. Since Extended DDL uses accessibility relations that are tied to one generic agent type declared as a constant, no further constraints can be imposed on this agent type. If it is stated -

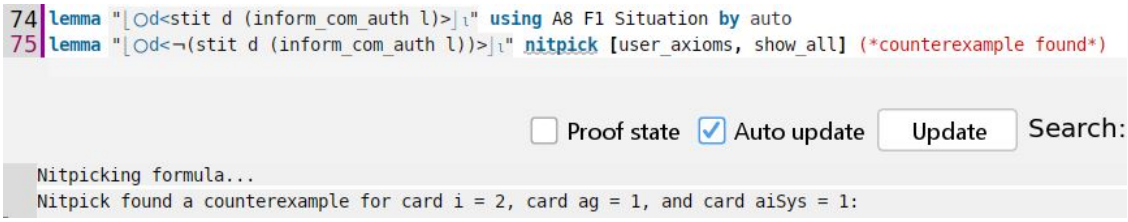


Figure 5.44: Successful tests for Extended DDL CTD

as in the example above - that the agent constant d represents providers, the obligations stated for providers using the *obd* relation (Figure 5.30) will always hold for all providers. Even if Extended DDL included predicates for defining such subgroups and relations and quantifying agents (such as *provider_of* d x denoting that d is the provider of x), it would be impossible to define obligations that affect only the providers of system x. This is not optimal. Ideally, it should be possible to further restrict agent types to subgroups fulfilling certain criteria or standing in a specific relation, e.g., to all providers of high-risk AI systems, to all providers within a certain member state, etc.

Closely related is another particularity of agents within the AIA: Articles 31, 32, and 33 talk about conformity assessment bodies and notified bodies (*Artificial Intelligence Act*, 2021). During the text, it becomes clear that a conformity assessment body can become a notified body if it fulfills certain conditions. A similar notion appears in Article 28: "Any distributor, importer, user or other third-party shall be considered a provider for the purposes of this Regulation and shall be subject to the obligations of the provider under Article 16, in any of the following circumstances (...)" (*Artificial Intelligence Act*, 2021, p. 57), as well as in Articles 17, 18, 19, 20, in which the following sentence reappears: "Providers that are credit institutions regulated by Directive 2013/36/EU (...)" (*Artificial Intelligence Act*, 2021, p. 54). For the representation of the AIA, this means that types of agents are not necessarily distinct. One agent can be a conformity assessment body **and** a notified body, an importer **and** a provider, a provider **and** a credit institution, etc. Consequently, Extended DDL will lead to problems since its agents are distinct, and the corresponding accessibility

relations on which the agentive obligations are based always belong to one generic agent, which represents an agent type.

Based on these insights, it seems necessary to modify Extended DDL and the way agents are represented. This will be attempted in the following Section.

5.3.2.2 Extended DDL 2: A Modification for a Flexible Representation of Agents

This Section introduced a modified version of Extended DDL, which will be referred to as *Extended DDL 2* from now on. All differences between Extended DDL and Extended DDL will be explained step by step. The full file containing Extended DDL 2 can be found in Appendix Four.

Let's begin with the representation of agents: Instead of the different constants (Figure 5.29) that were used in Extended DDL, Extended DDL 2 introduces predicates indicating the type of agents (Figure 5.45). The only constant that is introduced is *eu_comm* in line 71. This constant denotes the EU commission, and since there is only one EU commission, this kind of representation makes sense. Using predicates to identify the other agents makes it possible to differentiate types of agents while still having the possibility to impose further restrictions on a selection of agents, characterize subgroups, and quantify them. A second type file, *types_2*, is created, which contains the same content as *types* but replaces the constants from Figure 5.29 with the predicates and the EU commission constant in Figure 5.45. This types_2 file is imported into Extended DDL 2.

Next, the modeling of the *av, pv,* and *ob* relations must be changed. Instead of one set of relations per agent (Figure 5.30), the relations *av_g, pv_g*, and *ob_g* are used, which take an agent as a parameter, as visible in Figure 5.46. Consequently, it is easy to employ predicates to restrict this agent and specify its type and relations. Based on this new representation of the accessibility relations, the corresponding axioms must be adapted as well (Figure 5.47).

```
consts
    (*identify agents:*)
    eu_comm::ag
    provider::"ag⇒σ"
    importer::"ag⇒σ"
    notif_authority::"ag⇒σ"
    member_state::"ag⇒σ"
    conf_ass_body::"ag⇒σ"
```

Figure 5.45: Indicating agent types via predicates

```
consts
 cw::i (*current world*)
 av::"i⇒σ" pv::"i⇒σ" ob::"σ⇒(σ⇒bool)" (*general accessibility relations*)

 av_g::"ag⇒i⇒σ" pv_g::"ag⇒i⇒σ" ob_g::"ag⇒(σ⇒(σ⇒bool))" (*agent-dependent accessibility relations*)
```

Figure 5.46: Relations with agent term as parameter

```
(*agent-dependent axioms*)
axg_3a: "∀w a.∃x. av_g a (w)(x)" and axg_4a: "∀w x a. av_g a (w)(x) ⟶ pv_g a (w)(x)" and axg_4ba: "∀w a. pv_g a (w)(w)" and
axg_5a: "∀X a.¬ob_g a (X)(λx. False)" and
axg_5b: "∀X Y Z a. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (ob_g a(X)(Y) ⟷ ob_g a (X)(Z))" and
axg_5ca: "∀X β a. ((∀Z. β(Z) ⟶ ob_g a(X)(Z)) ∧ (∃Z. β(Z))) ⟶
     (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ ob_g a (X)(λw. ∀Z. (β Z) ⟶ (Z w)))" and
axg_5c: "∀X Y Z a. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  ob_g a (X)(Y)  ∧  ob_g a(X)(Z))  ⟶  ob_g a(X)(λw. Y(w) ∧ Z(w)))" and
axg_5d: "∀X Y Z a. ((∀w. Y(w) ⟶ X(w)) ∧ ob_g a (X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                  ⟶ ob_g a(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
axg_5e: "∀X Y Z a. ((∀w. Y(w) ⟶ X(w)) ∧ ob_g a(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ ob_g a(Y)(Z)" and
```

Figure 5.47: Adapted axioms

The agentive obligation, necessity, and possibility operators must also change. Instead of taking in a relation as a parameter, they now accept only an agent term that is then passed into the related agent-dependent relation, either *av_g, pv_g*, or *ob_g* (Figure 5.48). To maintain the option of expressing possibility, necessity, and obligation without association with a specific agent, the non-agentive operators from the original DDL embedding (Benzmüller et al., 2022) must be included as well (Figure 5.49).

Furthermore, the abbreviations for the agentive obligation operators are adapted. Instead of one abbreviation per agent, there now is only a single abbreviation taking an agent term as a parameter (Figure 5.50).

The STIT operator and its axiomatization must not be changed since this operator

```
48 (*Necessity/possibility for agents*)
49 abbreviation ddlboxa_g::η ("□ₐ") where "□ₐ i A ≡ λw. (∀x. av_g i (w)(x) ⟶ A(x))"  (*in all actual worlds*)
50 abbreviation ddlboxp_g::η ("□ₚ") where "□ₚ i A ≡ λw. (∀x. pv_g i (w)(x) ⟶ A(x))" (*in all potential worlds*)
51 abbreviation ddldiaa_g::η ("◇ₐ") where "◇ₐ i A ≡ ¬□ₐ i (¬A)"
52 abbreviation ddldiap_g::η ("◇ₚ") where "◇ₚ i A ≡ ¬□ₚ i (¬A)"
53 (*generalised operators with agents as a parameter*)
54 abbreviation ddlo_g::χ ("O _ ⟨_|_⟩") where "O i ⟨B|A⟩ ≡ λw. ob_g i A B"  (*Agent i ought to A, given B *)
55 abbreviation ddloa_g::η  ("Oₐ _ ") where "Oₐ i A ≡ λw. ob_g i (av_g i (w))(A) ∧ (∃x. av_g i (w)(x) ∧ ¬A(x))" (*actual obligation*)
56 abbreviation ddlop_g::η  ("Oₚ _") where "Oₚ i A ≡ λw. ob_g i (pv_g i (w))(A) ∧ (∃x. pv_g i (w)(x) ∧ ¬A(x))"  (*primary obligation*)
```

Figure 5.48: Adapted agentive Necessity, Possibility, and Obligation operators

```
58 (*non-agentive necessity, possibility and obligation operators*)
59 abbreviation ddlboxa::γ ("□ₐ") where "□ₐA ≡ λw. (∀x. av(w)(x) ⟶ A(x))"  (*in all actual worlds*)
60 abbreviation ddlboxp::γ ("□ₚ") where "□ₚA ≡ λw. (∀x. pv(w)(x) ⟶ A(x))" (*in all potential worlds*)
61 abbreviation ddldiaa::γ ("◇ₐ") where "◇ₐA ≡ ¬□ₐ(¬A)"
62 abbreviation ddldiap::γ ("◇ₚ") where "◇ₚA ≡ ¬□ₚ(¬A)"
63 abbreviation ddlo::ϱ ("O⟨_|_⟩"[52]53) where "O⟨B|A⟩ ≡ λw. ob(A)(B)"  (*it ought to be ψ, given φ *)
64 abbreviation ddloa::γ  ("Oₐ") where "OₐA ≡ λw. ob(av(w))(A) ∧ (∃x. av(w)(x) ∧ ¬A(x))" (*actual obligation*)
65 abbreviation ddlop::γ  ("Oₚ") where "OₚA ≡ λw. ob(pv(w))(A) ∧ (∃x. pv(w)(x) ∧ ¬A(x))"  (*primary obligation*)
```

Figure 5.49: Non-agentive Necessity, Possibility, and Obligation operators

already accepts an agent term as an input.

As in Extended DDL, the obligation for an agent a that P can be expressed in the following way in Extended DDL 2: Oa <P>. However, it is now possible to quantify agents and express their relations to one another. For example, predicates could be used to state that agent a is an importer of a high-risk system x and located in Austria. Obligations specified for agent a in Extended DDL 2 would then not hold for all importers, like in Extended DDL, but only for those satisfying the given predicates.

Before Extended DDL 2 is employed to represent parts from the AIA, its consistency must be verified. Fortunately, all lemmas (originally formulated by Benzmüller et al. for DDL) that were proven for Extended DDL (Benzmüller et al., 2022) can also be proven for Extended DDL 2. The complete test file can be found in Appendix Seven.

However, Extended DDL 2 brings different problems, as visible in the consistency check in line 84. While it does find a nitpick model for a cardinality of i (denoting the possible worlds) of one, it fails to do so for a cardinality of two or higher (Figures 5.51, 5.52).

```
79 (*New syntax *)
80 abbreviation ddlobl::γ ("O<_>") where "O<A> ≡  O⟨A|⊤⟩"
81 abbreviation ddlobl_g::η ("O_<_>") where "O i <A> ≡  O i ⟨A|⊤⟩"
```

Figure 5.50: Abbreviation for agentive Obligation operator

```
83 (* Consistency *)
84 lemma True nitpick [satisfy,user_axioms,show_all,card i=1] (*no model for i>1*)

                                  Proof state  Auto update  Update  Search:

Nitpicking formula...
Nitpick found a model for card i = 1 and card ag = 1:
  Types:
    (i ⇒ bool) × (i ⇒ bool) [boxed] =
      {((λx. _)(i₁ := True), (λx. _)(i₁ := True)), ((λx. _)(i₁ := True), (λx. _)(i₁ := False)), ((λx. _)(i₁ := False), (λx. _)(i₁ := True)),
       ((λx. _)(i₁ := False), (λx. _)(i₁ := False))}
    (i ⇒ bool) × i [boxed] = {((λx. _)(i₁ := True), i₁), ((λx. _)(i₁ := False), i₁)}
    i × i [boxed] = {(i₁, i₁)}
  Constants:
    av = (λx. _)(i₁ := (λx. _)(i₁ := True))
    av_g = (λx. _)(a₁ := (λx. _)(i₁ := (λx. _)(i₁ := True)))
    ob = (λx. _)
         ((λx. _)(i₁ := True) := (λx. _)((λx. _)(i₁ := True) := False, (λx. _)(i₁ := False) := False),
          (λx. _)(i₁ := False) := (λx. _)((λx. _)(i₁ := True) := False, (λx. _)(i₁ := False) := False))
    ob_g =
      (λx. _)
      (a₁ := (λx. _)
         ((λx. _)(i₁ := True) := (λx. _)((λx. _)(i₁ := True) := False, (λx. _)(i₁ := False) := False),
          (λx. _)(i₁ := False) := (λx. _)((λx. _)(i₁ := True) := False, (λx. _)(i₁ := False) := False)))
    pv = (λx. _)(i₁ := (λx. _)(i₁ := True))
    pv_g = (λx. _)(a₁ := (λx. _)(i₁ := (λx. _)(i₁ := True)))
    stit = (λx. _)(a₁ := (λx. _)((λx. _)(i₁ := True) := (λx. _)(i₁ := False), (λx. _)(i₁ := False) := (λx. _)(i₁ := False)))
```

Figure 5.51: Nitpick model found for cardinality of i=1

```
83 (* Consistency *)
84 lemma True nitpick [satisfy,user_axioms,show_all,card i=2] (*no model for i>1*)

                                  Proof state  Auto update  Update

Nitpicking formula...
Nitpick found no model
```

Figure 5.52: No Nitpick model found for cardinality of i=2

This can happen for a variety of reasons, such as the complexity of the model or limitations of Nitpick. Some tests were executed to find out whether a model can be found when removing some of the constraints on the *av_g*, *pv_g*, and *ob_g* relations. Indeed, this is the case: When omitting the axioms commented out in Figure 5.53, Nitpick manages to find a model for higher cardinalities (Figure 5.54).

```
(*agent-dependent axioms*)
axg_3a: "∀w a.∃x. av_g a (w)(x)" and axg_4a: "∀w x a. av_g a (w)(x) ⟶ pv_g a (w)(x)" and axg_4ba: "∀w a. pv_g a (w)(w)" and
axg_5a: "∀X a.¬ob_g a (X)(λx. False)" and
(*axg_5b: "∀X Y Z a. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (ob_g a(X)(Y) ⟷ ob_g a (X)(Z))" and
axg_5ca: "∀X β a. ((∀Z. β(Z) ⟶ ob_g a(X)(Z)) ∧ (∃Z. β(Z))) ⟶
    (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ ob_g a (X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
axg_5c: "∀X Y Z a. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  ob_g a (X)(Y)  ∧  ob_g a(X)(Z))  ⟶  ob_g a(X)(λw. Y(w) ∧ Z(w)))" and*)
axg_5d: "∀X Y Z a. ((∀w. Y(w) ⟶ X(w)) ∧ ob_g a (X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                  ⟶ ob_g a(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
(*axg_5e: "∀X Y Z a. ((∀w. Y(w) ⟶ X(w)) ∧ ob_g a(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ ob_g a(Y)(Z)" and*)
```

Figure 5.53: Commenting out certain axioms

Figure 5.54: Nitpick model found for cardinality of 2

Analysing why these axioms cause problems for higher cardinalities of i is not within the scope of this thesis but should be a topic of future work. For now, it must be acknowledged that while Extended DDL 2 offers more flexibility to represent agents, subgroups of agents, and relations between agents, Nitpick only finds a model confirming its consistency up to a cardinality of i=1.

In the following Section, a representation of an example from Article 31 and the CTD already formalized in DDL (Chapter 5.2.2) and Extended DDL (Chapter 5.3.2.1.1) will be created in Extended DDL 2. Thereby, it shall be tested how well Sledgehammer can reason with the result.

5.3.2.2.1 Using Extended DDL 2 to Represent Parts from the AIA

This Section discusses a representation of the following excerpt from Article 31: "Confor-

mity assessment bodies shall submit an application for notification to the notifying authority of the Member State in which they are established" (*Artificial Intelligence Act*, 2021, p. 59). The complete representation can be found in Figure 5.55. Explanations are given below.

```
1  theory "extendedDDL-mult-agents"
2    imports
3      DDL_agents_mod
4  begin
5
6  (*Article 31*)
7  consts
8    e::ag
9    notif_auth_of::"ag⇒ag" (*gives back notifying authority of a member state*)
10   established_in::"ag⇒ag" (*member state in which conf ass body is established*)
11   submit_appl_for_notific::"ag⇒ag⇒σ" (*conf_ass_body submits application to a
12   notifying authority*)
13
14 axiomatization where
15   F1: "⌊conf_ass_body e⌋ι" and
16   R1: "⌊∀a.(conf_ass_body a) → Oa<stit a (submit_appl_for_notific a (notif_auth_of (established_in a)))>⌋"
17
18 lemma test1: "⌊Oe<stit e (submit_appl_for_notific e (notif_auth_of (established_in e)))>⌋ι" using F1 R1 by auto
19
20 lemma True nitpick [satisfy, user_axioms, show_all] (*Consistency-check: Nitpick finds a model.*)
```

Figure 5.55: Example Article 31 in Extended DDL 2

To start with, the file DDL_agents_mod is imported. It contains Extended DDL 2, as explained above. In line 8, *e* is declared to be an agent. Next, three predicates needed to express the example sentence are defined: *notif_auth_of* takes a notifying authority and gives back the member state it belongs to; *submit_appl_for_notific* checks whether a conformity assessment applies for notification with a notifying authority, and *established_in* takes a conformity assessment body and gives back the member state it is established in (lines 9-12). Afterward, the sentence in question is formalized. Axiom *F1* states the fact that in the current world, agent *e* is a conformity assessment body (line 15). Axiom *R1* then represents the example using the previously defined constants: If an agent *a* is a conformity assessment body, it is obligated to see to it that an application for notification is submitted to the notifying authority of the member state it is established in (line 16). The lemma in line 18 tests whether everything works as expected: Since agent *e* is a conformity assessment body, it should be possible to deduce the obligation stated in R1 for it in the current world. Fortunately, the Sledgehammer tool finds a proof. Also, Nitpick confirms consistency, as

visible in 5.56. These results show that Sledgehammer can - at least in this example - consistently reason with the created representation.

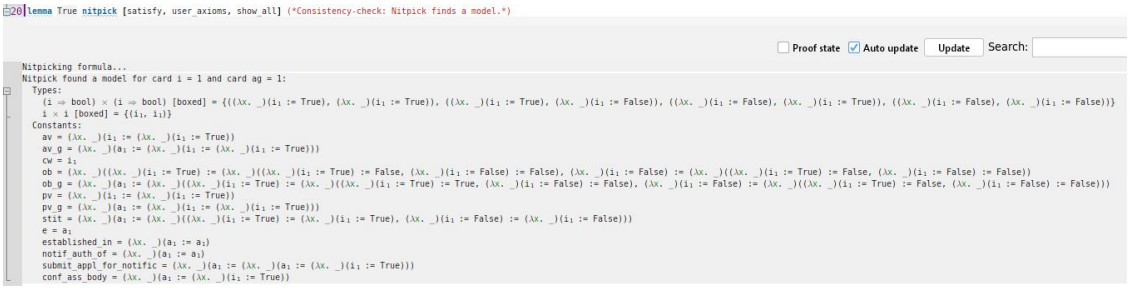


Figure 5.56: Nitpick model Article 31

Next, it shall be tested whether Extended DDL 2 can reason with CTDs. Therefore, the CTD from Article 16 (*Artificial Intelligence Act*, 2021) that has already been formalized in both DDL and Extended DDL will be represented in Extended DDL 2. The result is visible in Figure 5.57. Explanations are provided below.

```
theory "high-risk-3-3-16-DDL2"
  imports
  DDL_agents_mod
begin

(*CTD example extended DDL2:*)
consts
  d::ag
  l::aiSys
  provider_of::"ag⇒aiSys⇒σ"
  compliance_req_chap2::"aiSys⇒σ"
  inform_authorities::"aiSys⇒σ"

axiomatization where
(*rules, e.g. CTD*)
A1: "⌊∀x p. (high_risk x ∧ provider p ∧ provider_of p x) → ○p<stit p (compliance_req_chap2 x)>⌋" and
A8: "⌊∀x p. (high_risk x ∧ provider p ∧ provider_of p x) → (¬ (compliance_req_chap2 x) → ○p<(stit p (inform_authorities x))>)⌋" and
(*implicit: If the compliance with the requirements is a given, the provider is obligated to not inform authorities
of non-compliance (since that would make no sense*)
AX: "⌊∀x p. (high_risk x ∧ provider p ∧ provider_of p x) → (○p<(compliance_req_chap2 x) → ¬ stit p (inform_authorities x)>)⌋" and

(*facts*)
F1: "⌊(high_risk l)⌋ₜ" and
F2: "⌊(provider d)⌋ₜ" and
F3: "⌊(provider_of d l)⌋ₜ" and
Situation: "⌊¬ (compliance_req_chap2 l)⌋ₜ"

(***Some Experiments***)
lemma True nitpick [satisfy, user_axioms, show_all, card i=1] oops (*ran out of time*)

lemma "⌊○d<stit d (inform_authorities l)>⌋ₜ" using A8 F1 F2 F3 Situation by blast
lemma "⌊○d<¬ stit d (inform_authorities l)>⌋ₜ" nitpick [user_axioms, show_all, card i=1] oops (*ran out of time*)


end
```

Figure 5.57: CTD Article 16 in Extended DDL 2

In lines 8 and 9, constants for an agent d and an AI system l are declared. Lines 10-12 then declare the predicates needed to formalize the CTD. In addition to the predicates *compliance_req_chap2* and *inform_authorities*, which are already known from the previous formalizations in this or a similar version, the predicate *provider_of* defines the relationship between a provider and an AI system (line 10).

Afterwards, the axiomatization starts by formalizing the CTD in question using the rules from Article 16 (*Artificial Intelligence Act*, 2021) in lines 14-20. Axiom A1 states that if an AI system x is a high-risk system, and an agent p is a provider, and the agent p is the provider of the high-risk system x, the agent p is obligated to see to it that system x fulfills the requirements from chapter 2 (line 16). In axiom A8, it is claimed that if an AI system x is a high-risk system, and an agent p is a provider, and the agent p is the provider of the high-risk system x if system x does not fulfill the requirements from chapter 2, it is obligated for the agent p to see to it that the authorities are informed about system x (line 17). Afterward, the implicit obligation AX is formulated: If the system does indeed comply with the requirements in chapter 2, the agent p should not see to it that the authorities are informed (line 20).

Having declared the rules, the next part of the axiomatization defines facts about the current world: It is stated that system l is a high-risk system (line 23), that agent d is a provider (line 24), and that agent d is the provider of system l (line 25). Line 26 then defines the current situation, in which system l does not comply with the requirements in chapter 2.

As usual, Nitpick is used to perform a consistency check and find a model. Unfortunately, this is not successful: As visible in Figure 5.58, Nitpick runs out of time and can not find a model.

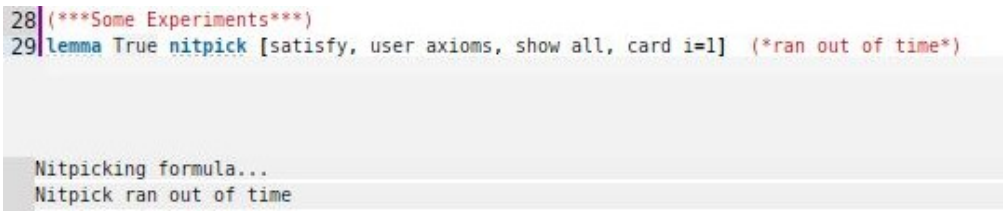


Figure 5.58: CTD Article 16 in Extended DDL 2: Nitpick runs out of time

Figure 5.59: CTD Article 16 in Extended DDL 2: No countermodel

Lines 31 and 32 contain two lemmas, of which the first one should be proven and the second one refuted within a faithful representation. As visible in Figure 5.57, Sledgehammer is able to prove the first lemma. When it comes to the second one, Sledgehammer does not find proof as desired. However, Nitpick is not able to find a counterexample either. Instead, it runs out of time again (Figure 5.59).

To ensure that the chosen CTD is not an exception or a singular problematic case, another CTD has been formalized in Extended DDL 2 (and Extended DDL), which led to the same observations. The Isabelle file containing this CTD can be found in Appendix Ten.

It is likely that the models become too complex for Nitpick to handle. The exact reason for the difficulties must be clarified in the future. For now, it can be concluded that Extended DDL 2 has problems with the formalization of CTDs. While it does not lead to obvious inconsistencies or contradictions, Nitpick fails to provide a model for the representations of the example CTDs and is unable to refute statements that it should disprove. This puts Extended DDL 2 at a disadvantage compared to Extended DDL, which does not struggle with the representation of CTDs.

In the following Section, both Extended DDL and Extended DDL 2 will be used to represent a complete article from the AIA. That way, it will be tested how well the reasoning

works in both variants with a larger excerpt involving multiple agents.

5.3.2.3 Representing Article 32 in Extended DDL/Extended DDL 2

Article 32 has been picked here since it involves four different types of agents: The EU Commission, notifying authorities, member states, and conformity assessment bodies. Also, Article 32 contains no CTDs. This is desirable because it is already known that Extended DDL 2 struggles with this modality. The full article reads as follows:

> "*Notification procedure*
>
> 1. Notifying authorities may notify only conformity assessment bodies that have satisfied the requirements laid down in Article 33.
> 2. Notifying authorities shall notify the Commission and the other Member States using the electronic notification tool developed and managed by the Commission.
> 3. The notification shall include full details of the conformity assessment activities, the conformity assessment module or modules, and the artificial intelligence technologies concerned.
> 4. The conformity assessment body concerned may perform the activities of a notified body only where no objections are raised by the Commission or the other Member States within one month of notification.
> 5. Notifying authorities shall notify the Commission and the other Member States of any subsequent relevant changes to the notification" (*Artificial Intelligence Act*, 2021, 59f.).

First, Article 32 will be represented using Extended DDL with four agents. Therefore, constants denoting the different agents are needed: constant c for the EU Commission, constant f for the notifying authority, constant h for the member state, and constant e for the conformity assessment bodies. The use of the singular is intentional here since - as explained above - Extended DDL only allows for representing a single generic agent that stands for a group of agents. The agent constants are included in the types file, which is imported into Extended DDL. It is the same as before, except for the addition of a type

for a notification. The complete Extended DDL file with four agents can be found in the appendix.

Additional predicates needed for the representation of Article 32 are shown in figure 5.60, including short explanations. Within the axiomatization, the rules from the article are declared as Isabelle statements (lines 21-30, Figure 5.61). Afterward, five facts about the current world are specified so that they can be used to employ the given rules (lines 32-37, Figure 5.61).

```
8  consts
9    notify::"ag⇒ag⇒σ" (*Notifying authorities notify conformity assessment body*)
10   satisfies_req_Art33::"ag⇒σ" (*Conformity assessment body satisfies requirements Article 33*)
11   notification_changes::"notification⇒σ" (*Notification for a conformity assessment body changes*)
12   inform::"ag⇒ag⇒(σ)⇒σ" (*Notifying authority informs another agent of the notification change*)
13   use_elec_not_tool::"ag⇒σ" (*Notifying authority uses electronic notification tool*)
14   includes_details_conf_ass_activities::"notification⇒σ" (*notification includes details on conformity assessment activities*)
15   includes_details_conf_ass_modules::"notification⇒σ" (*notification includes details on conformity assessment modules*)
16   includes_details_ai_used::"notification⇒σ" (*notification includes details on AI technology used*)
17   act_as_notified_body::"ag⇒σ" (*conformity assessment body acts as notified body*)
18   objection_raised_within_1m::"ag⇒(σ)⇒σ" (*objection raised by other agent on conformity assessment body acting as notified body*)
19   notification_for::"ag⇒notification"
```

Figure 5.60: Constants Article 32 Extended DDL

```
20 axiomatization where
21   (*rules from Article 32*)
22   A32_1: "⌊¬Of<¬ (stit f (notify f e))> ↔ (satisfies_req_Art33 e)⌋" and
23   A32_2: "⌊(stit f (notify f h) ∨ stit f (notify f c)) → Of<stit f (use_elec_not_tool f)>⌋" and
24   A32_3: "⌊(stit f (notify f e) →
25     Of<stit f ((includes_details_conf_ass_activities (notification_for e)) ∧ (includes_details_conf_ass_modules (notification_for e)) ∧
26   (includes_details_ai_used (notification_for e)))>)⌋" and
27   A32_4: "⌊¬Of<¬ stit f (act_as_notified_body e)>  ↔
28     ¬(objection_raised_within_1m h (act_as_notified_body e) ∧ (objection_raised_within_1m c (act_as_notified_body e)))⌋" and
29   A32_5: "⌊(notification_changes (notification_for e)) → Of<stit f (inform f c (notification_changes (notification_for e)) ∧
30   (inform f h (notification_changes (notification_for e))))>⌋" and
31
32   (*facts for tests of the rules*)
33   F1: "⌊satisfies_req_Art33 e⌋ι" and
34   F2: "⌊(stit f (notify f h))⌋ι" and
35   F3: "⌊stit f (notify f e)⌋ι" and
36   F4: "⌊¬(objection_raised_within_1m h (act_as_notified_body e) ∧ (objection_raised_within_1m c (act_as_notified_body e)))⌋ι" and
37   F5: "⌊notification_changes (notification_for e)⌋ι"
```

Figure 5.61: Axiomatization Article 32 Extended DDL

Finally, several lemmas that should hold in the situation described in F1-F5 are constructed, and Sledgehammer is employed to prove them (Figure 5.62). Sledgehammer uses F1 and rule 1 from Article 32 to prove T1, e.g., that it is permissible for the notifying authority *(f)* to see to it that the conformity assessment body *(e)* is notified (line 39). Using rule 2 and F2, Sledgehammer finds proof for lemma T2, which states that the notifying authority *(f)* is obligated to see to it that the electronic notification tool is used (line 40). Lemma T3

holds that the notifying authority *(f)* is obligated to see to it that the notification for the conformity assessment body *(e)* includes details about the conformity assessment activities, the conformity assessment modules, and the AI technology used. This can also be proved by Sledgehammer via rule 3 and F3 (lines 41f.). In line 43, Sledgehammer can prove that it is permissible for the notifying authority *(f)* to see to it that the conformity assessment body *(e)* acts as a notified body using rules A4 and F4. Finally, the lemma T5 claims that it is obligatory for the notifying authority *(f)* to see to it that the EU commission *(c)* and the member state *(h)* are informed if there are changes to the notification of the conformity assessment body *(e)*, which is proven by Sledgehammer via rule 5 and F5 (line 44f.).

```
39 lemma T1: "⌊¬⊙f<¬ (stit f (notify f e))>⌋ι" using A32_1 F1 by auto
40 lemma T2: "⌊⊙f<stit f (use_elec_not_tool f)>⌋ι" using A32_2 F2 by auto
41 lemma T3: "⌊⊙f<stit f ((includes_details_conf_ass_activities (notification_for e)) ∧
42    (includes_details_conf_ass_modules (notification_for e))  ∧ (includes_details_ai_used (notification_for e)))>⌋ι" using A32_3 F3 by auto
43 lemma T4: "⌊¬⊙f<¬ stit f (act_as_notified_body e)>⌋ι" using A32_4 F4 by auto
44 lemma T5: "⌊⊙f<stit f ((inform f c (notification_changes (notification_for e))) ∧ (inform f h (notification_changes (notification_for e))))>⌋ι"
45    using A32_5 F5 by auto
46
47 lemma True nitpick [satisfy, user_axioms, show_all]  (*Consistency-check: Nitpick finds a model.*)
```

Figure 5.62: Tests Article 32 Extended DDL

The fact that all the lemmas were proven as desired shows that Sledgehammer can reason with the created representation. Additionally, the consistency check in line 47 is successful, meaning that the set of axioms and definitions within the file is consistent and doesn't lead to contradictions. It is also visible here that the model involves four distinct agents (card ag=4, c=a2, e=a4, f=a3, h=a1), as desired (Figure 5.63). However, as already known, Nitpick only provides a model up to a cardinality of i=2.

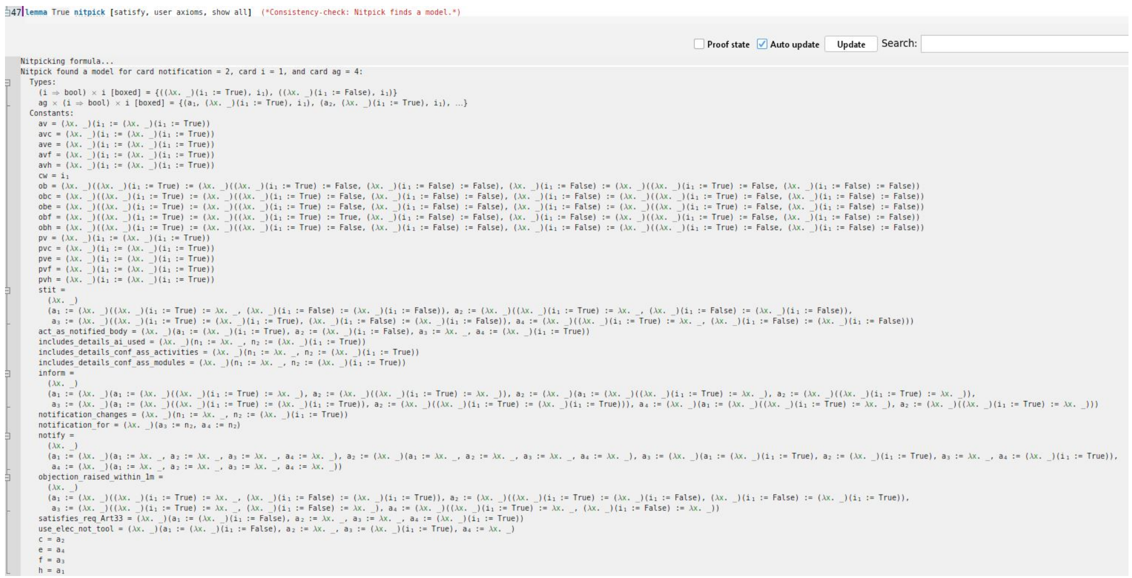


Figure 5.63: Nitpick model found Extended DDL

Next, the same article will be represented in Extended DDL 2. In addition to the constants in figure 5.60, three constants *f*, *h*, and *e* are introduced for the involved agents (Figure 5.64). No constant is needed for the EU commission because it is already defined in the types_2 file, which is imported in Extended DDL 2.

Afterward, the rules from Article 32 are represented as axioms (lines 24-36, Figure 5.65). These look different from the ones in Extended DDL due to the way agents are represented. Axiom A32_1 formalizes rule 1 and states that for all agents *a* and *b*, if *a* is a notifying authority and *b* a conformity assessment body, *a* may see to it that *b* is notified only if *b* satisfies the requirements in Article 33. The other rules are formalized similarly.

The second part of the axiomatization (lines 38-46, Figure 5.65) contains, again, facts about the current situation. There now are 14 axioms since the types of the agents *f*, *h*, and *e*, and the fact that they are of distinct types must be specified (F1-F3, Figure 5.65).

Next, the same five lemmas are proven via Sledgehammer. As in the formalization in Extended DDL, Sledgehammer successfully provides proof for all lemmas, using the rules from Article 32 and the facts about the current situation (Figure 5.66).

```
consts
  f::ag
  h::ag
  e::ag
  notify::"ag⇒ag⇒σ" (*Notifying authorities notify conformity assessment body*)
  satisfies_req_Art33::"ag⇒σ" (*Conformity assessment body satisfies requirements Article 33*)
  notification_changes::"notification⇒σ" (*Notification for a conformity assessment body changes*)
  inform::"ag⇒ag⇒(σ)⇒σ" (*Notifying authority informs another agent of the notification change*)
  use_elec_not_tool::"ag⇒σ" (*Notifying authority uses electronic notification tool*)
  includes_details_conf_ass_activities::"notification⇒σ" (*notification includes details on conformity assessment activities*)
  includes_details_conf_ass_modules::"notification⇒σ" (*notification includes details on conformity assessment modules*)
  includes_details_ai_used::"notification⇒σ" (*notification includes details on AI technology used*)
  act_as_notified_body::"ag⇒σ" (*conformity assessment body acts as notified body*)
  objection_raised_within_1m::"ag⇒(σ)⇒σ" (*objection raised by other agent on conformity assessment body acting as notified body*)
  notification_for::"ag⇒notification"
```

Figure 5.64: Constants Article 32 Extended DDL 2

```
axiomatization where
  (*rules from Article 32*)
  A32_1: "⌊∀a b. (notif_authority a) ∧ (conf_ass_body b) → ¬○a<¬ (stit a (notify a b))> ↔ (satisfies_req_Art33 b)⌋" and
  A32_2: "⌊∀a b. (notif_authority a) ∧ (member_state b) →
  (stit a (notify a b) ∨ stit a (notify a eu_comm)) → ○a<stit a (use_elec_not_tool a)>⌋" and
  A32_3: "⌊∀a b. (notif_authority a) ∧ (conf_ass_body b) → (stit f (notify a b) →
    ○a<stit a ((includes_details_conf_ass_activities (notification_for b)) ∧
    (includes_details_conf_ass_modules (notification_for b)) ∧ (includes_details_ai_used (notification_for b)))>)⌋" and
  A32_4: "⌊∀a b. (conf_ass_body a) ∧ (member_state b) →
     ¬○a<¬ stit a (act_as_notified_body a)> ↔
     ¬(objection_raised_within_1m b (act_as_notified_body a) ∧ (objection_raised_within_1m c (act_as_notified_body a)))⌋" and
  A32_5: "⌊∀a b d. ((conf_ass_body a) ∧ (notif_authority b) ∧ (member_state h))
    → (notification_changes (notification_for a)) → ○b<stit b ((inform b eu_comm (notification_changes (notification_for a))) ∧
    (inform b d (notification_changes (notification_for a))))>⌋" and

  (*facts for tests of the rules*)
  F1a: "⌊(notif_authority f)⌋ι" and F1b: "⌊¬(member_state f)⌋ι" and F1c: "⌊¬(conf_ass_body f)⌋ι"  and
  F2a: "⌊(member_state h)⌋ι" and F2b:"⌊¬(notif_authority h)⌋ι" and F2c: "⌊¬(conf_ass_body h)⌋ι"  and
  F3a: "⌊(conf_ass_body e)⌋ι" and F3b:"⌊¬(notif_authority e)⌋ι" and F3c: "⌊¬(member_state e)⌋ι" and
  F4: "⌊satisfies_req_Art33 e⌋ι" and
  F5: "⌊(stit f (notify f h))⌋ι" and
  F6: "⌊stit f (notify f e)⌋ι" and
  F7: "⌊¬(objection_raised_within_1m h (act_as_notified_body e) ∧ (objection_raised_within_1m eu_comm (act_as_notified_body e)))⌋ι" and
  F8: "⌊notification_changes (notification_for e)⌋ι"
```

Figure 5.65: Axiomatization Article 32 Extended DDL 2

```
lemma T1: "⌊¬○f<¬ (stit f (notify f e))>⌋ι" using A32_1 F1 F3 F4 by auto
lemma T2: "⌊○f<stit f (use_elec_not_tool f)>⌋ι" using A32_2 F1 F2 F5 by blast
lemma T3: "⌊○f<stit f ((includes_details_conf_ass_activities (notification_for e)) ∧ (includes_details_conf_ass_modules (notification_for e)) ∧
  (includes_details_ai_used (notification_for e)))>⌋ι" using A32_3 F1 F3 F6 by blast
lemma T4: "⌊¬○e<¬ stit e (act_as_notified_body e)>⌋ι" using A32_4 F2 F3 F7 by blast
lemma T5: "⌊○f<stit f ((inform f eu_comm (notification_changes (notification_for e))) ∧
   (inform f h (notification_changes (notification_for e))))>⌋ι" using A32_5 F1 F2 F3 F8 by auto

lemma True nitpick [satisfy, user_axioms, show_all, card ag=4] oops (*Consistency-check: Nitpick finds a model.*)
```

Figure 5.66: Test Article 32 Extended DDL 2

Also, the consistency test via Nitpick in line 56 is successful (Figure 5.67). A model with four distinct agents is found (card ag = 4), of which one is the EU commission (eu_comm = a2). If the cardinality of i is set to a number higher than one, Nitpick runs out of time and does not find a model.

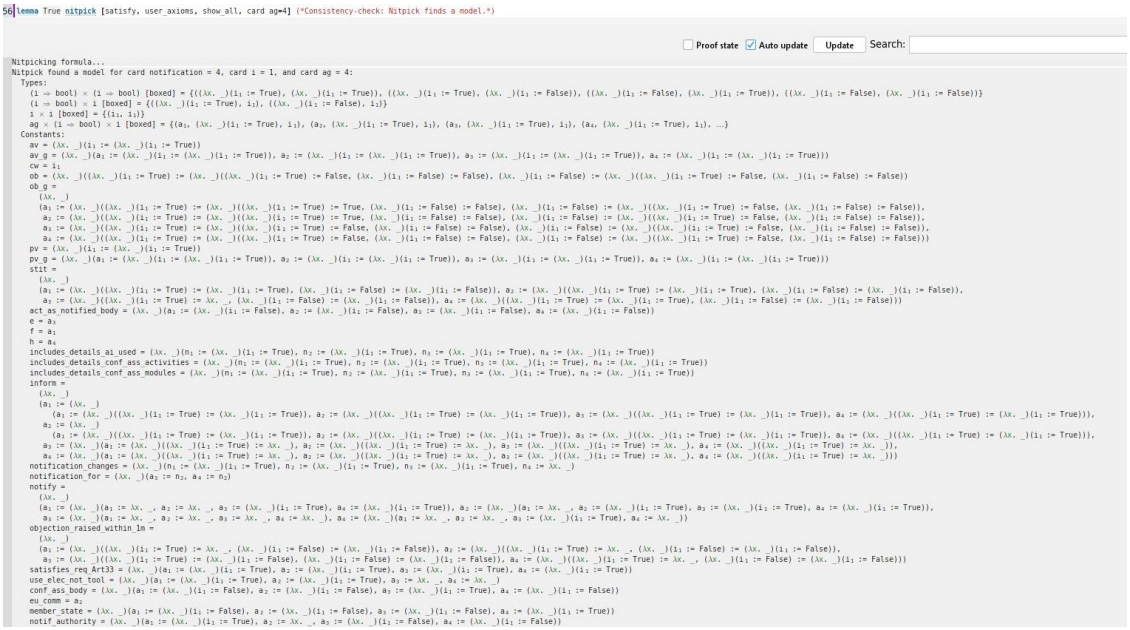


Figure 5.67: Nitpick Article 32 Extended DDL 2

To summarize, it was possible to formalize Article 32 in Extended DDL and Extended DDL 2. Even though both variants have their weaknesses, Sledgehammer can reason with the rules from the formalized Article 32 and draw correct conclusions within this example case.

To end the discussion of agency, the limitations of representing agency and agentive representations discovered during this thesis will be summarized.

5.3.3 Limitations of Representing Agency and Agentive Obligations

The last two sections have explored possibilities to represent agency and agentive obligations. Within the first section, the TDS logic by van Berkel and Lyon(2019) was identified as a suitable candidate and has been embedded in Isabelle (Chapter 5.3.1.1). However, when testing this embedding, it was discovered that Nitpick fails to find a model, which has led to the conclusion that TDS logic has infinite models. This is a big limitation since Nitpick can not work with such infinite models (J. Blanchette, 2023), ultimately making TDS logic unsuitable for the purpose of this thesis.

The second part of this Chapter has explored the option of extending DDL to integrate a STIT operator and agentive obligation operators. Within this process and the experimentation with examples from the AIA, two extensions of DDL were developed.

For the first one, which was referred to as Extended DDL, Nitpick finds a model up to a cardinality of i = 2. Also, all but one of the lemmas (lemma C_8) that were proven to hold for DDL by Benzmüller et al. (Benzmüller et al., 2022) were also proven for Extended DDL. Within the conducted experiments, neither Sledgehammer nor Nitpick had problems in reasoning with representations in Extended DDL.

However, the possibilities to model agents, types of agents, and relations between agents in Extended DDL are not ideal. Due to the usage of agent constants and accessibility relations related to exactly one of these constants, it is only possible to formalize obligations for generic agent types represented by the agent constants. This is limiting because it is rarely enough to state obligations that hold for all providers, importers, distributors, etc. Instead, obligations are declared for subgroups of agents fulfilling certain criteria or agents in specific relation to other agents, for example, providers of high-risk AI systems or notified bodies situated in a particular member state.

To summarize, Extended DDL achieves a consistent representation of the parts from the AIA that were tested. However, this representation is restricted in that it is not able to

capture the details of the types and sub-types of agents and their relation to one another.

To improve this, a second extension of DDL, called Extended DDL 2, has been developed. In this variant, the accessibility relations accept an agent as an input so that further constraints on that agent, like its type or relations to other agents, can easily be specified. Agents and their relations can hence be modeled more accurately in Extended DDL 2.

For Extended DDL 2, Nitpick also finds a model, but only for a cardinality i=1. Also, the testing of the lemmas postulated for DDL (Benzmüller et al., 2022) led to the same results as the tests for Extended DDL: All but one of the lemmas (lemma C_8) were proven to hold in Extended DDL 2.

One excerpt and one article from the AIA were formalized in Extended DDL 2. This was a success since, in both of these representations, Sledgehammer and Nitpick could correctly reason with the formalized rules. However, formalizing CTDs in Extended DDL 2 leads to problems. In two example cases, Nitpick failed to find a model for the formalized CTDs and was consequently not able to refute lemmas that should be refuted. This is a weakness of Extended DDL 2.

Neither Extended DDL nor Extended DDL 2 can provide an optimal solution to the problem of representing agency and agentive obligations. Also, not all problems of the two variants have been understood completely yet. However, Extended DDL 2, with its flexible representation of agents, seems to be the strategy more apt for capturing the AIA once its difficulties have been understood and resolved. This indicates a need for further research, analysis of the embeddings and the formalized parts from the AIA, and experimentation with additional examples.

Chapter 6

Conclusion

Finally, this last Chapter will summarize the key findings, discuss limitations, and present several open questions for future work.

6.1 Summary of Key Findings

In the first part of this thesis, a thorough analysis of the AIA was conducted, which has led to an answer to research question R1:

- **What modalities (deontic, epistemic, temporal, etc.) are relevant within the AIA?**

Within the analysis, the following modalities were identified: obligations (and permissions and prohibitions), fuzziness, exceptions, temporality, and temporal notions, agency and agentive obligations, CTDs, and beliefs of agents (Chapter Four). The number of different modalities within the AIA demonstrates its complexity and indicates that representing the document requires an expressive logic or combination of logics.

Since it was not feasible to focus on all the modalities, this thesis considered only obligations (Chapter 5.1), CTDs (Chapter 5.2), and agency and agentive obligations (Chapter 5.3). Unfortunately, this means that research questions R2 and R3 can only be answered for

the chosen modalities. The two research questions are related and will be answered jointly for each of the selected modalities. They read as follows:

- **How can concrete examples from the AIA, employing these modalities, be represented in Isabelle, and which logics need to be embedded for this purpose?**

- **Are the theorem provers available in Isabelle/HOL effective in reasoning with these representations?**

To represent obligations (as well as permissions and prohibitions), SDL was identified as a suitable logic in Chapter 4.1. Since a trustworthy embedding of SDL in Isabelle/HOL already exists (Benzmüller & Parent, 2018), this embedding could easily be used to represent examples from the AIA in Chapter 5.1.2. Additionally, an alternative strategy for representing exceptions in SDL was tested (Chapter 5.1.3) and proved to be a viable workaround. There were no problems with the representations of the example parts, and both Sledgehammer and Nitpick could reason with them correctly.

DDL is a suitable logic to represent CTDs (Chapter 4.6). Like SDL, DDL already has a faithful embedding in Isabelle/HOL (Benzmüller et al., 2022). This embedding could also be used to represent selected CTDs from the AIA without problems. The formalization of the selected examples from the AIA was straightforward, and fortunately, Nitpick and Sledgehammer could work with them successfully. Since DDL can express everything SDL can and more (Dov Gabbay et al., 2013), the DDL embedding is suitable for representing obligations and CTDs. Consequently, a logic fit to represent two of the chosen modalities has been discovered.

Finally, suitable logics for the representation of agency and agentive obligations were explored, which proved more challenging. Chapter 5.3 began by introducing an embedding of TDS by van Berkel and Lyon (2019) in Isabelle/HOL, a logic appropriate for representing both actions of agents and agentive obligations. While most axioms proclaimed by van

Berkel and Lyon were provable via Sledgehammer, and none were refuted (Chapter 5.3.1.2), Nitpick could not find a model for the embedding. This revealed that TDS logic has infinite models, making it unsuitable for model checking and representing the AIA.

The second part of Chapter 5.3 delved into extending DDL with additional operators to represent agency and agentive obligations. Two extensions were developed: Extended DDL (Chapter 5.3.2.1) and Extended DDL 2 (Chapter 5.3.2.2). For both variants, all but one of the lemmas postulated for DDL by Benzmüller et al. were proven. However, each has its limitations. Extended DDL restricts the representation of agents, with agentive obligations always applying to a generic agent type without the possibility of specifying further constraints or relations. In contrast, Extended DDL 2 offers flexibility in representing agents, allowing groupings based on characteristics or relations and formulating obligations for specific subsets of agents. However, Extended DDL 2 struggles with CTDs from the AIA, where Nitpick fails to find a model. While the representation does not lead to contradictions or inconsistencies, some incorrect claims cannot be refuted. In contrast, reasoning with CTDs in Extended DDL does not pose problems, giving it an advantage over Extended DDL 2 in this regard. Another benefit of Extended DDL is that Nitpick finds a model for it up to a cardinality of i=2, whereas for Extended DDL 2, Nitpick only does so for a cardinality of i=1.

Both DDL variants are suboptimal and have weaknesses. Nevertheless, they represent a starting point for the representation of agency and agentive obligations, which requires further development in the future.

To summarize, this thesis has produced a structured overview of the modalities that are relevant within the AIA and investigated possibilities for representing three of them (obligations, CTDs, and agency and agentive obligations) in Isabelle/HOL. It successfully represented excerpts from the AIA containing obligations and CTDs using the DDL embedding by Benzmüller et al. (2022) and demonstrated that Sledgehammer and Nitpick can reason with these representations. Additionally, it introduced different approaches for representing agency and agentive obligations and discussed their problems, which must be solved

in future work.

6.2 Limitations

It is the goal of this thesis to both analyze the AIA and its modalities and contribute towards its representation in a logical language within the LogiKEy framework (Benzmüller et al., 2020). Naturally, there are some limitations to this endeavor, which shall be discussed here.

While the AIA has been carefully analyzed, as shown in Chapter Four and the figures in Appendix One, this analysis was conducted from the lay perspective of someone not knowledgeable about or involved in the legal domain. Therefore, it is possible that parts of the AIA content were misinterpreted or overlooked. To discover and address any mistakes or shortcomings, it would be beneficial to seek feedback from a legal expert and review analyses conducted by other researchers.

Another limitation of this thesis is that, although many different modalities were identified during the analysis, practical attempts to represent them in Isabelle/HOL and formalize examples from the AIA have only been conducted with three modalities. Consequently, any analysis of the interplay of modalities also considers only these selected modalities. Therefore, while research question one was answered, research questions two and three were only addressed for the chosen modalities.

This thesis has successfully created faithful representations of example paragraphs containing obligations and CTDs using DDL (Chapters 5.1 and 5.2). However, no ideal solution for representing agency and agentive obligations has been found. Instead, several approaches were attempted: an embedding of TDS logic (van Berkel & Lyon, 2019) and two extensions of DDL, Extended DDL and Extended DDL 2 (Chapter 5.3.2). While shortcomings and problems of each approach were identified, it was not always possible to determine all their causes. For example, it remains unclear why Nitpick does not find models for Extended

DDL with a cardinality of i > 2 (Chapter 5.3.2.1), for Extended DDL 2 with a cardinality of i > 1 (Chapter 5.3.2.2), and for representations of CTDs in Extended DDL 2 (Chapter 5.2.2.2.1). Moreover, the question of why lemma C_8, which holds for DDL (Benzmüller et al., 2022), cannot be proven for Extended DDL and Extended DDL 2 (Chapter 5.3.2) remains unanswered.

This thesis also did not analyze or visualize in detail the models that Nitpick found for the embeddings of the different logics and the example representations. This would have been valuable in verifying the provided models or discovering possible problems.

Another limitation of the thesis is that only a few parts from the AIA, including the chosen modalities, were represented in Isabelle/HOL with the embedded logics. These examples are not representative of the full text. Consequently, it is possible that not all problems were discovered. Other sections of the AIA may contain constructions whose formalization leads to currently unknown difficulties with certain embeddings.

Finally, the last Section will list open questions to be studied in future research.

6.3 Open Questions and Future Work

As stated previously, this thesis was not able to completely answer all of the three research questions that have been formulated in the beginning. Additionally, difficulties of different logics, their embeddings, and Nitpick and Sledgehammer were discovered, which could not yet be overcome. These unresolved questions and problems are a good starting point for future projects and shall be recorded.

First, it would be sensible to focus on the modalities within the AIA that were not discussed in depth in this thesis. Open tasks include the identification of suitable logics to represent them, the creation of embeddings of these logics, and the experimentation with examples from the AIA to verify that the embedded logics work as intended.

Within Chapter 5.3.1, an embedding of TDS logic by van Berkel and Lyon (2019) has

been created. This logic has only infinite models and consequently is rather unsuitable for the representation of the AIA from a practical perspective. However, it would make sense to investigate strategies to weaken TDS logic to achieve finite models. If this were achieved, TDS logic could support an accurate representation of the AIA.

Moreover, as already discussed in the previous section, it was not possible to analyze or visualize the Nitpick models that were found during this thesis. This would make a valuable future project, contributing towards a deeper understanding of the models and the verification of their quality.

This is not the only open task relating to Nitpick. Nitpick is unable to find models for Extended DDL with a cardinality of i $>$ 2 and for Extended DDL 2 with a cardinality of i $>$ 1. The reasons for this have not been clarified yet but should be investigated in the future. Further analyses or visualizations of the Nitpick models found for lower cardinalities could support this process.

Another problem of the two extensions of DDL is that lemma C_8 which was postulated by Benzmüller et al. (2022) and holds for DDL could not be proven for Extended DDL and Extended DDL 2. It would be useful to understand why this is the case and investigate whether the embeddings can be adapted so that the lemma is provable.

Apart from answering the open questions on Extended DDL and Extended DDL 2, it is sensible to consider related research that could potentially solve the problems of the discussed approaches. For instance, while a flexible representation of agents, as achieved in Extended DDL 2, is desirable, its models are too complex and currently struggle with the representation of CTDs and higher cardinalities of i (potential worlds). A possible solution to this could be considering different semantics for DDL, for example, the less complex Hanson-type preference-based semantics (Danielsson, 1969; Hansson, 1969), which was discussed in-depth by Parent (Parent, 2021). In his work, Parent has also created an embedding of DDL logic with preference-based semantics in Isabelle/HOL, which is available in the LogiKEy GitHub repository (Benzmüller et al., 2023). It would be interesting to attempt the integration of a STIT operator and an operator for agentive obligations into this embedding, analyze the

performance of Nitpick and Sledgehammer, and check whether the problems discovered with Extended DDL 2 can be avoided.

Similarly, it would make sense to look at Term-modal Deontic Logic (TMDL) s as an alternative to Extended DDL. In TMDLs, obligation operators are indexed with variables or constants denoting agents and use ternary or quaternary accessibility relations instead of binary ones. Consequently, obligations can be indexed by agent terms, and these agent terms can be quantified. This strategy appears to solve the main problem of Extended DDL: The lack of the possibilities to quantify over types of agents and characterize subgroups or relations among them (Frijters, 2023). In future research, an embedding of TMDLs in HOL must be created. It should then be tested whether this embedding is a suitable option that solves the problems of Extended DDL and is manageable for Nitpick and Sledgehammer.

Finally, it is advisable to formalize larger parts of the AIA in Isabelle/HOL using the embedded logic discussed in this thesis or others created in future projects. This may lead to the discovery of new problems that must be handled before a complete representation of the AIA can be achieved. Another relevant factor is the combination of logics. While different logics can be used to formalize different modalities, they must eventually be combined to formalize the full document. In the case of obligations and CTDs, DDL has proven to represent both modalities. This might not always be the case, especially when all the modalities found in Chapter Four are included in the discussion. Hence, strategies that allow for the combination of different modalities and logics must be explored.

Appendix

1 Analysis of the AIA: Tables and Visualizations

Further visualizations created during the analysis of the AIA can be found on miro via the following links:

- First mind map
- Analysis of modalities

2 Extended DDL: Complete Isabelle/HOL File with Two Agents

Figures 1 and 2 show the complete Extended DDL file with two agents.

```
theory DDL_agents_clean  (*DDL including STIT operator and agentive obligations*)
  imports
    Main
    types
begin

consts
cw::i (*current world*)
av::"i⇒σ" pv::"i⇒σ" ob::"σ⇒(σ⇒bool)" (*general accessibility relations*)
avd::"i⇒σ" pvd::"i⇒σ" obd::"σ⇒(σ⇒bool)" (*accessibility relations for agent d*)
avb::"i⇒σ" pvb::"i⇒σ" obb::"σ⇒(σ⇒bool)" (*accessibility relations for agent b*)

(*stit operator*)
stit::"ag⇒σ⇒σ" (*ag sees to it that*)

axiomatization where
  ax_distinct: "d ≠ b" and
  ax_3a: "∀w.∃x. av(w)(x)" and ax_4a: "∀w x. av(w)(x) ⟶ pv(w)(x)" and ax_4b: "∀w. pv(w)(w)" and
  ax_5a: "∀X.¬ob(X)(λx. False)" and
  ax_5b: "∀X Y Z. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (ob(X)(Y) ⟷ ob(X)(Z))" and
  ax_5ca: "∀X β. ((∀Z. β(Z) ⟶ ob(X)(Z)) ∧ (∃Z. β(Z))) ⟶
      (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ ob(X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
  ax_5c: "∀X Y Z. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  ob(X)(Y)  ∧  ob(X)(Z))  ⟶  ob(X)(λw. Y(w) ∧ Z(w)))" and
  ax_5d: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ ob(X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                  ⟶ ob(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
  ax_5e: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ ob(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ ob(Y)(Z)" and

  (*for agent d: providers*)
  axd_3a: "∀w.∃x. avd(w)(x)" and axd_4a: "∀w x. avd(w)(x) ⟶ pvd(w)(x)" and axa_4ba: "∀w. pvd(w)(w)" and
  axd_5a: "∀X.¬obd(X)(λx. False)" and
  axd_5b: "∀X Y Z. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (obd(X)(Y) ⟷ obd(X)(Z))" and
  axd_5ca: "∀X β. ((∀Z. β(Z) ⟶ obd(X)(Z)) ∧ (∃Z. β(Z))) ⟶
      (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ obd(X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
  axd_5c: "∀X Y Z. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  obd(X)(Y)  ∧  obd(X)(Z))  ⟶  obd(X)(λw. Y(w) ∧ Z(w)))" and
  axd_5d: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obd(X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                  ⟶ obd(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
  axd_5e: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obd(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ obd(Y)(Z)" and

    (*for agent b: importers*)
  axb_3a: "∀w.∃x. avb(w)(x)" and axb_4a: "∀w x. avb(w)(x) ⟶ pvb(w)(x)" and axb_4ba: "∀w. pvb(w)(w)" and
  axb_5a: "∀X.¬obb(X)(λx. False)" and
  axb_5b: "∀X Y Z. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (obb(X)(Y) ⟷ obb(X)(Z))" and
  axb_5ca: "∀X β. ((∀Z. β(Z) ⟶ obb(X)(Z)) ∧ (∃Z. β(Z))) ⟶
      (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ obb(X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
  axb_5c: "∀X Y Z. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  obb(X)(Y)  ∧  obb(X)(Z))  ⟶  obb(X)(λw. Y(w) ∧ Z(w)))" and
  axb_5d: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obb(X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                  ⟶ obb(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
  axb_5e: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obb(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ obb(Y)(Z)" and

  stit1: "∀a F w. ((stit a F) w) ⟶ F w"
```

Figure 1: Extended DDL with 2 agents part 1

```
51
52 abbreviation ddlneg::γ ("¬_"[52]53) where "¬A ≡ λw. ¬A(w)"
53 abbreviation ddland::ϱ (infixr"∧"51) where "A∧B ≡ λw. A(w)∧B(w)"
54 abbreviation ddlor::ϱ (infixr"∨"50) where "A∨B ≡ λw. A(w)∨B(w)"
55 abbreviation ddlimp::ϱ (infixr"→"49) where "A→B ≡ λw. A(w)⟶B(w)"
56 abbreviation ddlequiv::ϱ (infixr"↔"48) where "A↔B ≡ λw. A(w)⟷B(w)"
57
58 abbreviation ddlbox::γ ("□") where "□A ≡ λw.∀v. A(v)"  (*A = (λw. True)*)
59 abbreviation ddldia::γ  ("◇") where "◇ A ≡ ¬□(¬A)"
60
61 (*Necessity/possibility for agents*)
62 abbreviation ddlboxa_g::ζ ("□ₐ") where "□ₐ rel A ≡ λw. (∀x. (rel (w)(x) ⟶ A(x)))"  (*in all actual worlds*)
63 abbreviation ddlboxp_g::ζ ("□ₚ") where "□ₚ rel A ≡ λw. (∀x. (rel (w)(x) ⟶ A(x)))" (*in all potential worlds*)
64 abbreviation ddldia_g::ζ ("◇ₐ") where "◇ₐ rel A ≡ ¬□ₐ rel (¬A)"
65 abbreviation ddldiap_g::ζ ("◇ₚ") where "◇ₚ rel A ≡ ¬□ₚ rel (¬A)"
66
67 (*generalised obligation operators with relation as a parameter*)
68 abbreviation ddlo_g::ν ("O _ (_|_)") where "O rel ⟨B|A⟩ ≡ λw. rel (A)(B)"  (*it ought to be A, given B *)
69 abbreviation ddloa_g::μ  ("Oₐ ") where "Oₐ rel1 rel2 A ≡ λw. rel1(rel2(w))(A) ∧ (∃x. rel2(w)(x) ∧ ¬A(x))" (*actual obligation*)
70 abbreviation ddlop_g::μ  ("Oₚ") where "Oₚ rel1 rel2 A ≡ λw. rel1(rel2(w))(A) ∧ (∃x. rel2(w)(x) ∧ ¬A(x))"  (*primary obligation*)
71
72 abbreviation ddltop::σ ("⊤") where "⊤ ≡ λw. True"
73 abbreviation ddlbot::σ ("⊥") where "⊥ ≡ λw. False"
74
75 (*Possibilist Quantification.*)
76 abbreviation ddlforall ("∀") where "∀Φ ≡ λw.∀x. (Φ x w)"
77 abbreviation ddlforallB (binder"∀"[8]9) where "∀x. φ(x) ≡ ∀φ"
78 abbreviation ddlexists ("∃") where "∃Φ ≡ λw.∃x. (Φ x w)"
79 abbreviation ddlexistsB (binder"∃"[8]9) where "∃x. φ(x) ≡ ∃φ"
80
81 abbreviation ddlvalid::"σ ⇒ bool" ("⌊_⌋"[7]105) where "⌊A⌋ ≡ ∀w. A w"   (*Global validity*)
82 abbreviation ddlvalidcw::"σ ⇒ bool" ("⌊_⌋ₜ"[7]105) where "⌊A⌋ₜ ≡ A cw" (*Local validity (in cw)*)
83
84 (* A is obligatory *)
85 abbreviation ddlobl::γ ("O<_>") where "O<A> ≡  0 ob ⟨A|⊤⟩"  (*New syntax: A is obligatory.*)
86 abbreviation ddlobld::γ ("Od<_>") where "Od<A> ≡  0 obd ⟨A|⊤⟩"  (*New syntax: A is obligatory for agent d.*)
87 abbreviation ddloblb::γ ("Ob<_>") where "Ob<A> ≡  0 obb ⟨A|⊤⟩"  (*New syntax: A is obligatory for agent b.*)
88
89 (* Consistency *)
90 lemma True nitpick [satisfy,user_axioms,show_all, card i = 2] oops
91
92 end
```

Figure 2: Extended DDL with 2 agents part 2

3 Extended DDL: Complete Isabelle/HOL File with Four Agents

Figures 3 and 4 show the complete Extended DDL file with four agents.

```
theory DDL_agents_4  (*DDL including STIT operator and agentive obligations*)
  imports
    Main
    types
begin

consts
cw::i (*current world*)
av::"i⇒σ" pv::"i⇒σ" ob::"σ⇒(σ⇒bool)" (*general accessibility relations*)
avh::"i⇒σ" pvh::"i⇒σ" obh::"σ⇒(σ⇒bool)" (*accessibility relations for agent b*)
avc::"i⇒σ" pvc::"i⇒σ" obc::"σ⇒(σ⇒bool)" (*accessibility relations for agent c*)
ave::"i⇒σ" pve::"i⇒σ" obe::"σ⇒(σ⇒bool)" (*accessibility relations for agent e*)
avf::"i⇒σ" pvf::"i⇒σ" obf::"σ⇒(σ⇒bool)" (*accessibility relations for agent f*)

(*stit operator*)
stit::"ag⇒σ⇒σ" (*ag sees to it that*)

axiomatization where
  ax_distinct1: "h ≠ c" and ax_distinct2: "h ≠ e" and
  ax_distinct3: "h ≠ f" and ax_distinct4: "c ≠ e" and
  ax_distinct5: "c ≠ f" and ax_distinct6: "e ≠ f" and

  ax_3a: "∀w.∃x. av(w)(x)" and ax_4a: "∀w x. av(w)(x) ⟶ pv(w)(x)" and ax_4b: "∀w. pv(w)(w)" and
  ax_5a: "∀X.¬ob(X)(λx. False)" and
  ax_5b: "∀X Y Z. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (ob(X)(Y) ⟷ ob(X)(Z))" and
  ax_5ca: "∀X β. ((∀Z. β(Z) ⟶ ob(X)(Z)) ∧ (∃Z. β(Z))) ⟶
      (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ ob(X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
  ax_5c: "∀X Y Z. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  ob(X)(Y)  ∧  ob(X)(Z))  ⟶  ob(X)(λw. Y(w) ∧ Z(w)))" and
  ax_5d: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ ob(X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                  ⟶ ob(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
  ax_5e: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ ob(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ ob(Y)(Z)" and

  (*for agent c: eu commission*)
  axc_3: "∀w.∃x. avc(w)(x)" and axl_4a: "∀w x. avc(w)(x) ⟶ pvc(w)(x)" and axm_4ba: "∀w. pvc(w)(w)" and
  axc_5a: "∀X.¬obc(X)(λx. False)" and
  axc_5b: "∀X Y Z. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (obc(X)(Y) ⟷ obc(X)(Z))" and
  axc_5ca: "∀X β. ((∀Z. β(Z) ⟶ obc(X)(Z)) ∧ (∃Z. β(Z))) ⟶
      (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ obc(X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
  axc_5c: "∀X Y Z. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  obc(X)(Y)  ∧  obc(X)(Z))  ⟶  obc(X)(λw. Y(w) ∧ Z(w)))" and
  axc_5d: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obc(X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                  ⟶ obc(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
  axc_5e: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obc(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ obc(Y)(Z)" and

  (*for agent e: conformity assessment bodies*)
  axe_3a: "∀w.∃x. ave(w)(x)" and axe_4a: "∀w x. ave(w)(x) ⟶ pve(w)(x)" and axe_4ba: "∀w. pve(w)(w)" and
  axe_5a: "∀X.¬obe(X)(λx. False)" and
  axe_5b: "∀X Y Z. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (obe(X)(Y) ⟷ obe(X)(Z))" and
  axe_5ca: "∀X β. ((∀Z. β(Z) ⟶ obe(X)(Z)) ∧ (∃Z. β(Z))) ⟶
      (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ obe(X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
  axe_5c: "∀X Y Z. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  obe(X)(Y)  ∧  obe(X)(Z))  ⟶  obe(X)(λw. Y(w) ∧ Z(w)))" and
  axe_5d: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obe(X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                  ⟶ obe(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
  axe_5e: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obe(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ obe(Y)(Z)" and
```

Figure 3: Extended DDL with 4 agents part 1

```
(*for agent f: notifying authority*)
axf_3a: "∀w.∃x. avf(w)(x)" and axf_4a: "∀w x. avf(w)(x) ⟶ pvf(w)(x)" and axf_4ba: "∀w. pvf(w)(w)" and
axf_5a: "∀X.¬obf(X)(λx. False)" and
axf_5b: "∀X Y Z. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (obf(X)(Y) ⟷ obf(X)(Z))" and
axf_5ca: "∀X β. ((∀Z. β(Z) ⟶ obf(X)(Z)) ∧ (∃Z. β(Z))) ⟶
    (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ obf(X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
axf_5c: "∀X Y Z. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  obf(X)(Y)  ∧  obf(X)(Z))  ⟶  obf(X)(λw. Y(w) ∧ Z(w)))" and
axf_5d: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obf(X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                 ⟶ obf(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
axf_5e: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obf(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ obf(Y)(Z)" and

(*for agent h: member state*)
axh_3a: "∀w.∃x. avh(w)(x)" and axh_4a: "∀w x. avh(w)(x) ⟶ pvh(w)(x)" and axh_4ba: "∀w. pvh(w)(w)" and
axh_5a: "∀X.¬obh(X)(λx. False)" and
axh_5b: "∀X Y Z. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (obh(X)(Y) ⟷ obh(X)(Z))" and
axh_5ca: "∀X β. ((∀Z. β(Z) ⟶ obh(X)(Z)) ∧ (∃Z. β(Z))) ⟶
    (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ obh(X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
axh_5c: "∀X Y Z. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  obh(X)(Y)  ∧  obh(X)(Z))  ⟶  obh(X)(λw. Y(w) ∧ Z(w)))" and
axh_5d: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obh(X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                 ⟶ obh(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
axh_5e: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ obh(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ obh(Y)(Z)" and

stit1: "∀a F w. ((stit a F) w) ⟶ F w"

abbreviation ddlneg::γ ("¬_"[52]53) where "¬A ≡ λw. ¬A(w)"
abbreviation ddland::ϱ (infixr"∧"51) where "A∧B ≡ λw. A(w)∧B(w)"
abbreviation ddlor::ϱ (infixr"∨"50) where "A∨B ≡ λw. A(w)∨B(w)"
abbreviation ddlimp::ϱ (infixr"⟶"49) where "A⟶B ≡ λw. A(w)⟶B(w)"
abbreviation ddlequiv::ϱ (infixr"⟷"48) where "A⟷B ≡ λw. A(w)⟷B(w)"

abbreviation ddlbox::γ ("□A") where "□A ≡ λw.∀v. A(v)"  (*A = (λw. True)*)
abbreviation ddldia::γ  ("◇A") where "◇ A ≡ ¬□(¬A)"

(*Necessity/possibility for agents*)
abbreviation ddlboxa_g::ζ ("□ₐ") where "□ₐ rel A ≡ λw. (∀x. (rel (w)(x) ⟶ A(x)))"  (*in all actual worlds*)
abbreviation ddlboxp_g::ζ ("□ₚ") where "□ₚ rel A ≡ λw. (∀x. (rel (w)(x) ⟶ A(x)))" (*in all potential worlds*)
abbreviation ddldia_g::ζ ("◇ₐ") where "◇ₐ rel A ≡ ¬□ₐ rel (¬A)"
abbreviation ddldiap_g::ζ ("◇ₚ") where "◇ₚ rel A ≡ ¬□ₚ rel (¬A)"

(*generalised obligation operators with relation as a parameter*)
abbreviation ddlo_g::ν ("O _ ⟨_|_⟩") where "O rel ⟨B|A⟩ ≡ λw. rel (A)(B)"  (*it ought to be A, given B *)
abbreviation ddloa_g::μ  ("Oₐ ") where "Oₐ rel1 rel2 A ≡ λw. rel1(rel2(w))(A) ∧ (∃x. rel2(w)(x) ∧ ¬A(x))" (*actual obligation*)
abbreviation ddlop_g::μ  ("Oₚ") where "Oₚ rel1 rel2 A ≡ λw. rel1(rel2(w))(A) ∧ (∃x. rel2(w)(x) ∧ ¬A(x))"  (*primary obligation*)

abbreviation ddltop::σ ("⊤") where "⊤ ≡ λw. True"
abbreviation ddlbot::σ ("⊥") where "⊥ ≡ λw. False"

(*Possibilist Quantification.*)
abbreviation ddlforall ("∀") where "∀Φ ≡ λw.∀x. (Φ x w)"
abbreviation ddlforallB (binder"∀"[8]9) where "∀x. φ(x) ≡ ∀φ"
abbreviation ddlexists ("∃") where "∃Φ ≡ λw.∃x. (Φ x w)"
abbreviation ddlexistsB (binder"∃"[8]9) where "∃x. φ(x) ≡ ∃φ"

abbreviation ddlvalid::"σ ⇒ bool" ("⌊_⌋"[7]105) where "⌊A⌋ ≡ ∀w. A w"   (*Global validity*)
abbreviation ddlvalidcw::"σ ⇒ bool" ("⌊_⌋₁"[7]105) where "⌊A⌋₁ ≡ A cw" (*Local validity (in cw)*)

(* A is obligatory *)
abbreviation ddlobl::γ ("O<_>") where "O<A> ≡  0 ob ⟨A|⊤⟩"  (*New syntax: A is obligatory.*)
abbreviation ddloblc::γ ("Oc<_>") where "Oc<A> ≡  0 obc ⟨A|⊤⟩"  (*New syntax: A is obligatory for agent c.*)
abbreviation ddloble::γ ("Oe<_>") where "Oe<A> ≡  0 obe ⟨A|⊤⟩"  (*New syntax: A is obligatory for agent e.*)
abbreviation ddloblf::γ ("Of<_>") where "Of<A> ≡  0 obf ⟨A|⊤⟩"  (*New syntax: A is obligatory for agent f.*)
abbreviation ddloblh::γ ("Oh<_>") where "Oh<A> ≡  0 obh ⟨A|⊤⟩"  (*New syntax: A is obligatory for agent h.*)

(* Consistency *)
lemma True nitpick [satisfy, user_axioms, show_all, card ag=4, card i=2] oops

end
```

Figure 4: Extended DDL with 4 agents part 2

4 Extended DDL 2: Complete Isabelle/HOL File

Figure 5 shows the complete Extended DDL 2 file.

```
theory DDL_agents_mod  (*DDL including STIT operator and agentive obligations*)
  imports
    Main
    types_2
begin

consts
 cw::i (*current world*)
 av::"i⇒σ" pv::"i⇒σ" ob::"σ⇒(σ⇒bool)" (*general accessibility relations*)

 av_g::"ag⇒i⇒σ" pv_g::"ag⇒i⇒σ" ob_g::"ag⇒(σ⇒(σ⇒bool))" (*agent-dependent accessibility relations*)

 (*stit operator*)
 stit::"ag⇒σ⇒σ" (*ag sees to it that*)

axiomatization where
 ax_3a: "∀w.∃x. av(w)(x)" and ax_4a: "∀w x. av(w)(x) ⟶ pv(w)(x)" and ax_4b: "∀w. pv(w)(w)" and
 ax_5a: "∀X.¬ob(X)(λx. False)" and
 ax_5b: "∀X Y Z. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (ob(X)(Y) ⟷ ob(X)(Z))" and
 ax_5ca: "∀X β. ((∀Z. β(Z) ⟶ ob(X)(Z)) ∧ (∃Z. β(Z))) ⟶
     (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ ob(X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
 ax_5c: "∀X Y Z. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  ob(X)(Y)  ∧  ob(X)(Z))  ⟶  ob(X)(λw. Y(w) ∧ Z(w)))" and
 ax_5d: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ ob(X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                  ⟶ ob(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
 ax_5e: "∀X Y Z. ((∀w. Y(w) ⟶ X(w)) ∧ ob(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ ob(Y)(Z)" and

 (*agent-dependent axioms*)
 axg_3a: "∀w a.∃x. av_g a (w)(x)" and axg_4a: "∀w x a. av_g a (w)(x) ⟶ pv_g a (w)(x)" and axg_4ba: "∀w a. pv_g a (w)(w)" and
 axg_5a: "∀X a.¬ob_g a (X)(λx. False)" and
 axg_5b: "∀X Y Z a. (∀w. ((Y(w) ∧ X(w)) ⟷ (Z(w) ∧ X(w)))) ⟶ (ob_g a(X)(Y) ⟷ ob_g a (X)(Z))" and
 axg_5ca: "∀X β a. ((∀Z. β(Z) ⟶ ob_g a(X)(Z)) ∧ (∃Z. β(Z))) ⟶
     (((∃y. ((λw. ∀Z. (β Z) ⟶ (Z w))(y) ∧ X(y))) ⟶ ob_g a (X)(λw. ∀Z. (β Z) ⟶ (Z w))))" and
 axg_5c: "∀X Y Z a. (((∃w. (X(w) ∧ Y(w) ∧ Z(w))) ∧  ob_g a (X)(Y)  ∧  ob_g a(X)(Z))  ⟶  ob_g a(X)(λw. Y(w) ∧ Z(w)))" and
 axg_5d: "∀X Y Z a. ((∀w. Y(w) ⟶ X(w)) ∧ ob_g a (X)(Y) ∧ (∀w. X(w) ⟶ Z(w)))
                  ⟶ ob_g a(Z)(λw. (Z(w) ∧ ¬X(w)) ∨ Y(w))" and
 axg_5e: "∀X Y Z a. ((∀w. Y(w) ⟶ X(w)) ∧ ob_g a(X)(Z) ∧ (∃w. Y(w) ∧ Z(w))) ⟶ ob_g a(Y)(Z)" and

 stit1: "∀a F w. ((stit a F) w) ⟶ F w"

 abbreviation ddlneg::γ ("¬_"[52]53) where "¬A ≡ λw. ¬A(w)"
 abbreviation ddland::φ (infixr"∧"51) where "A∧B ≡ λw. A(w)∧B(w)"
 abbreviation ddlor::φ (infixr"∨"50) where "A∨B ≡ λw. A(w)∨B(w)"
 abbreviation ddlimp::φ (infixr"→"49) where "A→B ≡ λw. A(w)⟶B(w)"
 abbreviation ddlequiv::φ (infixr"↔"48) where "A↔B ≡ λw. A(w)⟷B(w)"
 abbreviation ddlbox::γ ("□") where "□A ≡ λw.∀v. A(v)"
 abbreviation ddldia::γ  ("◇") where "◇ A ≡ ¬□(¬A)"

 (*Necessity/possibility for agents*)
 abbreviation ddlboxa_g::η ("□ₐ") where "□ₐ i A ≡ λw. (∀x. av_g i (w)(x) ⟶ A(x))"  (*in all actual worlds*)
 abbreviation ddlboxp_g::η ("□ₚ") where "□ₚ i A ≡ λw. (∀x. pv_g i (w)(x) ⟶ A(x))" (*in all potential worlds*)
 abbreviation ddldiaa_g::η ("◇ₐ") where "◇ₐ i A ≡ ¬□ₐ i (¬A)"
 abbreviation ddldiap_g::η ("◇ₚ") where "◇ₚ i A ≡ ¬□ₚ i (¬A)"
 (*generalised operators with agents as a parameter*)
 abbreviation ddlo_g::χ ("O _ ⟨_|_⟩") where "O i ⟨B|A⟩ ≡ λw. ob_g i A B"  (*Agent i ought to A, given B *)
 abbreviation ddloa_g::η  ("Oₐ _") where "Oₐ i A ≡ λw. ob_g i (av_g i (w))(A) ∧ (∃x. av_g i (w)(x) ∧ ¬A(x))" (*actual obligation*)
 abbreviation ddlop_g::η  ("Oₚ _") where "Oₚ i A ≡ λw. ob_g i (pv_g i (w))(A) ∧ (∃x. pv_g i (w)(x) ∧ ¬A(x))"  (*primary obligation*)

 (*non-agentive necessity, possibility and obligation operators*)
 abbreviation ddlboxa::γ ("□ₐ") where "□ₐA ≡ λw. (∀x. av(w)(x) ⟶ A(x))"  (*in all actual worlds*)
 abbreviation ddlboxp::γ ("□ₚ") where "□ₚA ≡ λw. (∀x. pv(w)(x) ⟶ A(x))" (*in all potential worlds*)
 abbreviation ddldiaa::γ ("◇ₐ") where "◇ₐA ≡ ¬□ₐ(¬A)"
 abbreviation ddldiap::γ ("◇ₚ") where "◇ₚA ≡ ¬□ₚ(¬A)"
 abbreviation ddlo::φ ("O⟨_|_⟩"[52]53) where "O⟨B|A⟩ ≡ λw. ob(A)(B)"  (*it ought to be ψ, given φ *)
 abbreviation ddloa::γ  ("Oₐ") where "OₐA ≡ λw. ob(av(w))(A) ∧ (∃x. av(w)(x) ∧ ¬A(x))" (*actual obligation*)
 abbreviation ddlop::γ  ("Oₚ") where "OₚA ≡ λw. ob(pv(w))(A) ∧ (∃x. pv(w)(x) ∧ ¬A(x))"  (*primary obligation*)

 abbreviation ddltop::σ ("⊤") where "⊤ ≡ λw. True"
 abbreviation ddlbot::σ ("⊥") where "⊥ ≡ λw. False"

(*Possibilist Quantification.*)
 abbreviation ddlforall ("∀") where "∀Φ ≡ λw.∀x. (Φ x w)"
 abbreviation ddlforallB (binder"∀"[8]9) where "∀x. φ(x) ≡ ∀φ"
 abbreviation ddlexists ("∃") where "∃Φ ≡ λw.∃x. (Φ x w)"
 abbreviation ddlexistsB (binder"∃"[8]9) where "∃x. φ(x) ≡ ∃φ"

 abbreviation ddlvalid::"σ ⇒ bool" ("|_|"[7]105) where "|A| ≡ ∀w. A w"   (*Global validity*)
 abbreviation ddlvalidcw::"σ ⇒ bool" ("|_|₁"[7]105) where "|A|₁ ≡ A cw" (*Local validity (in cw)*)

 (*New syntax *)
 abbreviation ddlobl::γ ("O<_>") where "O<A> ≡  O⟨A|⊤⟩"
 abbreviation ddlobl_g::η ("O_<_>") where "O i <A> ≡  O i ⟨A|⊤⟩"

 (* Consistency *)
 lemma True nitpick [satisfy,user_axioms,show_all,card i=1] (*no model for i>1*) oops

end
```

Figure 5: Extended DDL 2

5 Test Files for DDL

Figures 6 and 7 shows the tests for DDL by Benzmüller et al. (2022).

```
theory CJ_DDL_Tests imports DDL          (* Christoph Benzmüller, Ali Farjami, Xavier Parent, 2020  *)
begin (* Some Tests on the Meta-Theory of DDL*)
lemma True nitpick [satisfy,user_axioms,expect=genuine] oops  (* Consistency confirmed by Nitpick *)

lemma MP: "⌊[A]; ⌊A → B⌋⌋ ⟹ ⌊B⌋" by simp
lemma Nec: "⌊A⌋ ⟹ ⌊□A⌋" by simp
lemma Neca: "⌊A⌋ ⟹ ⌊□ₐA⌋"  by simp
lemma Necp: "⌊A⌋ ⟹ ⌊□ₚ A⌋"  by simp

(* "□" is an S5 modality *)
lemma C_1_refl: "⌊□A → A⌋" by simp
lemma C_1_trans: "⌊□A → (□(□A))⌋"  by simp
lemma C_1_sym: "⌊A → (□(◇A))⌋"  by simp

(* "□ₚ" is an KT modality *)
lemma C_9_p_refl: "⌊□ₚA → A⌋"  by (simp add: ax_4b)
lemma "⌊□ₚA → (□ₚ(□ₚA))⌋" nitpick [user_axioms] oops (* countermodel *)
lemma "⌊A → (□ₚ(◇ₚA))⌋"   nitpick [user_axioms] oops (* countermodel *)

(* "□ₐ" is an KD modality *)
lemma C_10_a_serial: "⌊□ₐA → ◇ₐA⌋"  by (simp add: ax_3a)
lemma "⌊□ₐA → A⌋" nitpick [user_axioms] oops (* countermodel *)
lemma "⌊□ₐA → (□ₐ(□ₐA))⌋" nitpick [user_axioms] oops (* countermodel *)
lemma "⌊A → (□ₐ(◇ₐA))⌋" nitpick [user_axioms] oops (* countermodel *)

(* Relationship between "□,□ₐ,□ₚ" *)
lemma C_11: "⌊□A → □ₚA⌋" sledgehammer by simp
lemma C_12: "⌊□ₚA → □ₐA⌋"  using ax_4a by auto

(* Observation II-2-1 *)
lemma ax_5b': "ob X Y ⟷ ob X (λz. X z ∧ Y z)" by (metis (no_types, lifting) ax_5b)
lemma ax_5b'': "ob X Y ⟷ ob X (λz. Y z ∧ X z)" by (metis (no_types, lifting) ax_5b)

(* Characterisation of "O" *)
lemma C_2: "⌊O⟨A|B⟩ → ◇(B ∧ A)⌋"  by (metis ax_5a ax_5b)
lemma C_3: "⌊(◇(A ∧ B ∧ C) ∧ O⟨B|A⟩ ∧ O⟨C|A⟩ ) → O⟨(B ∧ C)|A⟩⌋" using ax_5c by auto
lemma C_4: "⌊(□(A → B) ∧ (◇(A ∧ C)) ∧ O⟨C|B⟩) → O⟨C|A⟩⌋"   using ax_5e by blast
lemma C_5: "⌊□(A ↔ B) → (O⟨C|A⟩ ↔ O⟨C|B⟩)⌋"  by presburger
lemma C_6: "⌊□(C → (A ↔ B)) → (O⟨A|C⟩ ↔ O⟨B|C⟩)⌋"  by (smt ax_5b)
lemma C_7: "⌊O⟨B|A⟩ → □(O⟨B|A⟩)⌋"  by blast
lemma C_8: "⌊O⟨B|A⟩ → O⟨(A → B)|T⟩⌋"
 proof -
  have  "∀X Y Z. (ob X Y ∧ (∀w. X w  ⟶ Z w)) ⟶ ob Z (λw. (Z w ∧ ¬X w) ∨ Y w)" by (smt ax_5d  ax_5b ax_5b'')
  thus ?thesis  using ax_5b by fastforce qed
```

Figure 6: Tests for DDL part 1

```
(* Relationship between "O_a, O_p, □_a, □_p" *)
lemma C_13_a: "⌊□ₐA ⟶ (¬Oₐ⊥ ∧ ¬Oₐ(¬A))⌋"  by (metis (full_types) ax_5a ax_5b)
lemma C_13_b: "⌊□ₚA ⟶ (¬Oₚ⊥ ∧ ¬Oₚ(¬A))⌋"   by (metis (full_types) ax_5a ax_5b)
lemma C_14_a: "⌊□ₐ(A ↔ B) ⟶ (OₐA ↔ OₐB)⌋"  by (metis ax_5b)
lemma C_14_b: "⌊□ₚ(A ↔ B) ⟶ (OₚA ↔ OₚB)⌋"  by (metis ax_5b)

(* Relationship between "O, O_a, O_p, □_a, □_p" *)
lemma C_15_a: "⌊(O⟨B|A⟩ ∧ □ₐA ∧ ◇ₚB ∧ ◇ₐ(¬B)) ⟶ OₐB⌋"  using ax_5e by blast
lemma C_15_b: "⌊(O⟨B|A⟩ ∧ □ₚA ∧ ◇ₚB ∧ ◇ₚ(¬B)) ⟶ OₚB⌋"  using ax_5e by blast

(* Soundness and consistency *)
lemma II_3_1: "((⌊O⟨B|A⟩⌋) ∧ (∃x. Z(x) ∧ A(x) ∧ B(x))) ⟶ ob(Z)(A ⟶ B)"
  proof
     assume "(⌊O⟨B|A⟩⌋) ∧ (∃x. Z(x) ∧ A(x) ∧ B(x))"
      hence "ob (λz. A z ∧ Z z) (λz. A z ∧ Z z ∧ B z)" using ax_5e ax_5b  ax_5b' ax_5d by smt
      hence "ob (λz. Z z ∧ A z) (λz. Z z ∧ A z ∧ B z)" using ax_5e ax_5b  ax_5b' ax_5d by smt
      hence "ob Z (λw. (Z w ∧ ¬(Z w ∧ A w)) ∨ (Z w ∧ A w ∧ B w))" by (metis (mono_tags) ax_5d)
      from this show  L19: "ob(Z)(A ⟶ B)" by (smt ax_5b)  qed

(* Some theorems and derived (proof) rules *)
lemma II_4_1: "⌊□(A ↔ B) ⟶ (C(A) ↔ C(B))⌋"  using ext by blast
lemma obs_II_4_1_a  : "⌊A ↔ B⌋ ⟹ ⌊C(A) ↔ C(B)⌋"  using ext by blast
lemma obs_II_4_1_b  : "⌊A ↔ B⌋ ⟹ ⌊(◇(A ∧ C) ∧ O⟨C|B⟩) ⟶ O⟨C|A⟩⌋"  using ax_5e by blast
lemma obs_II_4_1_c_1: "⌊◇(O⟨B|A⟩) ⟶ ◇(□(O⟨B|A⟩))⌋"  by blast
lemma obs_II_4_1_c_2: "⌊◇(□(O⟨B|A⟩)) ⟶ ◇(O⟨B|A⟩)⌋"  by auto
lemma obs_II_4_1_c_3: "⌊◇(O⟨B|A⟩) ⟶ □(O⟨B|A⟩)⌋"  by blast
lemma obs_II_4_1_c_4: "⌊◇(¬(O⟨B|A⟩)) ⟶ □(¬(O⟨B|A⟩))⌋"  by blast
lemma res_II_4_1_a_1: "⌊¬(O⟨⊥|A⟩)⌋"  by (simp add: ax_5a)
lemma res_II_4_1_a_2: "⌊(◇ₚ(A ∧ B ∧ C) ∧ O⟨B|A⟩ ∧ O⟨C|A⟩) ⟶ O⟨(B ∧ C)|A⟩⌋"  using C_3  by auto
lemma res_II_4_1_a_3: "⌊O⟨B|A⟩ ⟶ O⟨B|(A ∧ B)⟩⌋"  by (smt ax_5a ax_5b ax_5e)
lemma res_II_4_1_a_4: "⌊◇ₚ(O⟨B|A⟩) ⟶ □ₚ(O⟨B|(A ∧ B)⟩)⌋"  by (smt ax_5a ax_5b ax_5e)
lemma res_II_4_1_a_5: "⌊(◇ₚ(A ∧ B ∧ C) ∧ O⟨C|A⟩) ⟶ O⟨C|(A ∧ B)⟩⌋"  by (smt ax_5a ax_5b ax_5e)
lemma res_II_4_1_b_1:  "⌊A ↔ B⌋ ⟹ ⌊O⟨C|A⟩ ↔ O⟨C|B⟩⌋"  by (smt ax_5a ax_5b ax_5e)
lemma res_II_4_1_b_2:  "⌊C ⟶ (A ↔ B)⌋ ⟹ ⌊O⟨A|C⟩ ↔ O⟨B|C⟩⌋"  by (smt ax_5b)
lemma obs_II_4_2_1: "⌊(O⟨B|A⟩ ∧ ◇ₐ(A ∧ B) ∧ ◇ₐ(A ∧ ¬B)) ⟶ (O⟨B|A⟩ ∧ ◇ₐ(A ⟶ B) ∧ ◇ₐ(¬(A ⟶ B)))⌋"  by blast
lemma obs_II_4_2_2: "⌊O⟨B|A⟩ ⟶ O⟨(A ⟶ B)|⊤⟩⌋" by (simp add: C_8)
lemma obs_II_4_2_3: "⌊(O⟨(A ⟶ B)|⊤⟩ ∧ □ₐ⊤ ∧ ◇ₐ(A ⟶ B) ∧ ◇ₐ(¬(A ⟶ B))) ⟶ Oₐ(A ⟶ B)⌋"  by (smt ax_5e)
lemma obs_II_4_2_4: "⌊□ₐ⊤⌋"  by simp
lemma obs_II_4_2_5: "⌊(O⟨(A ⟶ B)|⊤⟩ ∧ ◇ₐ(A ⟶ B) ∧ ◇ₐ(¬(A ⟶ B))) ⟶ Oₐ(A ⟶ B)⌋"  by (smt ax_5e)
lemma obs_II_4_2_6: "⌊(O⟨B|A⟩ ∧ ◇ₐ(A ∧ B) ∧ ◇ₐ(A ∧ ¬B)) ⟶ Oₐ(A ⟶ B)⌋"   by (simp add: II_3_1)
lemma obs_II_4_2_6_p: "⌊(O⟨B|A⟩ ∧ ◇ₚ(A ∧ B) ∧ ◇ₚ(A ∧ ¬B)) ⟶ Oₚ(A ⟶ B)⌋"   by (simp add: II_3_1)

lemma Oa_C: "⌊◇ₐ(A ∧ B) ∧ OₐA ∧ OₐB ⟶  Oₐ(A ∧ B)⌋" using ax_5c by auto
lemma Op_C: "⌊◇ₚ(A ∧ B) ∧ OₚA ∧ OₚB ⟶  Oₚ(A ∧ B)⌋" using ax_5c by auto
lemma Oa_DD: "⌊(OₐA ∧ O⟨B|A⟩ ∧ ◇ₐ(A ∧ B)) ⟶ Oₐ(A ∧ B)⌋" using ax_5b ax_5c obs_II_4_2_6 by smt
declare [[smt_timeout=300]]
lemma Op_DD: "⌊(OₚA ∧ O⟨B|A⟩ ∧ ◇ₚ(A ∧ B)) ⟶ Oₚ(A ∧ B)⌋" using ax_5b ax_5c obs_II_4_2_6_p by smt

end
```

Figure 7: Tests for DDL part 2

6 Test File for Extended DDL with Two Agents

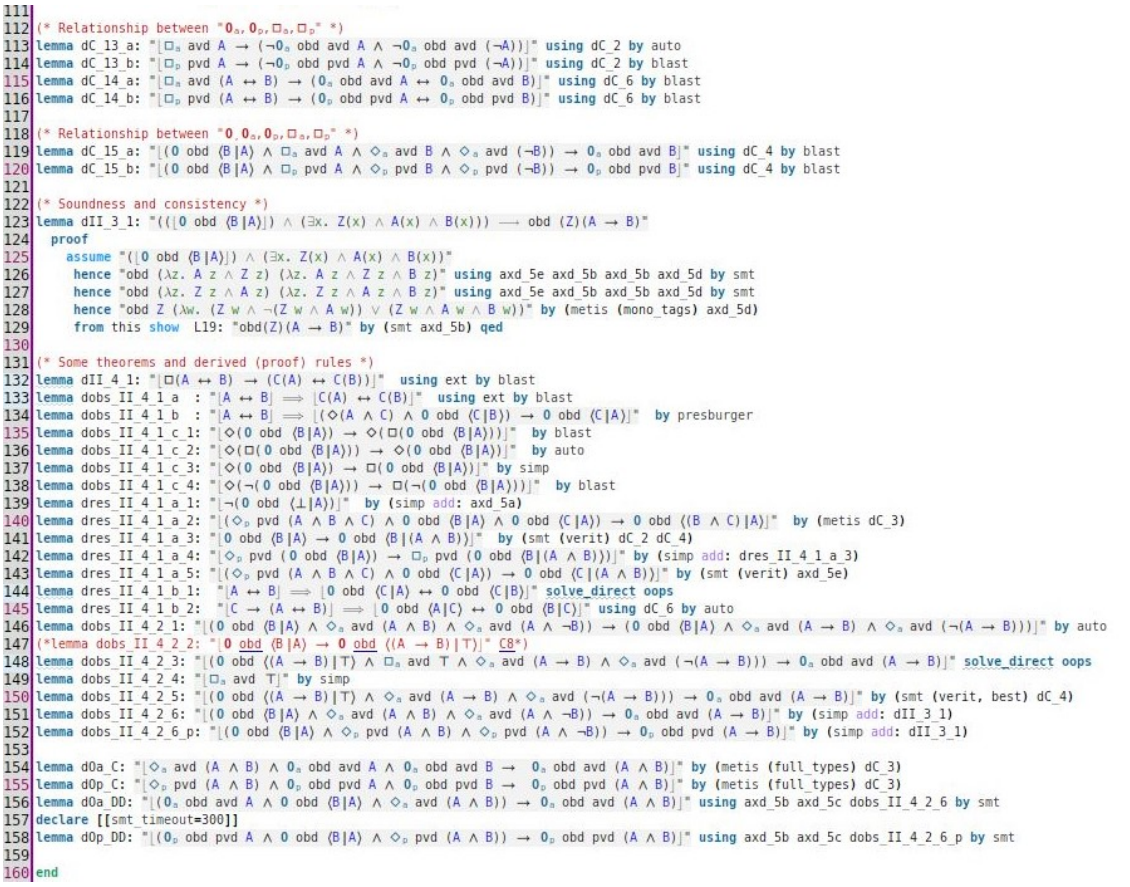
Figures 8, 9, and 10 show the tests (created by Benzmüller et al. (2022)) for Extended DDL with two agents.

```
theory CJ_DDL_agents_Tests imports DDL_agents_clean          (* Christoph Benzmüller, Ali Farjami, Xavier Parent, 2020  *)
begin (* Some Tests on the Meta-Theory of DDL*)
lemma True nitpick [satisfy,user_axioms,expect=genuine] oops  (* Consistency confirmed by Nitpick *)

lemma MP: "⌊⌊A⌋ ; ⌊A → B⌋⌋ ⟹ ⌊B⌋" by simp
lemma Nec: "⌊A⌋ ⟹ ⌊□A⌋" by simp
lemma Neca: "⌊A⌋ ⟹ ⌊□ₐ av A⌋"  by simp
lemma Necp: "⌊A⌋ ⟹ ⌊□ₚ pv A⌋"  by simp

(* "□" is an S5 modality *)
lemma C_1_refl: "⌊□A → A⌋" by simp
lemma C_1_trans: "⌊□A → (□(□A))⌋"  by simp
lemma C_1_sym: "⌊A → (□(◇A))⌋"  by simp

(* "□ₚ" is an KT modality *)
lemma C_9_p_refl: "⌊□ₚ pv A → A⌋"  by (simp add: ax_4b)
lemma "⌊□ₚ pv A → (□ₚ pv (□ₚ pv A))⌋" nitpick [user_axioms] oops (*ran out of time*)
lemma "⌊A → (□ₚ pv (◇ₚ pv A))⌋"   nitpick [user_axioms] oops (* counterexample*)

(* "□ₐ" is an KD modality *)
lemma C_10_a_serial: "⌊□ₐ av A → ◇ₐ av A⌋"  by (simp add: ax_3a)
lemma "⌊□ₐ av A → A⌋" nitpick [user_axioms] oops (* countermodel *)
lemma "⌊□ₐ av A → (□ₐ av (□ₐ av A))⌋" nitpick [user_axioms] oops (* countermodel *)
lemma "⌊A → (□ₐ av (◇ₐ av A))⌋" nitpick [user_axioms] oops (* countermodel *)

(* Relationship between "□,□ₐ,□ₚ" *)
lemma C_11: "⌊□ A → □ₚ pv A⌋" by simp
lemma C_12: "⌊□ₚ pv A → □ₐ av A⌋"  using ax_4a by auto

(* Observation II-2-1 *)
lemma ax_5b': "ob X Y ⟷ ob X (λz. X z ∧ Y z)" by (metis (no_types, lifting) ax_5b)
lemma ax_5b'': "ob X Y ⟷ ob X (λz. Y z ∧ X z)" by (metis (no_types, lifting) ax_5b)

(* Characterisation of "O" *)
lemma C_2: "⌊O ob ⟨A|B⟩ → ◇(B ∧ A)⌋" by (metis ax_5a ax_5b)
lemma C_3: "⌊(◇(A ∧ B ∧ C) ∧ O ob ⟨B|A⟩ ∧ O ob ⟨C|A⟩ ) → O ob⟨(B ∧ C)|A⟩⌋" using ax_5c by auto
lemma C_4: "⌊(□(A → B) ∧ (◇(A ∧ C)) ∧ O ob ⟨C|B⟩) → O ob ⟨C|A⟩⌋"  using ax_5e by blast
lemma C_5: "⌊□(A ↔ B) → (O ob ⟨C|A⟩ ↔ O ob ⟨C|B⟩)⌋"  by presburger
lemma C_6: "⌊□(C → (A ↔ B)) → (O ob ⟨A|C⟩ ↔ O ob ⟨B|C⟩)⌋"  by (metis ax_5b)
lemma C_7: "⌊O ob ⟨B|A⟩ → □(O ob ⟨B|A⟩)⌋"  by blast
lemma C_8: "⌊O ob ⟨B|A⟩ → O ob ⟨(A → B)|⊤⟩⌋"
 proof -
   have  "∀X Y Z. (ob X Y ∧ (∀w. X w  ⟶ Z w)) ⟶ ob Z (λw. (Z w ∧ ¬X w) ∨ Y w)"
     by  (smt ax_5d  ax_5b ax_5b'')
     thus ?thesis using ax_5b by fastforce qed

(* Relationship between "Oₐ,Oₚ,□ₐ,□ₚ" *)
lemma C_13_a: "⌊□ₐ av A → (¬Oₐ ob av A ∧ ¬Oₐ ob av (¬A))⌋"  by (metis (full_types) ax_5a ax_5b)
lemma C_13_b: "⌊□ₚ pv A → (¬Oₚ ob pv A ∧ ¬Oₚ ob pv (¬A))⌋"   by (metis (full_types) ax_5a ax_5b)
lemma C_14_a: "⌊□ₐ av (A ↔ B) → (Oₐ ob av A ↔ Oₐ ob av B)⌋"  by (metis ax_5b)
lemma C_14_b: "⌊□ₚ pv (A ↔ B) → (Oₚ ob pv A ↔ Oₚ ob pv B)⌋"  by (metis ax_5b)

(* Relationship between "O,Oₐ,Oₚ,□ₐ,□ₚ" *)
lemma C_15_a: "⌊(O ob ⟨B|A⟩ ∧ □ₐ av A ∧ ◇ₐ av B ∧ ◇ₐ av (¬B)) → Oₐ ob av B⌋"  using ax_5e by blast
lemma C_15_b: "⌊(O ob ⟨B|A⟩ ∧ □ₚ pv A ∧ ◇ₚ pv B ∧ ◇ₚ pv (¬B)) → Oₚ ob pv B⌋"  using ax_5e by blast
```

Figure 8: Tests for Extended DDL with 2 agents part 1

```
(* Soundness and consistency *)
lemma II_3_1: "(([0 ob ⟨B|A⟩]) ∧ (∃x. Z(x) ∧ A(x) ∧ B(x))) ⟶ ob(Z)(A → B)"
  proof
    assume "([0 ob ⟨B|A⟩]) ∧ (∃x. Z(x) ∧ A(x) ∧ B(x))"
     hence "ob (λz. A z ∧ Z z) (λz. A z ∧ Z z ∧ B z)" using ax_5e ax_5b  ax_5b' ax_5d by smt
     hence "ob (λz. Z z ∧ A z) (λz. Z z ∧ A z ∧ B z)" using ax_5e ax_5b  ax_5b' ax_5d by smt
     hence "ob Z (λw. (Z w ∧ ¬(Z w ∧ A w)) ∨ (Z w ∧ A w ∧ B w))" by (metis (mono_tags) ax_5d)
     from this show  L19: "ob(Z)(A → B)" by (smt ax_5b) qed

(* Some theorems and derived (proof) rules *)
lemma II_4_1: "[□(A ↔ B) → (C(A) ↔ C(B))]"  using ext by blast
lemma obs_II_4_1_a  : "[A ↔ B] ⟹ [C(A) ↔ C(B)]"  using ext by blast
lemma obs_II_4_1_b  : "[A ↔ B] ⟹ [(◇(A ∧ C) ∧ 0 ob ⟨C|B⟩) → 0 ob ⟨C|A⟩]"  using ax_5e by blast
lemma obs_II_4_1_c_1: "[◇(0 ob ⟨B|A⟩) → ◇(□(0 ob ⟨B|A⟩)))]"  by blast
lemma obs_II_4_1_c_2: "[◇(□( 0 ob ⟨B|A⟩)) → ◇(0 ob ⟨B|A⟩))]"  by auto
lemma obs_II_4_1_c_3: "[◇(0 ob ⟨B|A⟩) → □( 0 ob ⟨B|A⟩))]"  by blast
lemma obs_II_4_1_c_4: "[◇(¬( 0 ob ⟨B|A⟩)) → □(¬(0 ob ⟨B|A⟩)))]"  by blast
lemma res_II_4_1_a_1: "[¬(0 ob ⟨⊥|A⟩)]"  by (simp add: ax_5a)
lemma res_II_4_1_a_2: "[(◇ₚ pv (A ∧ B ∧ C) ∧ 0 ob ⟨B|A⟩ ∧ 0 ob ⟨C|A⟩) → 0 ob ⟨(B ∧ C)|A⟩]"  using C_3  by auto
lemma res_II_4_1_a_3: "[0 ob ⟨B|A⟩ → 0 ob ⟨B|(A ∧ B)⟩]"  by (smt ax_5a ax_5b ax_5e)
lemma res_II_4_1_a_4: "[◇ₚ pv (0 ob ⟨B|A⟩) → □ₚ pv (0 ob ⟨B|(A ∧ B)⟩)]"  by (smt ax_5a ax_5b ax_5e)
lemma res_II_4_1_a_5: "[(◇ₚ pv (A ∧ B ∧ C) ∧ 0 ob ⟨C|A⟩) → 0 ob ⟨C|(A ∧ B)⟩]"  by (smt ax_5a ax_5b ax_5e)
lemma res_II_4_1_b_1:  "[A ↔ B] ⟹ [0 ob ⟨C|A⟩ ↔ 0 ob ⟨C|B⟩]"  by (smt ax_5a ax_5b ax_5e)
lemma res_II_4_1_b_2:  "[C → (A ↔ B)] ⟹ [0 ob ⟨A|C⟩ ↔ 0 ob ⟨B|C⟩]"  by (smt ax_5b)
lemma obs_II_4_2_1: "[(0 ob ⟨B|A⟩ ∧ ◇ₐ av (A ∧ B) ∧ ◇ₐ av (A ∧ ¬B)) → (0 ob ⟨B|A⟩ ∧ ◇ₐ av (A → B) ∧ ◇ₐ av (¬(A → B)))]"  by blast
lemma obs_II_4_2_2: "[0 ob ⟨B|A⟩ → 0 ob ⟨(A → B)|⊤⟩]" by (simp add: C_8)
lemma obs_II_4_2_3: "[(0 ob ⟨(A → B)|⊤⟩ ∧ □ₐ av ⊤ ∧ ◇ₐ av (A → B) ∧ ◇ₐ av (¬(A → B))) → 0ₐ ob av (A → B)]"  by (smt ax_5e)
lemma obs_II_4_2_4: "[□ₐ av ⊤]"  by simp
lemma obs_II_4_2_5: "[(0 ob ⟨(A → B)|⊤⟩ ∧ ◇ₐ av (A → B) ∧ ◇ₐ av (¬(A → B))) → 0ₐ ob av (A → B)]"  by (smt ax_5e)
lemma obs_II_4_2_6: "[(0 ob ⟨B|A⟩ ∧ ◇ₐ av (A ∧ B) ∧ ◇ₐ av (A ∧ ¬B)) → 0ₐ ob av (A → B)]"  by (simp add: II_3_1)
lemma obs_II_4_2_6_p: "[(0 ob ⟨B|A⟩ ∧ ◇ₚ pv (A ∧ B) ∧ ◇ₚ pv (A ∧ ¬B)) → 0ₚ ob pv (A → B)]"  by (simp add: II_3_1)

lemma Oa_C: "[◇ₐ av (A ∧ B) ∧ 0ₐ ob av A ∧ 0ₐ ob av B →  0ₐ ob av (A ∧ B)]" using ax_5c by auto
lemma Op_C: "[◇ₚ pv (A ∧ B) ∧ 0ₚ ob pv A ∧ 0ₚ ob pv B →  0ₚ ob pv (A ∧ B)]" using ax_5c by auto
lemma Oa_DD: "[(0ₐ ob av A ∧ 0 ob ⟨B|A⟩ ∧ ◇ₐ av (A ∧ B)) → 0ₐ ob av (A ∧ B)]" using ax_5b ax_5c obs_II_4_2_6 by smt
declare [[smt_timeout=300]]
lemma Op_DD: "[(0ₚ ob pv A ∧ 0 ob ⟨B|A⟩ ∧ ◇ₚ pv (A ∧ B)) → 0ₚ ob pv (A ∧ B)]" using ax_5b ax_5c obs_II_4_2_6_p by smt

(*Tests for agent d*)
(* Observation II-2-1 *)
lemma dax_5b': "obd X Y ⟷ obd X (λz. X z ∧ Y z)" by (metis (no_types, lifting) axd_5b)
lemma dax_5b'': "obd X Y ⟷ obd X (λz. Y z ∧ X z)" by (simp add: axd_5b)

(* Characterisation of "0" *)
lemma dC_2: "[0 obd ⟨A|B⟩ → ◇(B ∧ A)]" by (metis axd_5a axd_5b)
lemma dC_3: "[(◇(A ∧ B ∧ C) ∧ 0 obd ⟨B|A⟩ ∧ 0 obd ⟨C|A⟩ ) → 0 obd ⟨(B ∧ C)|A⟩]" by (metis axd_5c)
lemma dC_4: "[(□(A → B) ∧ (◇(A ∧ C)) ∧ 0 obd ⟨C|B⟩) → 0 obd ⟨C|A⟩]" using axd_5e by blast
lemma dC_5: "[□(A ↔ B) → (0 obd ⟨C|A⟩ ↔ 0 obd ⟨C|B⟩)]" by presburger
lemma dC_6: "[□(C → (A ↔ B)) → (0 obd ⟨A|C⟩ ↔ 0 obd ⟨B|C⟩)]" by (metis axd_5b)
lemma dC_7: "[0 obd ⟨B|A⟩ → □(0 obd ⟨B|A⟩)]" by simp
(*lemma dC_8: "[0 obd ⟨B|A⟩ → 0 obd ⟨(A → B)|⊤⟩]"
 proof -
   have  "∀X Y Z. (obd X Y ∧ (∀w. X w  ⟶ Z w)) ⟶ obd Z (λw. (Z w ∧ ¬X w) ∨ Y w)" by (smt axd_5d  axd_5b axd_5b)
     thus ?thesis using axd_5b by fastforce qed*)
```

Figure 9: Tests for Extended DDL with 2 agents part 2



```
(* Relationship between "0ₐ, 0ₚ, □ₐ, □ₚ" *)
lemma dC_13_a: "⌊□ₐ avd A → (¬0ₐ obd avd A ∧ ¬0ₐ obd avd (¬A))⌋" using dC_2 by auto
lemma dC_13_b: "⌊□ₚ pvd A → (¬0ₚ obd pvd A ∧ ¬0ₚ obd pvd (¬A))⌋" using dC_2 by blast
lemma dC_14_a: "⌊□ₐ avd (A ↔ B) → (0ₐ obd avd A ↔ 0ₐ obd avd B)⌋" using dC_6 by blast
lemma dC_14_b: "⌊□ₚ pvd (A ↔ B) → (0ₚ obd pvd A ↔ 0ₚ obd pvd B)⌋" using dC_6 by blast

(* Relationship between "0, 0ₐ, 0ₚ, □ₐ, □ₚ" *)
lemma dC_15_a: "⌊(0 obd ⟨B|A⟩ ∧ □ₐ avd A ∧ ◇ₐ avd B ∧ ◇ₐ avd (¬B)) → 0ₐ obd avd B⌋" using dC_4 by blast
lemma dC_15_b: "⌊(0 obd ⟨B|A⟩ ∧ □ₚ pvd A ∧ ◇ₚ pvd B ∧ ◇ₚ pvd (¬B)) → 0ₚ obd pvd B⌋" using dC_4 by blast

(* Soundness and consistency *)
lemma dII_3_1: "((⌊0 obd ⟨B|A⟩⌋) ∧ (∃x. Z(x) ∧ A(x) ∧ B(x))) ⟶ obd (Z)(A → B)"
  proof
    assume "(⌊0 obd ⟨B|A⟩⌋) ∧ (∃x. Z(x) ∧ A(x) ∧ B(x))"
     hence "obd (λz. A z ∧ Z z) (λz. A z ∧ Z z ∧ B z)" using axd_5e axd_5b axd_5b axd_5d by smt
     hence "obd (λz. Z z ∧ A z) (λz. Z z ∧ A z ∧ B z)" using axd_5e axd_5b axd_5b axd_5d by smt
     hence "obd Z (λw. (Z w ∧ ¬(Z w ∧ A w)) ∨ (Z w ∧ A w ∧ B w))" by (metis (mono_tags) axd_5d)
     from this show  L19: "obd(Z)(A → B)" by (smt axd_5b) qed

(* Some theorems and derived (proof) rules *)
lemma dII_4_1: "⌊□(A ↔ B) → (C(A) ↔ C(B))⌋"  using ext by blast
lemma dobs_II_4_1_a  : "⌊A ↔ B⌋ ⟹ ⌊C(A) ↔ C(B)⌋"  using ext by blast
lemma dobs_II_4_1_b  : "⌊A ↔ B⌋ ⟹ ⌊(◇(A ∧ C) ∧ 0 obd ⟨C|B⟩) → 0 obd ⟨C|A⟩⌋"  by presburger
lemma dobs_II_4_1_c_1: "⌊◇(0 obd ⟨B|A⟩) → ◇(□(0 obd ⟨B|A⟩))⌋"  by blast
lemma dobs_II_4_1_c_2: "⌊◇(□(0 obd ⟨B|A⟩)) → ◇(0 obd ⟨B|A⟩)⌋"  by auto
lemma dobs_II_4_1_c_3: "⌊◇(0 obd ⟨B|A⟩) → □(0 obd ⟨B|A⟩)⌋" by simp
lemma dobs_II_4_1_c_4: "⌊◇(¬(0 obd ⟨B|A⟩)) → □(¬(0 obd ⟨B|A⟩))⌋"  by blast
lemma dres_II_4_1_a_1: "⌊¬(0 obd ⟨⊥|A⟩)⌋"  by (simp add: axd_5a)
lemma dres_II_4_1_a_2: "⌊(◇ₚ pvd (A ∧ B ∧ C) ∧ 0 obd ⟨B|A⟩ ∧ 0 obd ⟨C|A⟩) → 0 obd ⟨(B ∧ C)|A⟩⌋"  by (metis dC_3)
lemma dres_II_4_1_a_3: "⌊0 obd ⟨B|A⟩ → 0 obd ⟨B|(A ∧ B)⟩⌋"  by (smt (verit) dC_2 dC_4)
lemma dres_II_4_1_a_4: "⌊◇ₚ pvd (0 obd ⟨B|A⟩) → □ₚ pvd (0 obd ⟨B|(A ∧ B)⟩)⌋" by (simp add: dres_II_4_1_a_3)
lemma dres_II_4_1_a_5: "⌊(◇ₚ pvd (A ∧ B ∧ C) ∧ 0 obd ⟨C|A⟩) → 0 obd ⟨C|(A ∧ B)⟩⌋" by (smt (verit) axd_5e)
lemma dres_II_4_1_b_1:  "⌊A ↔ B⌋ ⟹ ⌊0 obd ⟨C|A⟩ ↔ 0 obd ⟨C|B⟩⌋" solve_direct oops
lemma dres_II_4_1_b_2:  "⌊C → (A ↔ B)⌋ ⟹ ⌊0 obd ⟨A|C⟩ ↔ 0 obd ⟨B|C⟩⌋" using dC_6 by auto
lemma dobs_II_4_2_1: "⌊(0 obd ⟨B|A⟩ ∧ ◇ₐ avd (A ∧ B) ∧ ◇ₐ avd (A ∧ ¬B)) → (0 obd ⟨B|A⟩ ∧ ◇ₐ avd (A → B) ∧ ◇ₐ avd (¬(A → B)))⌋" by auto
(*lemma dobs_II_4_2_2: "⌊0 obd ⟨B|A⟩ → 0 obd ⟨(A → B)|⊤⟩⌋" C8*)
lemma dobs_II_4_2_3: "⌊(0 obd ⟨(A → B)|⊤⟩ ∧ □ₐ avd ⊤ ∧ ◇ₐ avd (A → B) ∧ ◇ₐ avd (¬(A → B))) → 0ₐ obd avd (A → B)⌋" solve_direct oops
lemma dobs_II_4_2_4: "⌊□ₐ avd ⊤⌋" by simp
lemma dobs_II_4_2_5: "⌊(0 obd ⟨(A → B)|⊤⟩ ∧ ◇ₐ avd (A → B) ∧ ◇ₐ avd (¬(A → B))) → 0ₐ obd avd (A → B)⌋" by (smt (verit, best) dC_4)
lemma dobs_II_4_2_6: "⌊(0 obd ⟨B|A⟩ ∧ ◇ₐ avd (A ∧ B) ∧ ◇ₐ avd (A ∧ ¬B)) → 0ₐ obd avd (A → B)⌋" by (simp add: dII_3_1)
lemma dobs_II_4_2_6_p: "⌊(0 obd ⟨B|A⟩ ∧ ◇ₚ pvd (A ∧ B) ∧ ◇ₚ pvd (A ∧ ¬B)) → 0ₚ obd pvd (A → B)⌋" by (simp add: dII_3_1)

lemma dOa_C: "⌊◇ₐ avd (A ∧ B) ∧ 0ₐ obd avd A ∧ 0ₐ obd avd B →  0ₐ obd avd (A ∧ B)⌋" by (metis (full_types) dC_3)
lemma dOp_C: "⌊◇ₚ pvd (A ∧ B) ∧ 0ₚ obd pvd A ∧ 0ₚ obd pvd B →  0ₚ obd pvd (A ∧ B)⌋" by (metis (full_types) dC_3)
lemma dOa_DD: "⌊(0ₐ obd avd A ∧ 0 obd ⟨B|A⟩ ∧ ◇ₐ avd (A ∧ B)) → 0ₐ obd avd (A ∧ B)⌋" using axd_5b axd_5c dobs_II_4_2_6 by smt
declare [[smt_timeout=300]]
lemma dOp_DD: "⌊(0ₚ obd pvd A ∧ 0 obd ⟨B|A⟩ ∧ ◇ₚ pvd (A ∧ B)) → 0ₚ obd pvd (A ∧ B)⌋" using axd_5b axd_5c dobs_II_4_2_6_p by smt

end
```

Figure 10: Tests for Extended DDL with 2 agents part 3

7 Test File for Extended DDL 2

Figure 11 shows the tests (created by Benzmüller et al. (2022)) for Extended DDL 2.

```
theory CJ_DDL_agents_mod_tests imports DDL_agents_mod          (* Christoph Benzmüller, Ali Farjami, Xavier Parent, 2020  *)
begin (* Some Tests on the Meta-Theory of DDL*)
lemma True nitpick [satisfy,user_axioms,expect=genuine] oops  (* Consistency confirmed by Nitpick *)

(* Observation II-2-1 *)
lemma dax_5b': "ob_g a X Y ⟷ ob_g a X (λz. X z ∧ Y z)" by (smt (verit, ccfv_SIG) axg_5b)
lemma dax_5b'': "ob_g a X Y ⟷ ob_g a X (λz. Y z ∧ X z)" by (simp add: axg_5b)

(* Characterisation of "O" *)
lemma dC_2: "⌊O a ⟨A|B⟩ → ◇(B ∧ A)⌋" by (metis axg_5a axg_5b)
lemma dC_3: "⌊(◇(A ∧ B ∧ C) ∧ O a ⟨B|A⟩ ∧ O a ⟨C|A⟩ ) → O a ⟨(B ∧ C)|A⟩⌋" by (metis axg_5c)
lemma dC_4: "⌊(□(A → B) ∧ (◇(A ∧ C)) ∧ O a ⟨C|B⟩) → O a ⟨C|A⟩⌋" using axg_5e by blast
lemma dC_5: "⌊□(A ↔ B) → (O a ⟨C|A⟩ ↔ O a ⟨C|B⟩)⌋"  by presburger
lemma dC_6: "⌊□(C → (A ↔ B)) → (O a ⟨A|C⟩ ↔ O a ⟨B|C⟩)⌋" by (smt (verit, del_insts) axg_5b)
lemma dC_7: "⌊O a ⟨B|A⟩ → □(O a ⟨B|A⟩)⌋"  by simp
(*lemma dC_8: "⌊O a ⟨B|A⟩ → O a ⟨(A → B)|⊤⟩⌋"
 proof -
  have  "∀X Y Z. (ob_g a X Y ∧ (∀w. X w  ⟷ Z w)) ⟷ ob_g a Z (λw. (Z w ∧ ¬X w) ∨ Y w)" by (smt axg_5d  axg_5b axg_5b)
  thus ?thesis using axg_5b by fastforce qed*)

(* Relationship between "O_a, O_p, □_a, □_p" *)
lemma dC_13_a: "⌊□ₐ a A → (¬Oₐ a A ∧ ¬Oₐ a (¬A))⌋" using dC_2 by auto
lemma dC_13_b: "⌊□ₚ a A → (¬Oₚ a A ∧ ¬Oₚ a (¬A))⌋" using dC_2 by blast
lemma dC_14_a: "⌊□ₐ a (A ↔ B) → (Oₐ a A ↔ Oₐ a B)⌋" using dC_6 by blast
lemma dC_14_b: "⌊□ₚ a (A ↔ B) → (Oₚ a A ↔ Oₚ a B)⌋" using dC_6 by blast

(* Relationship between "O, O_a, O_p, □_a, □_p" *)
lemma dC_15_a: "⌊(O a ⟨B|A⟩ ∧ □ₐ a A ∧ ◇ₐ a B ∧ ◇ₐ a (¬B)) → Oₐ a B⌋" using dC_4 by blast
lemma dC_15_b: "⌊(O a ⟨B|A⟩ ∧ □ₚ a A ∧ ◇ₚ a B ∧ ◇ₚ a (¬B)) → Oₚ a B⌋" using dC_4 by blast

(* Soundness and consistency *)
lemma dII_3_1: "((⌊O a ⟨B|A⟩⌋) ∧ (∃x. Z(x) ∧ A(x) ∧ B(x))) ⟶ ob_g a (Z)(A → B)"
  proof
    assume "(⌊O a ⟨B|A⟩⌋) ∧ (∃x. Z(x) ∧ A(x) ∧ B(x))"
     hence "ob_g a (λz. A z ∧ Z z) (λz. A z ∧ Z z ∧ B z)" using axg_5e  axg_5b axg_5b axg_5d by smt
     hence "ob_g a (λz. Z z ∧ A z) (λz. Z z ∧ A z ∧ B z)" using axg_5e  axg_5b axg_5b axg_5d by smt
     hence "ob_g a Z (λw. (Z w ∧ ¬(Z w ∧ A w)) ∨ (Z w ∧ A w ∧ B w))" by (metis (mono_tags) axg_5d)
     from this show  L19: "ob_g a(Z)(A → B)" by (smt axg_5b) qed

(* Some theorems and derived (proof) rules *)
lemma dII_4_1: "⌊□(A ↔ B) → (C(A) ↔ C(B))⌋"  using ext by blast
lemma dobs_II_4_1_a  : "⌊A ↔ B⌋ ⟹ ⌊C(A) ↔ C(B)⌋"  using ext by blast
lemma dobs_II_4_1_b  : "⌊A ↔ B⌋ ⟹ ⌊(◇(A ∧ C) ∧ O a ⟨C|B⟩) → O a ⟨C|A⟩⌋"  by presburger
lemma dobs_II_4_1_c_1: "⌊◇(O a ⟨B|A⟩) → ◇(□(O a ⟨B|A⟩))⌋"  by blast
lemma dobs_II_4_1_c_2: "⌊◇(□(O a ⟨B|A⟩)) → ◇(O a ⟨B|A⟩)⌋"  by auto
lemma dobs_II_4_1_c_3: "⌊◇(O a ⟨B|A⟩) → □(O a ⟨B|A⟩)⌋" by simp
lemma dobs_II_4_1_c_4: "⌊◇(¬(O a ⟨B|A⟩)) → □(¬(O a ⟨B|A⟩))⌋"  by blast
lemma dres_II_4_1_a_1: "⌊¬(O a ⟨⊥|A⟩)⌋"  by (simp add: axg_5a)
lemma dres_II_4_1_a_2: "⌊(◇ₚ a (A ∧ B ∧ C) ∧ O a ⟨B|A⟩ ∧ O a ⟨C|A⟩) → O a ⟨(B ∧ C)|A⟩⌋"  by (metis dC_3)
lemma dres_II_4_1_a_3: "⌊O a ⟨B|A⟩ → O a ⟨B|(A ∧ B)⟩⌋"  by (smt (verit) dC_2 dC_4)
lemma dres_II_4_1_a_4: "⌊◇ₚ a (O a ⟨B|A⟩) → □ₚ a (O a ⟨B|(A ∧ B)⟩)⌋" by (simp add: dres_II_4_1_a_3)
lemma dres_II_4_1_a_5: "⌊(◇ₚ a (A ∧ B ∧ C) ∧ O a ⟨C|A⟩) → O a ⟨C|(A ∧ B)⟩⌋" by (smt (verit) axg_5e)
lemma dres_II_4_1_b_1:  "⌊A ↔ B⌋ ⟹ ⌊O a ⟨C|A⟩ ↔ O a ⟨C|B⟩⌋" solve_direct oops
lemma dres_II_4_1_b_2:  "⌊C → (A ↔ B)⌋ ⟹ ⌊O a ⟨A|C⟩ ↔ O a ⟨B|C⟩⌋" using dC_6 by auto
lemma dobs_II_4_2_1: "⌊(O a ⟨B|A⟩ ∧ ◇ₐ a (A ∧ B) ∧ ◇ₐ a (A ∧ ¬B)) → (O a ⟨B|A⟩ ∧ ◇ₐ a (A → B) ∧ ◇ₐ a (¬(A → B)))⌋" by auto
(*lemma dobs_II_4_2_2: "⌊O obd ⟨B|A⟩ → O obd ⟨(A → B)|⊤⟩⌋" C8*)
lemma dobs_II_4_2_3: "⌊(O a ⟨(A → B)|⊤⟩ ∧ □ₐ a ⊤ ∧ ◇ₐ a (A → B) ∧ ◇ₐ a (¬(A → B))) → Oₐ a (A → B)⌋" solve_direct oops
lemma dobs_II_4_2_4: "⌊□ₐ a ⊤⌋" by simp
lemma dobs_II_4_2_5: "⌊(O a ⟨(A → B)|⊤⟩ ∧ ◇ₐ a (A → B) ∧ ◇ₐ a (¬(A → B))) → Oₐ a (A → B)⌋" by (smt (verit, best) dC_4)
lemma dobs_II_4_2_6: "⌊(O a ⟨B|A⟩ ∧ ◇ₐ a (A ∧ B) ∧ ◇ₐ a (A ∧ ¬B)) → Oₐ a (A → B)⌋" by (simp add: dII_3_1)
lemma dobs_II_4_2_6_p: "⌊(O a ⟨B|A⟩ ∧ ◇ₚ a (A ∧ B) ∧ ◇ₚ a (A ∧ ¬B)) → Oₚ a (A → B)⌋" by (simp add: dII_3_1)

lemma dOa_C: "⌊◇ₐ a (A ∧ B) ∧ Oₐ a A ∧ Oₐ a B →  Oₐ a (A ∧ B)⌋" by (metis (full_types) dC_3)
lemma dOp_C: "⌊◇ₚ a (A ∧ B) ∧ Oₚ a A ∧ Oₚ a B →  Oₚ a (A ∧ B)⌋" by (metis (full_types) dC_3)
lemma dOa_DD: "⌊(Oₐ a A ∧ O a ⟨B|A⟩ ∧ ◇ₐ a (A ∧ B)) → Oₐ a (A ∧ B)⌋" using axg_5b axg_5c dobs_II_4_2_6 by smt
declare [[smt_timeout=300]]
lemma dOp_DD: "⌊(Oₚ a A ∧ O a ⟨B|A⟩ ∧ ◇ₚ a (A ∧ B)) → Oₚ a (A ∧ B)⌋" using axg_5b axg_5c dobs_II_4_2_6_p by smt

end
```

Figure 11: Tests for Extended DDL 2

8 Prohibited (Chapter 5.1) in DDL

Figures 12 and 13 shows the example on prohibited AI systems from Chapter 5.1 in DDL.

```
theory prohibited
  imports
   DDL
   types
begin

consts (*Predicates/relations*)
  subliminal_technique::"aiSys⇒σ" (*deploys a subliminal technique beyond a persons consciousness*)
  has_consequence::"aiSys⇒consequence⇒σ" (*system has or may have a consequence*)
  has_purpose::"aiSys⇒purpose⇒σ" (*system has a purpose*)
  exploits_vulnerabilities_group::"aiSys⇒quality_person⇒σ"
  exploits_vulnerable_group::"aiSys⇒σ" (*exploits any of the vulnerabilities of a specific group due to a certain quality*)
  employed_by_pauthorities::"aiSys⇒σ" (*employed by public authorities or on their behalf*)
  prohibited :: "aiSys ⇒ σ" (*placing on market, putting into service, or use*)

abbreviation "H1 ≡ ⌊∀x::aiSys. ((has_consequence x harm_physical) ∨
(has_consequence x harm_psychological)) ↔ has_consequence x harm⌋"
abbreviation "H2 ≡ ⌊∀x::aiSys. ((exploits_vulnerabilities_group x age) ∨ (exploits_vulnerabilities_group x physcial_disability)
∨ (exploits_vulnerabilities_group x mental_disability)) ↔ exploits_vulnerable_group x⌋"
abbreviation "A1 ≡ ⌊∀x::aiSys. subliminal_technique x  ∧  has_consequence x harm ∧
has_purpose x distort_behavior → ○<prohibited x>⌋"
abbreviation "B1 ≡ ⌊∀x::aiSys. exploits_vulnerable_group x ∧ has_purpose x distort_behavior ∧
has_consequence x harm → ○<prohibited x>⌋"
abbreviation "C1 ≡ ⌊∀x::aiSys. employed_by_pauthorities x ∧ has_purpose x eval_trustworthiness_over_time ∧
(has_consequence x detri_treatment_unrelated_context ∨ has_consequence x detri_treatment_unjustified_disprop)
→ ○<prohibited x>⌋"

consts
  x::aiSys

abbreviation "F1 w ≡ (subliminal_technique x) w"
abbreviation "F2 w ≡ (has_consequence x harm_physical) w"
abbreviation "F3 w ≡ (has_purpose x distort_behavior) w"


theorem Result1a: "H1 ∧ H2 ∧ A1 ∧ B1 ∧ C1 ⟶ ⌊(F1 ∧ F2 ∧ F3)  → (○<prohibited x>)⌋"
  by metis

theorem Result1b: "H1 ∧ H2 ∧ A1 ∧ B1 ∧ C1 ⟶ ⌊F1 ∧ F2  → (○<prohibited x>)⌋"
  nitpick [user_axioms] oops (*counterexample found*)

lemma True nitpick [satisfy, user_axioms, show_all] oops

```

Figure 12: Prohibited example Chapter 5.1. in DDL part 1

```
consts
  real_time_bioid::"aiSys⇒σ" (*system is a real time bio identification system*)
  use_public_spaces::"aiSys⇒σ" (*system is planned to be used in public spaces*)
  use_law_enforcement::"aiSys⇒σ" (*system is used for law enforcement*)
  strictly_necessary_for::"aiSys⇒purpose⇒σ" (*use of system is strictly necessary for a purpose*)
  consider_consequence::"aiSys⇒consequence⇒σ" (*consider specific consequence of the use of a system*)
  consider_consequence_no_use::"aiSys⇒consequence⇒σ" (*consider specific consequence of not using the system*)
  consider_context::"aiSys⇒σ" (*consider the context in which the system is used*)
  complies_with_bioid_rules::"aiSys⇒σ" (*does system comply or not?*)

abbreviation "D1 ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
(((has_purpose x targeted_search) ∧ (¬(strictly_necessary_for x targeted_search))) ∨
((has_purpose x detection) ∧ (¬(strictly_necessary_for x detection))) ∨ ((has_purpose x prevention) ∧
(¬(strictly_necessary_for x prevention)))) → ○<prohibited x>⌋"
(*implicit: not prohibited*)
abbreviation "D1b ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
(((has_purpose x targeted_search) ∧ (strictly_necessary_for x targeted_search)) ∨
((has_purpose x detection) ∧ (strictly_necessary_for x detection)) ∨ ((has_purpose x prevention) ∧
(strictly_necessary_for x prevention))) → (¬(○<prohibited x>) ∧ (○<high_risk x>))⌋"
abbreviation "A2a ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
  (○<high_risk x>) → ○<consider_consequence_no_use x harm_psychological> ∧
  ○<consider_consequence_no_use x harm_physical>⌋"
abbreviation "A2b ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
  (○<high_risk x>) → ○<consider_consequence x affect_personal_rights> ∧ ○<consider_consequence x affect_personal_freedom>⌋"
abbreviation "A2c ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
  (○<high_risk x>) → ○<complies_with_bioid_rules x>⌋"

consts
  z::aiSys

abbreviation "F4 w ≡ (real_time_bioid z) w"
abbreviation "F5 w ≡ (use_public_spaces z) w"
abbreviation "F6 w ≡ (use_law_enforcement z) w"
abbreviation "F7 w ≡ (has_purpose z targeted_search) w"
abbreviation "F8 w ≡ ¬(strictly_necessary_for z targeted_search) w"
abbreviation "F9 w ≡ (strictly_necessary_for z targeted_search) w"

theorem Result2a: "D1 ∧ D1b ∧ A2a ∧ A2b ∧ A2c ⟶ ⌊F4 ∧ F5 ∧ F6 ∧ F7 ∧ F8 → (○<prohibited z>)⌋"
  by meson

theorem Result2b: "D1 ∧ D1b ∧ A2a ∧ A2b ∧ A2c ⟶ ⌊F4 ∧ F5 ∧ F6 ∧ F7 ∧ F9 → (○<prohibited z>)⌋"
  nitpick [user_axioms, card i = 2] oops (*found counterexample*)

lemma True nitpick [satisfy, user_axioms, show_all] oops

end
```

Figure 13: Prohibited example Chapter 5.1 in DDL part 2

9 Prohibited (Chapter 5.1) in Extended DDL

Figures 14 and 15 show the example on prohibited AI systems from Chapter 5.1 in Extended DDL.

```
theory prohibited
  imports
   DDL_agents_clean
   types
begin

consts (*Predicates/relations*)
  subliminal_technique::"aiSys⇒σ" (*deploys a subliminal technique beyond a persons consciousness*)
  has_consequence::"aiSys⇒consequence⇒σ" (*system has or may have a consequence*)
  has_purpose::"aiSys⇒purpose⇒σ" (*system has a purpose*)
  exploits_vulnerabilities_group::"aiSys⇒quality_person⇒σ"
  exploits_vulnerable_group::"aiSys⇒σ" (*exploits any of the vulnerabilities of a specific group due to a certain quality*)
  employed_by_pauthorities::"aiSys⇒σ" (*employed by public authorities or on their behalf*)
  prohibited :: "aiSys ⇒ σ" (*placing on market, putting into service, or use*)

abbreviation "H1 ≡ ⌊∀x::aiSys. ((has_consequence x harm_physical) ∨
(has_consequence x harm_psychological)) ↔ has_consequence x harm⌋"
abbreviation "H2 ≡ ⌊∀x::aiSys. ((exploits_vulnerabilities_group x age) ∨ (exploits_vulnerabilities_group x physcial_disability)
∨ (exploits_vulnerabilities_group x mental_disability)) ↔ exploits_vulnerable_group x⌋"
abbreviation "A1 ≡ ⌊∀x::aiSys. subliminal_technique x  ∧  has_consequence x harm ∧
has_purpose x distort_behavior → ○<prohibited x>⌋"
abbreviation "B1 ≡ ⌊∀x::aiSys. exploits_vulnerable_group x ∧ has_purpose x distort_behavior ∧
has_consequence x harm → ○<prohibited x>⌋"
abbreviation "C1 ≡ ⌊∀x::aiSys. employed_by_pauthorities x ∧ has_purpose x eval_trustworthiness_over_time ∧
(has_consequence x detri_treatment_unrelated_context ∨ has_consequence x detri_treatment_unjustified_disprop)
→ ○<prohibited x>⌋"

consts
  x::aiSys

abbreviation "F1 w ≡ (subliminal_technique x) w"
abbreviation "F2 w ≡ (has_consequence x harm_physical) w"
abbreviation "F3 w ≡ (has_purpose x distort_behavior) w"


theorem Result1a: "H1 ∧ H2 ∧ A1 ∧ B1 ∧ C1 ⟶ ⌊(F1 ∧ F2 ∧ F3)  → (○<prohibited x>)⌋"
  by metis

theorem Result1b: "H1 ∧ H2 ∧ A1 ∧ B1 ∧ C1 ⟶ ⌊F1 ∧ F2  → (○<prohibited x>)⌋"
  nitpick [user_axioms] oops (*counterexample found*)

lemma True nitpick [satisfy, user_axioms, show_all] oops
```

Figure 14: Prohibited example Chapter 5.1 in Extended DDL part 1

```
consts
  real_time_bioid::"aiSys⇒σ" (*system is a real time bio identification system*)
  use_public_spaces::"aiSys⇒σ" (*system is planned to be used in public spaces*)
  use_law_enforcement::"aiSys⇒σ" (*system is used for law enforcement*)
  strictly_necessary_for::"aiSys⇒purpose⇒σ" (*use of system is strictly necessary for a purpose*)
  consider_consequence::"aiSys⇒consequence⇒σ" (*consider specific consequence of the use of a system*)
  consider_consequence_no_use::"aiSys⇒consequence⇒σ" (*consider specific consequence of not using the system*)
  consider_context::"aiSys⇒σ" (*consider the context in which the system is used*)
  complies_with_bioid_rules::"aiSys⇒σ" (*does system comply or not?*)

abbreviation "D1 ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
(((has_purpose x targeted_search) ∧ (¬(strictly_necessary_for x targeted_search))) ∨
((has_purpose x detection) ∧ (¬(strictly_necessary_for x detection))) ∨ ((has_purpose x prevention) ∧
(¬(strictly_necessary_for x prevention)))) → ○<prohibited x>⌋"
(*implicit: not prohibited*)
abbreviation "D1b ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
(((has_purpose x targeted_search) ∧ (strictly_necessary_for x targeted_search)) ∨
((has_purpose x detection) ∧ (strictly_necessary_for x detection)) ∨ ((has_purpose x prevention) ∧
(strictly_necessary_for x prevention))) → (¬(○<prohibited x>) ∧ (○<high_risk x>))⌋"
abbreviation "A2a ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
  (○<high_risk x>) → ○<consider_consequence_no_use x harm_psychological> ∧
  ○<consider_consequence_no_use x harm_physical>⌋"
abbreviation "A2b ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
  (○<high_risk x>) → ○<consider_consequence x affect_personal_rights> ∧ ○<consider_consequence x affect_personal_freedom>⌋"
abbreviation "A2c ≡ ⌊∀x::aiSys. real_time_bioid x ∧ use_public_spaces x ∧ use_law_enforcement x ∧
  (○<high_risk x>) → ○<complies_with_bioid_rules x>⌋"

consts
  z::aiSys

abbreviation "F4 w ≡ (real_time_bioid z) w"
abbreviation "F5 w ≡ (use_public_spaces z) w"
abbreviation "F6 w ≡ (use_law_enforcement z) w"
abbreviation "F7 w ≡ (has_purpose z targeted_search) w"
abbreviation "F8 w ≡ ¬(strictly_necessary_for z targeted_search) w"
abbreviation "F9 w ≡ (strictly_necessary_for z targeted_search) w"

theorem Result2a: "D1 ∧ D1b ∧ A2a ∧ A2b ∧ A2c ⟶ ⌊F4 ∧ F5 ∧ F6 ∧ F7 ∧ F8 → (○<prohibited z>)⌋"
  by meson

theorem Result2b: "D1 ∧ D1b ∧ A2a ∧ A2b ∧ A2c ⟶ ⌊F4 ∧ F5 ∧ F6 ∧ F7 ∧ F9 → (○<prohibited z>)⌋"
  nitpick [user_axioms, card i = 2] oops (*found counterexample*)

lemma True nitpick [satisfy, user_axioms, show_all] oops

end
```

Figure 15: Prohibited example Chapter 5.1 in Extended DDL part 2

10 CTD from Article 26 in Extended DDL and Extended DDL 2

Figures 16 and 17 show an example CTD formalized in Extended DDL and Extended DDL 2. While the representation in Extended DDL is successful, the one in Extended DDL 2 leads to the same problems already discussed in Chapter 5.3.2.2.1.

```
theory "high-risk-3-3-26-DDL"
  imports
  DDL_agents_clean
begin

(*CTD structure*)
consts
  l::aiSys
  system_in_conformity::"aiSys⇒σ"
  not_on_market::"aiSys⇒σ"


axiomatization where
A0: "⌊high_risk l⌋ι" and
A1a: "⌊∀x. (high_risk x) → ⊙b<stit b (system_in_conformity x)>⌋" and
A8a: "⌊∀x. ¬ (system_in_conformity x) ∧ (high_risk x) → ⊙b<(stit b (not_on_market x))>⌋" and
(*implicit: If the system is in conformity, the importer is obligated to not see to it that the system is not placed
on the market*)
AXa: "⌊∀x. ⊙b<(stit b (system_in_conformity x)) → (¬(stit b (not_on_market x)))>⌋" and
Situationb: "⌊¬ (system_in_conformity l)⌋ι"

(***Some Experiments***)
lemma True nitpick [satisfy, user_axioms, show_all] oops (*Consistency-check: Nitpick finds a model.*)

lemma "⌊⊙b<(stit b (not_on_market l))>⌋ι" using A0 A8a Situationb by auto
lemma "⌊⊙b<¬ (stit b (not_on_market l))>⌋ι" nitpick [user_axioms] oops (*counterexample found*)

end
```

Figure 16: CTD from Chapter 26 in Extended DDL

```
theory "high-risk-3-3-26-DDL2"
  imports
    DDL_agents_mod
begin

consts
  system_in_conformity::"aiSys⇒σ"
  not_on_market::"aiSys⇒σ"
  l::aiSys
  b::ag
  importer_of::"ag⇒aiSys⇒σ"

axiomatization where
A1a: "⌊∀x i. (high_risk x ∧ importer i ∧ importer_of i x) → ○i<stit i (system_in_conformity x)>⌋" and
A8a: "⌊∀x i. (¬ (system_in_conformity x) ∧ (high_risk x ∧ importer i ∧ importer_of i x)) → ○i<(stit i (not_on_market x))>⌋" and
(*implicit: If the system is in conformity, the importer is obligated to not see to it that the system is not placed
on the market*)
AXa: "⌊∀x i. (high_risk x ∧ importer i ∧ importer_of i x) → (○i<stit i (system_in_conformity x) → (¬ (stit i (not_on_market x)))>)⌋" and

F0: "⌊high_risk l⌋ι" and F1: "⌊importer b⌋ι" and F2: "⌊importer_of b l⌋ι" and
Situationb: "⌊¬ (system_in_conformity l)⌋ι"

(***Some Experiments***)
lemma True nitpick [satisfy, user_axioms, card i=1] oops (*Consistency-check: Nitpick ran out of time*)

lemma "⌊○b<(stit b (not_on_market l))>⌋ι" using A8a F0 F1 F2 Situationb by blast
lemma "⌊○b<¬ (stit b (not_on_market l))>⌋ι" nitpick [user_axioms] oops (*Nitpick ran out of time*)

end
```

Figure 17: CTD from Chapter 26 in Extended DDL 2

11 Git Repository with all Isabelle/HOL Files

All Isabelle/HOL files created in this thesis can be found in this Git repository.

References

Anderson, A. R. (1962). Logic, norms, and roles. *Ratio (Misc.)*, *4*(1), 36–49.

Athan, T., Governatori, G., Palmirani, M., Paschke, A., & Wyner, A. (2015). LegalRuleML: Design Principles and Foundations [Series Title: Lecture Notes in Computer Science]. In W. Faber & A. Paschke (Eds.), *Reasoning Web. Web Logic Rules* (pp. 151–188, Vol. 9203). Springer International Publishing. https://doi.org/10.1007/978-3-319-21768-0_6

Belnap, N. (1991). Backwards and Forwards in the Modal Logic of Agency. *Philosophy and Phenomenological Research*, *51*(4), 777. https://doi.org/10.2307/2108182

Belnap, N., & Perloff, M. (1988). Seeing to it that: A canonical form for agentives. *Theoria*, *54*(3), 175–199. https://doi.org/10.1111/j.1755-2567.1988.tb00717.x

Belnap, N., Perloff, M., & Xu, M. (2001). *Facing the Future: Agents and Choices In Our Indertiministic World*. Oxford University Press.

Benzmüller, C. (2019). Universal (meta-)logical reasoning: Recent successes. *Science of Computer Programming*, *172*, 48–62. https://doi.org/10.1016/j.scico.2018.10.008

Benzmüller, C., & Andrews, P. (2022). Church's Type Theory. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford encyclopedia of philosophy* (Winter 2022). Metaphysics Research Lab, Stanford University.

Benzmüller, C., Brown, C. E., & Kohlhase, M. (2004). Higher-order semantics and extensionality. *The Journal of Symbolic Logic*, *69*(4), 1027–1088. Retrieved May 8, 2024, from http://www.jstor.org/stable/30041776

Benzmüller, C., Claus, M., & Sultana, N. (2015). Systematic Verification of the Modal Logic Cube in Isabelle/HOL. *Electronic Proceedings in Theoretical Computer Science*, *186*, 27–41. https://doi.org/10.4204/EPTCS.186.5

Benzmüller, C., Farjami, A., & Parent, X. (2018, October). *Åqvist's Dyadic Deontic Logic E in HOL* (EasyChair Preprints) (Series: EasyChair Preprints). EasyChair. https://doi.org/10.29007/t29j

Benzmüller, C., Farjami, A., & Parent, X. (2022). Dyadic Deontic Logic in HOL: Faithful Embedding and Meta-Theoretical Experiments [Series Title: Logic, Argumentation & Reasoning]. In S. Rahman, M. Armgardt, & H. C. N. Kvernenes (Eds.), *New Devel-*

opments in Legal Reasoning and Logic (pp. 353–377, Vol. 23). Springer International Publishing. https://doi.org/10.1007/978-3-030-70084-3_14

Benzmüller, C., & Fuenmayor, D. (2021). Value-Oriented Legal Argumentation in Isabelle/HOL. *LIPIcs, Volume 193, ITP 2021*, *193*, 7:1–7:20. https://doi.org/10.4230/LIPICS.ITP.2021.7

Benzmüller, C., Fuenmayor, D., Parent, X., iamsreiche, & lex-lex. (2023). Logikey. https://github.com/cbenzmueller/LogiKEy

Benzmüller, C., & Parent, X. (2018). First experiments with a flexible infrastructure for normative reasoning.

Benzmüller, C., Parent, X., & van der Torre, L. (2020, May 24). Designing normative theories for ethical and legal reasoning: LogiKEy framework, methodology, and tool support. Retrieved June 26, 2023, from http://arxiv.org/abs/1903.10187

Blackburn, P., de Rijke, M., & Venema, Y. (2001). Modal logic. *Studia Logica*, *76*(1), 142–148.

Blanchette, J. (2023). A User's Guide to Nitpick for Isabelle/HOL.

Blanchette, J. C., & Paulson, L. C. (2013). A User's Guide to Sledgehammer for Isabelle/HOL.

Boley, H., Paschke, A., & Shafiq, M. (2010). Ruleml 1.0: The overarching specification of web rules. https://doi.org/10.1007/978-3-642-16289-3_15

Brown, M. A. (1988). On the logic of ability. *Journal of Philosophical Logic*, *17*(1), 1–26. https://doi.org/10.1007/BF00249673

Carmo, J., & Jones, A. J. I. (2002a). Deontic Logic and Contrary-to-Duties. In D. M. Gabbay & F. Guenthner (Eds.), *Handbook of Philosophical Logic: Volume 8* (pp. 265–343). Springer Netherlands. https://doi.org/10.1007/978-94-010-0387-2_4

Carmo, J., & Jones, A. J. I. (2002b). Deontic Logic and Contrary-to-Duties. In D. M. Gabbay & F. Guenthner (Eds.), *Handbook of Philosophical Logic* (pp. 265–343). Springer Netherlands. https://doi.org/10.1007/978-94-010-0387-2_4

Carmo, J., & Jones, A. J. I. (2013). Completeness and decidability results for a logic of contrary-to-duty conditionals. *Journal of Logic and Computation*, *23*(3), 585–626. https://doi.org/10.1093/logcom/exs009

Carmo, J., & Jones, A. J. I. (2022). Carmo and jones' logic for contrary-to-duty obligations revised. *Journal of Logic and Computation*, *32*(7), 1352–1364. https://doi.org/10.1093/logcom/exac026

Chellas, B. F. (1992). Time and modality in the logic of agency. *Studia Logica*, *51*(3-4), 485–517. https://doi.org/10.1007/bf01028972

Chisholm, R. M. (1963). Contrary-to-duty imperatives and deontic logic. *Analysis*, *24*(2), 33–36. https://doi.org/10.1093/analys/24.2.33

Church, A. (1940). A formulation of the simple theory of types. *The Journal of Symbolic Logic*, *5*(2). https://doi.org/10.2307/2266170

Cintula, P., Fermüller, C. G., & Noguera, C. (2023). Fuzzy Logic. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford encyclopedia of philosophy* (Summer 2023). Metaphysics Research Lab, Stanford University.

Cornell Law School. (n.d.). Akoma ntoso. Retrieved June 6, 2024, from https://www.law.cornell.edu/wiki/lexcraft/akoma_ntoso

Danielsson, S. (1969). *Preference and obligation*. Filosofiska foereningen.

Dov Gabbay, John Horty, Xavier Parent, Ron van der Meyden, & Leendert van der Torre. (2013). *Handbook of Deontic Logic and Normative Systems*. http://www.collegepublications.co.uk/downloads/handbooks00001.pdf

European Parliament. (2023). *Eu ai act: First regulation on artificial intelligence*. https://www.europarl.europa.eu/news/en/headlines/society/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence#:~:text=It%20says%20that%20AI%20systems,world's%20first%20rules%20on%20AI.

Fitch, F. B. (1963). A logical analysis of some value concepts. *The Journal of Symbolic Logic*, *28*(2), 135–142. Retrieved March 11, 2024, from http://www.jstor.org/stable/2271594

Frijters, S. (2023). An andersonian-kangerian reduction of term-modal deontic logics.

Garson, J. (2023). Modal Logic. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford encyclopedia of philosophy* (Spring 2023). Metaphysics Research Lab, Stanford University. %5Curl%7Bhttps://plato.stanford.edu/archives/spr2023/entries/logic-modal/%7D

Gillmann, B., Neuerer, D., & Stiens, T. (2023). *'man kann technologie nicht mit verboten aufhalten': Diskussion um chatgpt-regulierung in deutschland*. https://www.handelsblatt.com/politik/deutschland/nach-vorstoss-in-italien-man-kann-technologie-nicht-mit-verboten-aufhalten-diskussion-um-chatgpt-regulierung-in-deutschland/29075002.html

Goranko, V., & Rumberg, A. (2023). Temporal Logic. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford encyclopedia of philosophy* (Fall 2023). Metaphysics Research Lab, Stanford University.

Governatori, G. (2023, June). Defeasible semantics for l4. https://doi.org/10.31219/osf.io/d6f2p

Governatori, G., Maher, M. J., Antoniou, G., & Billington, D. (2004). Argumentation semantics for defeasible logic. *Journal of Logic and Computation*, *14*(5), 675–702. https://doi.org/10.1093/logcom/14.5.675

Governatori, G., & Wong, M. W. (2023). Encoding Defeasible Deontic Logic in Answer Set Programming.

Hansson, B. (1969). An analysis of some deontic logics. *Noûs*, *3*(4), 373–398. https://doi.org/10.2307/2214372

Horty, J. (1989). *An alternative stit operator* [Manuscript, Philosophy Department, University of Maryland].

Horty, J. (2001). *Agency and deontic logic*. Oxford University Press.

Horty, J. F., & Belnap, N. (1995). The Deliberative Stit: A Study of Action, Omission, Ability, and Obligation. *Journal of Philosophical Logic*, *24*(6), 583–644. http://www.jstor.org/stable/30227231

Kammerl, R., & Kniess, M. (2023). *Bildung und ki: Wie umgehen mit chatgpt im schulunterricht?* https://www.zdf.de/nachrichten/digi%09tales/ki-chatgpt-bildungssystem-schule-100.html

Kanger, S. (1972). Law and logic. *Theoria*, *38*(3), 105–132. https://doi.org/https://doi.org/10.1111/j.1755-2567.1972.tb00928.x

Lejewski, C. (1959). Time and modality, by a. n. prior, clarendon press: Oxford university press, 1957. pp. viii + 148. *Philosophy*, *34*(128), 56–. https://doi.org/10.1017/s0031819100029776

Lorini, E. (2013). Temporal STIT logic and its application to normative reasoning. *Journal of Applied Non-Classical Logics*, *vol. 23, no 4*, 372–399.

McNamara, P., & Van De Putte, F. (2022). Deontic Logic. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford encyclopedia of philosophy* (Fall 2022). Metaphysics Research Lab, Stanford University.

Meder, P. J. Y. (2018). *Deontic agency and moral luck [bachelor/master dissertation, unilu - university of luxembourg]* [Master's thesis, ORBilu-University of Luxembourg]. https://orbilu.uni.lu/handle/10993/39770

Moini, B. (2023). *Paradies oder verderben: Chatgpt und die folgen. deutschlandfunk kultur*. https://www.deutschlandfunkkultur.de/kommen-%09tar-chat-gpt-ki-100.html

Murakami, Y. (1998). Utilitarian deontic logic. In M. Kracht, M. de Rijke, H. Wansing, & M. Zakharyaschev (Eds.), *Advances in modal logic* (pp. 211–230). CSLI Publications.

Murakami, Y. (2005). Utilitarian Deontic Logic. *Advances in Modal Logic*, *5*, 211–230.

Nipkow, T. (2013). Programming and Proving in Isabelle/HOL.

Nipkow, T., Paulson, L. C., & Wenzel, M. (2022, October 25). A proof assistant for higher-order logic.

OASIS Open. (2018). *Akoma ntoso version 1.0*. Retrieved June 6, 2024, from https://www.oasis-open.org/2018/09/11/akoma-ntoso-v1-0-akn-oasis-standard-published/

OASIS Open. (2020). *Legalruleml core specification v1.0 from the oasis legalruleml tc approved as committee specification 02*. Retrieved June 9, 2024, from https://www.oasis-open.org/news/announcements/legalruleml-core-specification-v1-0-from-the-oasis-legalruleml-tc-approved-as-com/

Palmirani, M., Governatori, G., Rotolo, A., Tabet, S., Boley, H., & Paschke, A. (2011). LegalRuleML: XML-based rules and norms. In F. Olken, M. Palmirani, & D. Sottara (Eds.), *Rule-Based Modeling and Computing on the Semantic Web: 5th International Symposium, RuleML 2011– America, Ft. Lauderdale, FL, Florida, USA, November*

3-5, 2011. Proceedings (Vol. 7018). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-24908-2

Parent, X. (2021). Preference semantics for hansson-type dyadic deontic logic: A survey of results. In D. Gabbay, J. Horty, R. van der Meyden, & L. van der Torre (Eds.), *Handbook of deontic logic and normative systems*. College Publications.

Prior, A. N. (1959). Thank goodness that's over. *Philosophy*, *34*(128), 12–17. https://doi.org/10.1017/s0031819100029685

Prior, A. N. (1967). *Past, present and future*. Clarendon P.

Prior, A. N. (1968). *Papers on time and tense* (P. F. V. Hasle, Ed.). Oxford University Press UK.

REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL LAYING DOWN HARMONISED RULES ON ARTIFICIAL INTELLIGENCE (ARTIFICIAL INTELLIGENCE ACT) AND AMENDING CERTAIN UNION LEGISLATIVE ACTS.

Rendsvig, R., Symons, J., & Wang, Y. (2024). Epistemic Logic. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford encyclopedia of philosophy* (Summer 2024). Metaphysics Research Lab, Stanford University.

Russell, B. (1908). Mathematical logic as based on the theory of types. *American Journal of Mathematics*, *30*(3), 222–262. Retrieved April 5, 2024, from http://www.jstor.org/stable/2369948

Strasser, C., & Antonelli, G. A. (n.d.). Non-monotonic Logic. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy*. https://plato.stanford.edu/archives/sum2019/entries/logic-nonmonotonic/

University of Cambridge & Technische Universität München. (2023, September 4). *Isabelle*. https://isabelle.in.tum.de/

Uuk, R. (2023, June). *The eu ai act newsletter 33*. https://artificialintelligenceact.substack.com/p/the-eu-ai-act-newsletter-32

Uuk, R. (2024a, March). *The eu ai act newsletter 48*. https://artificialintelligenceact.substack.com/p/the-eu-ai-act-newsletter-48-eu-needs

Uuk, R. (2024b, August). *The eu ai act newsletter 53*. https://artificialintelligenceact.substack.com/p/the-eu-ai-act-newsletter-53-the-law

van Berkel, K., & Lyon, T. (2019). A Neutral Temporal Deontic STIT Logic [arXiv:1907.03265 [cs, math]]. https://doi.org/10.1007/978-3-662-60292-8_25
Comment: Appended version of the paper "A Neutral Temporal Deontic STIT logic", accepted to the 7th International Conference on Logic, Rationality and Interaction (LORI 2019).

Von Kutschera, F. (1986). Bewirken. *Erkenntnis*, *24*(3), 253–281.

Von Wright, G. H. (1963). *Norm and action: A logical enquiry*. Routledge; Kegan Paul.

Von Wright, G. H. (1981). On the Logic of Norms and Actions. In R. Hilpinen (Ed.), *New Studies in Deontic Logic: Norms, Actions, and the Foundations of Ethics* (pp. 3–35). Springer Netherlands. https://doi.org/10.1007/978-94-009-8484-4_1

Xu, M. (2015). Combinations of stit with ought and know. *Journal of Philosophical Logic*, *44*(6), 851–877. https://doi.org/10.1007/s10992-015-9365-7