Friedrich-Alexander-Universität
Technische Fakultät

Master's Thesis in Computer Science

Formalization and Automation of Company Register Processes via Logic

Martin Helmke

Friedrich-Alexander-Universität
Erlangen-Nürnberg

April 14, 2023

Supervisor:
Prof. Dr. Michael Kohlhase

Abstract

Logic-based artificial intelligence appears to be an interesting approach to formalize legal norms and automate legal reasoning. In this respect, the subfield of register law deserves special attention because the corresponding processes in this scope are to some degree "naturally" formalized by legal regulations and practice.

In this thesis, we are concerned with the question to what extent logic, in contrast to other approaches in the area of artificial intelligence, is a suitable means for the formalization and automation of company register processes.

We represent a legal requirements check for an exemplary process from German register law—the appointment of a new director—in first-order logic and, for this purpose, establish a set of formalization conventions. We implement the resulting logic axioms in TPTP as well as in Prolog and identify the advantages and the difficulties of both technologies.

Furthermore, we perform the incorporation of ontological knowledge into our implementation and develop an approach to automatically decompose the formalized rules and to make their application traceable.

As these achievements exploit the logical structure of the formalized rules, the suitability of logic for the intended purposes is substantiated, albeit with considerable reservations associated to the characteristics of first-order logic. This is why we moreover infer that some other logic—possibly a non-monotonic one—might be more appropriate.

We confirm the generality of the conclusions by adapting the developments of this thesis to other company register processes.

Declaration

I confirm that I have written this thesis unaided and without using sources other than those listed and that this thesis has never been submitted to another examination authority and accepted as part of an examination achievement, neither in this form nor in a similar form. All content that was taken from a third party either verbatim or in substance has been acknowledged as such.

April 14, 2023

Martin Helmke

Contents

Chapter 1

Introduction

1.1 Motivation

The research area of *Artificial Intelligence & Law (AI & Law)* has made considerable progress in recent years. This is the case for several fronts, like automated contract analysis or legal prediction. [Vil+22] Governmental initiatives, e.g. the *National Strategy for Artificial Intelligence*[1] launched by the Federal Government of Germany, provide significant impulses for research in AI & Law. [Gug19]

At first glance, the legal domain appears particularly suitable for AI techniques which are based on logic. A conditional legal norm has a logical structure considering that the fulfillment of some prerequisites entails a legal consequence. [Häh+20] Apart from logic-based techniques, the range of AI also includes probabilistic approaches as well as machine learning. [RN09]

A conceivable scope of application for AI is the field of *register law*. Nowadays, German notary's offices perform any application for entry in the company register electronically using structured data formats. [Ste22] Besides the quite strict provisions regarding register processes, e.g. the obligation of their notarization and application, the usage of pre-formulated standard letters for the application is common practice. [GBM22] Thus, the processes are to some extent "naturally" formalized by law. This uniform setting of recurring tasks and the digital progress concerning the standard procedures make register law to a convenient area of application for AI & Law.

The *limited liability company (LLC* or in German the *Gesellschaft mit beschränkter Haftung (GmbH))* is by far the most common legal form of company in Germany. [Fle22] Thus, as processes in this subfield of register law cover a wide range of the daily practice of the notaries and the register

[1] https://www.ki-strategie-deutschland.de/

courts, they offer a great potential for improvement in efficiency by means of automation. In this thesis, we want to formalize and automate register processes via logic-based AI techniques and consider the LLC-related processes as a reasonable starting point.

As a standard compendium for register processes in the scope of Germany, the reference book [GBM22] contains guidelines, requirements, and specimen texts for several prototypical applications at the *German Commercial Register*. An amount of 47 register processes from the mentioned book are related to the LLC. A quite ordinary process with a low to medium complexity is the *appointment of a new director* (A96) for the LLC. This is the exemplary use case which we will be dealing with in this thesis.

Our concrete perspective on this use case is the question whether the requirements for the application are fulfilled and the application is therefore legally valid. A corresponding check involves both formal and material requirements and is otherwise performed manually by a legal expert. The formalization might be considered as a foundation for a legal *expert system*, i.e. a program that solves problems in a narrow area of law via knowledge and expertise in this area. [Ash17, p. 8]

The basis for the formalization is an examination scheme for the director application. Such schemes are often used in the legal domain. We assume the examination scheme as an already completed preparatory work.

1.2 Research Questions

In the course of this thesis, we concern ourselves with the formalization and automation of company register processes using logic. In this context, we intend to answer the following research questions:

1. **Sufficiency of Logic:** Is logic a sufficiently powerful means to formalize and automate company register processes?
2. **Necessity of Logic:** Is logic a necessary means to formalize and automate company register processes?
3. **Most Suitable Logic:** Which kind of logic is most suitable for formalizing and automating company register processes?

1.3 Contribution

Formalization In this thesis, the director application is formalized and a set of formalization conventions in many-sorted first-order logic is proposed.

These conventions establish a uniform and comprehensible structure of the axioms for the norms and for the case facts. The formalization delivers the insight that first-order logic is to some degree suitable for the task. As the open-world assumption of the first-order standard semantics turns out to be unfavorable for the legal domain, some "deformations" are inevitable.

Practical Implementation By implementing the formalization using the TPTP infrastructure, we identify some practical obstacles. Though not all of them can be solved perfectly, we succeed in performing automated reasoning on the application process. The implementation via the logic programming language Prolog, as an alternative approach, benefits from the database semantics, which assume a closed world, and from the good support for arithmetic reasoning. In spite of Prolog's somewhat restricted expressiveness compared to TPTP, it turns out to be sufficient for the use case.

Reasoning Traceability A mentionable accomplishment of this thesis is the development of a reasoning visualization which automatically subdivides the formalized rules and makes the constituent subconditions and their fulfillment traceable. We achieve this by exploiting the logic representation and the constituted formalization conventions. The resulting tool is a useful groundwork for realizing an end-user application, e.g. for use by a notary.

Ontology Incorporation Another subject is the incorporation of ontological knowledge into the implementation. We put this into practice for the legal ontology *LKIF Core* and motivate it for the legal markup language *LegalRuleML*. The compatibility between ontological structures and first-order logic substantiates the benefits of the logic-based approach.

General Suitability of Logic We eventually conclude that, despite some obstacles, the use of logic enables the intended formalization and turns out to be reasonable. Nevertheless, the restricted focus and scope of this thesis do not allow giving a conclusive answer to the question whether applying AI techniques other than logic is meaningful and whether logic is necessary or dispensable. At least, we can substantiate its benefits.

Limited Suitability of First-Order Logic However, first-order logic, which we made use of, gives rise to some difficulties related to the monotonicity and the semi-decidability. This is why it is not necessarily the best option. Instead, an alternative logic might be better suited.

1.4 Thesis Structure

Before we come to the main part of the thesis, some domain-related as well as technical foundations are laid. Chapter 2 clarifies relevant details about the legal domain and introduces an individual case as our running example.

In Chapter 3, we have a brief look at the world of TPTP and at programming in Prolog since these logic infrastructures will later be used for the implementation.

The relations between the subsequent chapters are illustrated in Fig. 1.1 as a kind of pipeline or workflow.

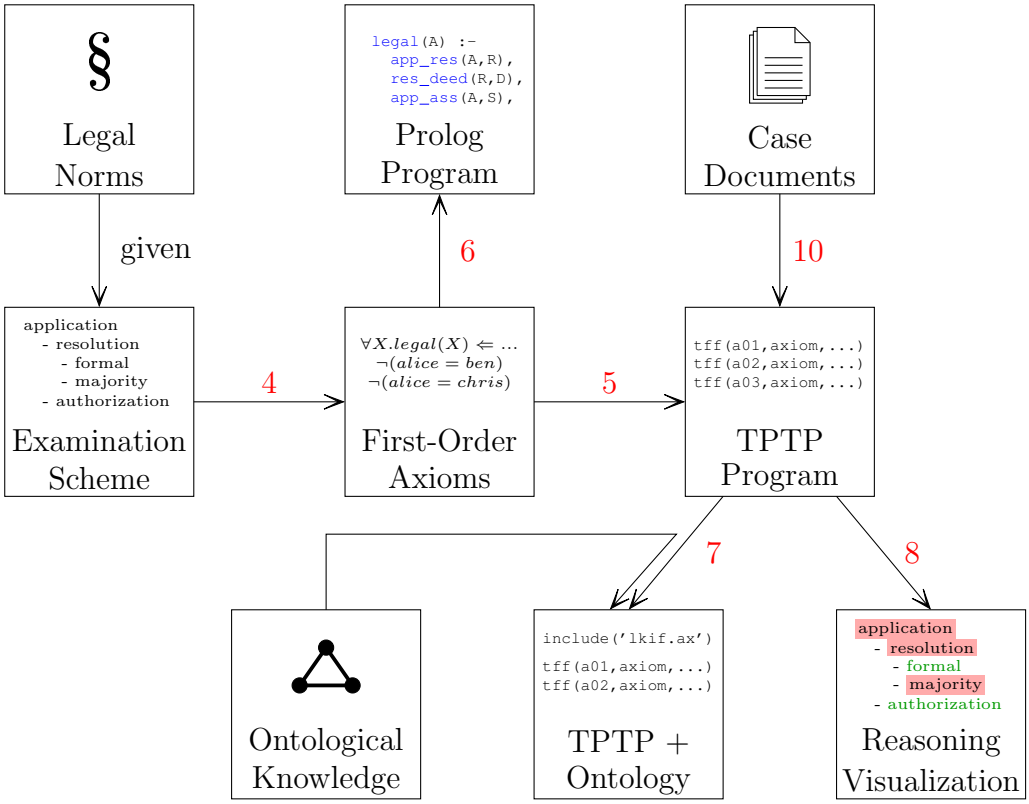

Figure 1.1: Succession of the steps in this thesis

The formalization in first-order logic based on the given examination scheme is performed in Chapter 4. The resulting axioms are then concretely implemented in TPTP (Chapter 5) and in Prolog (Chapter 6).

The next two steps build on the realized TPTP implementation. Chapter 7 discusses two ideas to incorporate ontological knowledge. The tool for

executing and visualizing the logic reasoning is described in Chapter 8.

Chapter 9 covers the evaluation of the adaptability of the developments to other register processes. Before the thesis is eventually concluded, the retrieval of structured case data for the register processes is briefly motivated in Chapter 10.

In the end of most of the chapters, there is an evaluation section in which we summarize the intermediate findings. These will be the basis for the final conclusions, particularly with respect to the research questions.

1.5 Further Remarks

1.5.1 Prerequisites

This thesis requires very basic understanding of

- classical logic, in particular first-order logic,
- ontological representation of knowledge, and
- procedural programming.

This should suffice to be able to follow the explanations.

1.5.2 Published Artifacts

For insights beyond the written content and for concrete implementation details, one can examine the artifacts that resulted from this thesis on GitHub[2]. This includes, in particular, the TPTP and Prolog implementations of the considered register processes as well as the reasoning visualization tool.

1.5.3 Acknowledgements

I hereby express my gratitude to Michael Kohlhase and Max Rapp for supervising this thesis, for accompanying my research, and for giving valuable impulses. This support facilitated my work in the first place. Moreover, I thank Axel Adrian for granting an insight into the everyday working of his notary's office, which allowed me to better relate to the legal environment. Last but not least, particular thanks go to my dear friends Kevin and Hannes for assiduously proofreading the thesis.

[2] https://github.com/eaglebirdi/company-register-processes

Chapter 2

Use Case: Director Application

2.1 Register Law Basics

2.1.1 Limited Liability Company

The scenario in the context of which we aim to do the formalization is the application of the appointment of a new company director for a limited liability company (LLC). The company is assumed to be already in existence and filed in the Commercial Register at the time of the application.

In Germany, the LLC is called *"Gesellschaft mit beschränkter Haftung" (GmbH)*. Its formation, management, and legal conditions are regulated by the act *"Gesetz betreffend die Gesellschaften mit beschränkter Haftung" (GmbHG)*[1]. All legal references in this chapter refer to the GmbHG unless otherwise specified. They will be written according to the pattern which is usual in German jurisprudence. The following explanations are to some extent also based on [GBM22].

The LLC is a collective participation of its shareholders. Its eponymous characteristic is the fact that solely the company capital (and not the shareholders' personal property) is affected by the company's accountability (§ 13 II). Closely related to this is the requirement for a share capital of at least 25,000 € (§ 5 I).

The basis of the company are its *articles of association (AoA)*. Apart from mandatory details, like the company's name and purpose as well as the share capital amount and its division into single shares (§ 3 I), the AoA may codify internal norms which deviate from the general legal regulations (§ 45 II). For example, a resolution is made with a simple majority of the votes

[1] see https://www.gesetze-im-internet.de/englisch_gmbhg/ for an English translation of the GmbHG

cast by law (§ 47 I). Nevertheless, the AoA can stipulate a stricter majority requirement for resolutions, e.g. three quarters of the votes cast.

The company is managed by one or several directors, who have to be natural persons (§ 6). A personal union between director and shareholder is permissible (§ 6 III 1). The directors' legal duties are mostly the company's representation (§ 35 I 1) and book-keeping (§ 41). Beyond this, applications for entry in the Commercial Register, like our director application, are accomplished by the directors as well (§ 78). Their authority finds its limits in the determinations of the AoA or the shareholders' resolutions (§ 37 I).

For the sake of comprehensibility, the previously described rough structure of the company is displayed in Fig. 2.1 as a UML class diagram, which will be further expanded hereinafter. We chose this way of visualization due to its facile tangibility and its conciseness. Furthermore, in Section 8.4, where we present the reasoning visualization, UML will also be the language which we utilize for modeling and visualizing the components.

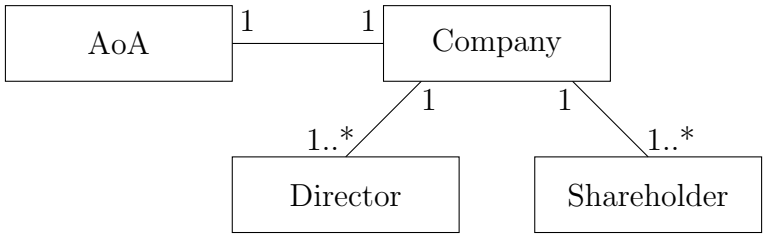

Figure 2.1: Rough structure of the LLC

2.1.2 Commercial Register

The universal provisions concerning the *Commercial Register* (in German *"Handelsregister"*) in Germany are stated in the *Handelsgesetzbuch (HGB)*[2], which is the German commercial code. The Commercial Register is a public register which is kept by the responsible register court and contains every commercial business enterprise (§ 2 S. 1 HGB).

Everyone is entitled to take insight into the register in order to retrieve information (§ 9 I 1 HGB). This information retrieval can be performed using the common register portal of the German federal states[3].

According to the GmbHG, the company's initial registration has a *constitutive effect*, i.e. the official registration marks the beginning of the company's

[2] see https://www.gesetze-im-internet.de/englisch_hgb/ for an English translation of the HGB

[3] https://www.handelsregister.de/

existence (§ 11 I). In contrast, most other actions which require a registration with the register court have a *declarative effect*, i.e. they are immediately effective in law upon the corresponding resolution.

Aside from the details in the register entry (e.g. name, address, share capital, directors), the register court files the AoA and the list of shareholders since both of them must be attached to the initial registration (§ 8 I 1).

Not only must changes in these two documents be reported to the register court in the form of an application, but also changes concerning the directors require this formality (§ 39 I). This leads us back to the scenario we intend to formalize: The application of a new company director.

2.2 Director Application

2.2.1 Checking the Application's Requirements

The primary perspective we want to take on the application is the question whether its legal requirements are satisfied. This denotes a look at the known facts and at a possible implication on the legality of the application. An application which cannot be inferred to be legal would be futile and should not be submitted to the register court.

Jurists often employ examination schemes to check prerequisites. Even though we consider the scheme for the director application's validity as an already finished preparatory work, let us briefly discuss how to come up with such a scheme and how the official walk to the application looks like. Both the official law and the company's AoA provisions contribute to that. § 39 addresses the director application in particular, yet it is only one chunk in the GmbHG which is relevant for the scheme.

As we assume the company to be already filed in the register, a new director can only be appointed by the shareholders (§ 6 III 2, § 46 no. 5). The shareholders' determinations result from a resolution, which is generally made in a personal meeting via a voting (§ 47 I). Given the consent of all shareholders, the meeting may be a teleconference or even be omitted (§ 48).

Submitting the application is the remit of the established directors (§ 78). It has to be executed by a set of directors which is entitled to represent the company. The default necessity of all directors (*joint representation*) can be overridden by the AoA (§ 35 II 1). Two common alternative variants are the *sole representation*, where one director is authorized by themself, and the *modified representation*, where two directors are sufficient.

Moreover, the application demands the attachment of the appointment's deed (§ 39 II) as well as of some assurance in which the new director asserts

the non-existence of reasons opposed to the appointment (§ 39 III 1).

Fig. 2.2 depicts all the entities which are involved in the application process. Aside from that, it delineates the distinction between the *operation-related* and the *operation-independent* objects. While the company, the directors, and the shareholders are static throughout all application processes for a particular company, the other objects, like the resolution and the assurance, are exclusively connected to the considered process instance. This is of course just a general and simplistic statement in view of the fact that the consequence of the director appointment is a change in that "static" data.

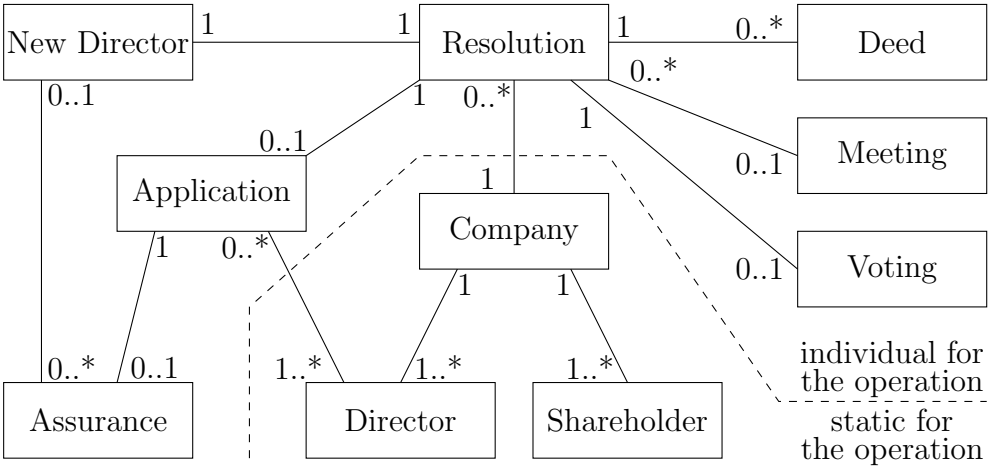

Figure 2.2: Entities in the director application process

After this prosaic overview about the prerequisites, we now present the examination Scheme 2.1, which we will work with. Note that the numbered bullet points in the scheme are necessary conditions in order to fulfill the parent requirement. In contrast, bullet points with the "case" keyword ahead represent a list of (usually mutually exclusive) alternatives, each of which is a sufficient condition for the parent.

The scheme's nested structure primarily comes from three constructs:

1. sets of necessary conditions, i.e. all of them must hold,
2. sets of sufficient conditions, i.e. one of them must hold, and
3. distinctions of cases.

It is likely that both logicians and jurists develop such a legal examination scheme in a way that it mainly consists of those constructs. This is quite convenient for the formalization using classical logic.

The application is legal.

1. The resolution is made legally.

   1.1. The resolution is formally legal.

      case 1.1.1. The resolution is made in a meeting.

         case 1.1.1.1. The meeting is personal.

         case 1.1.1.2. The meeting is a teleconference and all shareholders agree with this format.

      case 1.1.2. There is no meeting.

         case 1.1.2.1. All shareholders consent to the determination.

         case 1.1.2.2. All shareholders agree with voting in writing.

   1.2. The resolution is made with a majority vote.

      case 1.2.1. No special AoA majority requirement.

         1.2.1.1. More yes than no votes.

      case 1.2.2. Special AoA majority requirement.

         1.2.2.1. Yes ratio is beyond the AoA majority requirement.

2. The deed of the director's appointment is attached.

   case 2.1. The deed is original.

   case 2.2. The deed is a certified copy.

3. The new director has signed an assertion for being qualified and for having been instructed about the obligation to disclosure.

4. The applying directors are authorized to represent the company.

   case 4.1. One applicant has sole representation power (via AoA or individual stipulation).

   case 4.2. One applicant has modified representation power (via AoA or individual stipulation) and another director is present.

   case 4.3. All directors perform the application.

Scheme 2.1: Requirements for the director application

For the scope of this thesis, we limit ourselves to this rather coarse scheme even though it does not contain every eventuality and could be itemized much more in depth. For example, there exist a handful of further prerequisites concerning a meeting's form of convocation (§ 51).

Besides these curtailments, we make a couple of assumptions. As already stated, the company is explicitly assumed to be already filed in the Commercial Register. Furthermore, we take a few other commonsense and legal aspects, e.g. from the civil code, as granted. For instance, the shareholders and directors must have full legal capacity to perform their actions. A human intelligence can work out such minutiae on its own whereas an artificial intelligence must either just assume them or explicitly check for them.

Nevertheless, the scheme is a first approximation that handles the majority of possible cases. Moreover, due to a reasonable set of assumptions and a not inconsiderable nesting depth of the scheme, it definitely provides a suitable playground for the formalization.

2.2.2 Associated Documents

The practical procedure of a director application which is submitted to the Commercial Register goes along with the treatment of several documents by the notary's office and the court. Just as with the entities in Fig. 2.2, the documents can be distinguished on the basis of their invariability with respect to the application operation instance. The bundle of at least six documents and their mutual relations are visualized in Fig. 2.3.

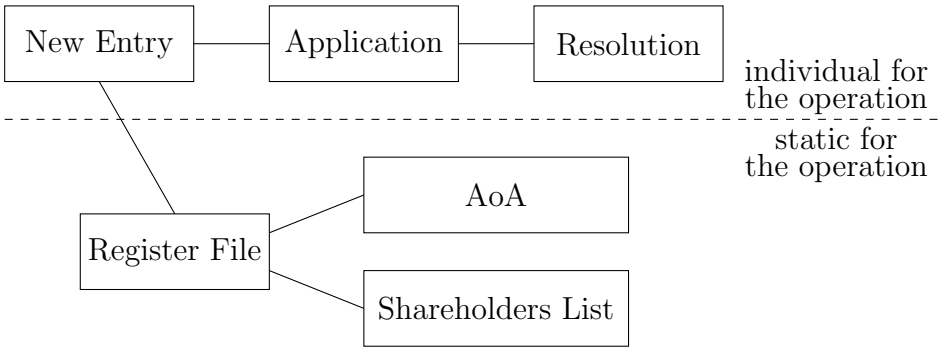

Figure 2.3: Documents in an application process

As mentioned before, the *resolution deed* must be attached to the *written application*. Both of these documents are individual for the application operation. In addition, the notary's office as well as the court have documents available which are largely static in the context of the operation. Those are

the *AoA*, the *list of shareholders*, and the *register file*, which contains the company's metadata, like the name and the registered office.

After the application has been processed by the court, a *new entry* is inserted into the register file.

The six documents can be employed as a source of information for the case-related data of the director application's requirements check. Most of the necessary case details are obtainable from those documents.

2.3 Running Example

In order to allow a better understanding of the formalization, we hereby introduce an exemplary application process by means of which we will explicate the abstract ideas in the following. The example application fulfills the prerequisites from Scheme 2.1 and is therefore legally valid.

Company-Related Details The *Exemplum GmbH*, an LLC with its registered office in Erlangen, is owned by the three shareholders

- *Alice* with 50,001 shares,
- *Ben* with 39,999 shares, and
- *Chris* with 10,000 shares.

The AoA determine that the joint representation power applies for all directors whose representation power is not further specified. Moreover, according to the AoA, resolutions are made with a majority of 60 percent of the votes cast. The company is directed by

- *Jacob* with no specific representation power,
- *Kate* with the modified representation power, and
- *Chris*, who is also a shareholder, with the sole representation power.

Resolution-Related Details The three shareholders recently held a meeting in the form of a videoconference. All of them agreed with this digital meeting format in advance.

One item on the agenda was the question whether a new director *Luca* is to be appointed and whether the modified representation power is to be granted. Alice and Chris vote for this determination whereas Ben rejects the proposal. So, none of the shareholders abstain from the voting.

The 60,001 (50,001 + 10,000) consenting votes of Alice and Chris against the 39,999 votes of Ben constitute a narrow majority of 60.001%. The successful adoption of the resolution was documented in writing.

Application-Related Details Jacob and Kate submit the application of Luca's appointment to the register court. They bring the original document which proves the corresponding resolution as well as the obligatory assurance which has been signed by Luca.

2.4 Evaluation

Examination Scheme as Basis The rules from Scheme 2.1 form an intelligible and modular basis for the upcoming formalization in Chapter 4.

Moderate Complexity of the Director Appointment Compared to the other application processes from [GBM22], the director appointment has a complexity in the low to medium range. As a start, it is therefore an appropriate example for the formalization.

Conjunction- & Disjunction-Based Scheme The examination scheme consists of sets of necessary and sufficient conditions as well as case distinctions. These constructs are associated to conjunctions and disjunctions in classical logic, which is beneficial for the formalization.

Explicit & Implicit Assumptions In order to form a manageable foundation, several implicit and explicit assumptions have to be made, e.g. the legal capacity of the directors and shareholders or the fact that the company is already filed in the Commercial Register. Hence, these aspects do not occur in the examination scheme.

Operation-Related & Operation-Independent Details With a view to the respective company, the details of some application case can be divided into the operation-related data, which is specific to the operation, and the operation-independent data, which is rather related to the company.

Running Example For illustrative purposes, the Exemplum GmbH and its valid director appointment serve as an example case. In the following, the case-related data in formulas and code snippets is taken from that example.

Chapter 3

Technical Preliminaries

3.1 TPTP

Besides the formalization, we also want to perform automated reasoning on the resulting axioms. This is frequently referred to as *Automated Theorem Proving (ATP)*. ATP is the automatically executed inference of conclusions based on a set of given facts. [Sut10] So, having specified the general rules and the known case facts as axioms, we can state some conjecture whose validity is of relevance. ATP responds to the question whether the conjecture can be derived from the axioms and is therefore valid. The world of the *Thousands of Problems for Theorem Provers (TPTP)* [SS01c] provides a suitable infrastructure for ATP in first-order logic (among other logics). We will rely on it as an infrastructure for the implementation of the formalized rules and for automated reasoning on them.

3.1.1 Overview

The TPTP's first release was in 1993. First and foremost, it consists of a library of problems, a library of the corresponding solutions, the TPTP languages, in whose text-based and human-readable syntax the problems are written, and a bunch of ATP tools and services. The TPTP world is not restricted to first-order logic. It also supports higher-order logic via its language THF. [Sut10] However, this is just a side note because our formalization will not go beyond first-order logic's boundaries of expressiveness.

The TPTP libraries provide, on the one hand, an exemplary reference for someone who intends to utilize the TPTP language. On the other hand, the problem and solution libraries are primarily addressed to ATP developers. They can been used as a sample set in order to validate the ATP software

with respect to its correctness as well as to evaluate the software with a view to its performance. [Sut10]

In the scope of this thesis, our role is not the one of a TPTP developer but a user. Our focus is on the TPTP language and the provided ATP tools since we want to use them to implement our formalized rules and facts and to perform automated reasoning on them. The TPTP's reasoning tools which we will use are listed below in Section 3.1.3.

3.1.2 Language Syntax

Disregarding higher-order logic as aforementioned, there exist the TPTP languages CNF (Clause Normal Form), FOF (First-Order Form), and TFF (Typed First-order Form). [Sut17, p. 483] CNF makes the restriction that the formulas must be a conjunction of disjunctions [Fit96, p. 28] whereas FOF does not have such a restriction but allows every kind of first-order formula. As we will use many-sorted first-order logic for the formalization (see Section 4.1.3) and as TFF provides the features of many-sorted logic, from now on, we limit ourselves to this TPTP language. Due to type checks, which are implemented by the TFF-supporting theorem provers, the usage of TFF assists the prevention of flaws. This advantage compensates the constricted conciseness and larger verbosity that arise from explicitly specifying type information.

TFF formulas with minimal information are written one after another as

```
tff(name, role, formula).
```

The first detail `name` in this predicate-like notation is a unique identifier for the formula. `role` describes the formula's function. For our purposes, the three roles `type`, `axiom`, and `conjecture` are sufficient. The third information contains the logical formula, which is phrased according to the syntax definition in [SS01a].

Besides the formula statements, there exists also the `include` keyword, which allows importing other axiom files. [Sut17, pp. 494–496] This enables the partitioning of axioms, which is well-suited for the distinction of different kinds of knowledge, e.g. law-related and case-related information.

The TFF programs in the context of this thesis begin with a set of `type` statements. Here, the sorts and the individuals are declared. Afterwards, the `axiom` statements describe our knowledge in the form of rules and facts. The final statement is the `conjecture`, i.e. the formula which the ATP tool should try to prove.

3.1.3 ATP Tools

In [SS01b], all ATP tools within the realm of the TPTP are enumerated along with their supported features. As it turns out, the tools are not equally suitable for reasoning on the TPTP axioms that will eventually result from our formalization. That is why it is necessary to try out a couple of tools instead of blindly choosing one. The question which of the tools we want to examine in the course of this thesis is dependent on our requirements. We will elaborate on these in the following.

As already mentioned, we want to formulate the axioms in the TFF dialect, which is not supported by all existing tools. Furthermore, arithmetic reasoning must be supported considering that we intend to conduct proofs concerning the majority of the resolutions' votings. There are about twelve tools which satisfy those two requirements. They should be further reduced to a manageable amount.

Once a year, the TPTP provers compete in the *CADE ATP System Competition (CASC)*. [Sut16] The competition's discipline *TFA* (Typed first-order with arithmetic) directly corresponds to our two requirements. So, we include the previous winners *cvc5*, *Vampire*, and *Princess* into the set of considered tools. We disregarded *SPASS+T*[1], the 2011 and 2013 winner, because its latest version 2.2.22 from 2015 is already eight years old today.

In addition, we include the tools which won at one of the remaining disciplines in the last five years (2018–2022) and at the same time fulfill our requirements according to [SS01b]. This holds for *iProver* and *Leo-III*. So, that makes a total of five TPTP tools which will be included in the further analysis.

In Table 3.1, we give a brief overview of these tools. For the sake of simplicity, we take the last version which was officially released on the website or the version control system (mostly GitHub) for each tool (as at January 2023). This pedantic proceeding might make a slight difference for iProver, which only provides one release without an official timestamp, and Leo-III, which has been developed further in the meantime.

As another side note, we did not examine the prover tools with regard to their inner workings, e.g. the implemented calculi or the utilized programming technologies. Instead, we remain in the user role and treat the tools as black boxes.

[1] https://people.mpi-inf.mpg.de/~uwe/software/

Name	Version	Version date
cvc5	1.0.3	2022-12-12
https://cvc5.github.io/		
iProver	3.5	2021-07-11
http://www.cs.man.ac.uk/~korovink/iprover/		
Leo-III	1.6	2021-10-15
http://page.mi.fu-berlin.de/lex/leo3		
Princess	20221103	2022-11-03
http://www.philipp.ruemmer.org/princess.shtml		
Vampire	4.7	2022-08-13
https://vprover.github.io/		

Table 3.1: Considered TPTP prover tools

3.2 Prolog

3.2.1 Overview

Another technology of interest is *Prolog*, which is a programming language that is based on the declarative paradigm. Unlike in conventional, purely procedurally oriented languages, a Prolog program only declares the known data and its relations rather than the procedural logic. Instead, the actual control flow of an executed program instance is in the responsibility of the Prolog interpreter and not of the program code. [Bra86, p. 24f.]

Similarly to predicate logic, Prolog distinguishes between *atoms* (individuals), *variables*, and *relations* (predicates). A relation can take both atoms and variables as arguments.

A program consists of a set of *clauses*. [Bra86, p. 7] There are two types of them. *Facts* declare unconditionally true things. In contrast, *rules* are tied to a set or, to be more precise, a sequence of necessary conditions. A rule is composed of a *body*, which is represented by those necessary conditions, and of a *head*, which is a single conclusion. Since a fact clause has no conditions, its body is empty. Variables in a clause are considered universally quantified. [Bra86, p. 13f.]

Given a set of clauses, a propositional question can be posed to the Prolog system. Prolog answers whether the proposition can be satisfied given the clauses. If this is the case, it also maps the variables that occur in the question to all combinations of particular objects for which the question's answer is positive. [Bra86, pp. 20, 4–7]

In Prolog, one can also perform list-related and arithmetic operations

as well as communication with files, to name only a few of the supported features. [Bra86, p. 64]

One of the existing Prolog implementations is SWI-Prolog[2], which is invoked as a command line tool interpreting a Prolog program and answering posed questions. The questions can be passed either interactively or as a so-called goal parameter via command line arguments. [SWI]

3.2.2 Syntax

In Prolog, the distinction between atoms and variables is made by the fact that atoms start with a lower-case character whereas variables are capitalized. [Bra86, p. 7] Relations are recognizable as their arguments are put in parentheses.

An exemplary rule is written as

```
is_director(X) :- person(X), company_director(_, X).
```

The two characters `:-` split the clause head on the left from the clause body, whose conditions are comma-separated, on the right. The underscore character as the first argument of the relation `company_director` denotes an anonymous variable, i.e. a variable which is not reused in the clause [Bra86, p. 30] but is of course also universally quantified like any other variable.

The example declares that the relation `is_director` holds for every individual for which also the two conditions in the body hold.

A fact, i.e. a clause without any conditions, is written without a body:

```
person(alice).
```

3.2.3 Backtracking

In the following, the procedure which is performed by a Prolog interpreter for answering a question is briefly described. This is interesting because it allows to reproduce how conclusions in Prolog are actually made.

When trying to satisfy a goal, i.e. prove a conjecture, the interpreter searches for a clause with a matching head. The considered goal is substituted with the propositions in the found clause's body as the new goals. This is recursively continued until all new goals can be traced back to unconditional fact clauses.

Whenever such an attempt fails due to the lack of an appropriate rule or fact, it is returned to the superordinate goal and it is searched for another matching clause. This procedure is called *backtracking*. [Bra86, pp. 19–24]

[2] https://www.swi-prolog.org/

As already mentioned, the programmer can concentrate on the program's declarative meaning and leave the procedural aspects out of consideration. [Bra86, p. 24] However, Prolog allows to comprehend and influence the proof procedure at least partially. Using the Prolog debugging features, the proof can be followed step by step. [Bra86, p. 187] Such an insight into the procedure reveals that a clause's body is processed from left to right. Using the *cut* facility (indicated by a `!`-symbol), the programmer can also instruct the interpreter to desist from further backtracking. [Bra86, p. 120]

3.2.4 Relation to First-Order-Logic

In terms of mathematical logic, the facts and rules of a Prolog program can be seen as a set of axioms whereas the question is a conjectured theorem which is tried to be deduced from the axioms. [Bra86, p. 20]

Furthermore, the syntax of Prolog is based on first-order predicate logic but with the restriction that formulas are written as *Horn clauses*. These are disjunctions of literals where at most one is non-negated. [Bra86, p. 61] A Prolog rule can be read as an implication $A_1 \wedge ... \wedge A_n \Rightarrow B$ where B corresponds to the Prolog rule's head and $A_1 \wedge ... \wedge A_n$ to the conditions in the body. This implication can be reformulated to a Horn clause as in Eq. (3.1).

$$\begin{aligned} & A_1 \wedge ... \wedge A_n && \Rightarrow && B \\ \equiv\ & \neg(A_1 \wedge ... \wedge A_n) && \vee && B \\ \equiv\ & \neg A_1 \vee ... \vee \neg A_n && \vee && B \end{aligned} \tag{3.1}$$

Universal quantifications are implicitly supported in Prolog by means of the variables. So, the set of syntactically correct formulas in Prolog could be seen as an equivalent subset of first-order formulas that includes neither existential quantifications nor non-Horn clauses. Another restriction is the fact that one cannot make negative assertions because Prolog uses the so-called *database semantics*. [RN09, p. 343]

Admittedly, the mentioned subset relation does not hold without reservation. Prolog supports features like lists, which are not necessarily in a first-order logic axiom set—at least not in the ones supported in the TPTP world. Furthermore, other than in first-order logic [Rei78], Prolog uses the already mentioned database semantics and makes therefore a so-called *closed-world assumption*. Whatever cannot be derived from the program, is assumed to be false. [Bra86, p. 135]

In summary, one can say that the Prolog world is a partially more restricted but partially more extensive environment than the world of TPTP. With SWI-Prolog, there exists a tool that can be used for ATP.

3.3 Evaluation

Infrastructures for Implementation Having introduced TPTP and Prolog, we can now rely on their infrastructures for the concrete implementation of our formalization and for automated reasoning. In both TPTP and Prolog, the infrastructure consists on the one hand of a language for recording the axioms and on the other hand of software tools which check the particular conjecture.

Limited Expressiveness of Prolog It should also be pointed out again that Prolog—even though it is related to first-order logic—restricts the axiom formulation to Horn clauses which may be wrapped in a universal quantification. Compared to the full expressiveness of first-order logic, this is a quite strong limitation.

Chapter 4

Formalization in First-Order Logic

A judge's decision-making is based on the classical juridical syllogism, which is composed of

1. a *major premise* according to a legal norm,
2. a *minor premise* according to the facts under consideration, and
3. a *conclusion* resulting from the subsumption. [Adr10, p. 529f.]

In general, the major premise takes on the following form: If a set of factual preconditions is fulfilled, then a legal consequence occurs. This reasoning corresponds to the *modus ponens* [Sie20, p. 24], which is an inference rule in propositional logic. Based on propositional logic but more expressive is *first-order logic*. [RN09, pp. 250, 288] Our formalization of the use case is done in first-order logic—more precisely in an extension called *many-sorted first-order logic*, as justified in Section 4.1.3. Utilizing it, we will state axioms from which theorems can be inferred.

First-order logic is built around objects and relations and allows statements about them in terms of quantified expressions. [RN09, p. 289] For this reason and due to its support of deduction applying the modus ponens, it is an appropriate and sufficiently expressive means for our issue at first glance.

In this chapter, we will constitute a set of conventions which we developed and applied in the course of the formalization of the director appointment's application.

4.1 General Set-Up

4.1.1 Basic Signature and Model

We follow the definitions of first-order logic signature and model in [Koh22, pp. 245–248]. In addition, we include the equality relation [End01, p. 127] as well as the basic arithmetic operations on rational numbers.

4.1.2 Law-Related vs. Case-Related Axioms

The source of the juridical syllogism's major premise is a legal norm whereas the minor premise's base is the factual situation of an individual case. In analogy to this distinction, we can assume two separate sets of axioms: the *law-related* ones and the *case-related* ones.

The former represent the general rules determined by the law. These rules constitute the relation between the factual preconditions and the legal consequence. As they feature a *universal applicability*, the axioms are preferably phrased as formulas built up of universal quantifications. Another similar property of these rule axioms is their *invariability*. It does not matter on which case the rules are to be applied. They always stay the same.

Those two properties make the crucial difference to the axioms for the case facts. These, on the contrary, are not universal but describe individual facts that hold only for the single case. Hence, they cannot be used for other operations but must be established for each individual case again.

4.1.3 Many-Sorted Logic

Standard first-order logic considers the universe of individuals as one homogeneous set $\mathcal{D}_\iota$. This is extended by [End01] to many-sorted logic where there exist several sorted universes and where every individual belongs to exactly one sort. Pursuant to this many-sorted concept, we modify our basic signature and model.

According to [End01, pp. 296–299], many-sorted logic can be reduced to one-sorted logic without any loss of expressiveness. As we will see in the following, given a sort indicator, which will be the predicate $\mathcal{Q}_s$, the definitions are based on the foundations of one-sorted logic.

General Modifications Considering $\mathcal{S}$ as the set of sorts and $\mathcal{D}_\iota^{(s)}$ as the universe for the single sort $s \in \mathcal{S}$, the overall universe $\mathcal{D}_\iota$ can be defined as the union of the disjoint sorted universes (see Eq. (4.1)). Mind that the s in $\mathcal{D}_\iota^{(s)}$ indicates the sort and is not an exponent.

$$\mathcal{D}_\iota = \bigcup_{s \in \mathcal{S}} \mathcal{D}_\iota^{(s)} \tag{4.1}$$

Consequently, the range of some k-ary predicate's interpretation function $\mathcal{I}$ is not $\mathcal{P}((\mathcal{D}_\iota)^k)$ anymore (here k is an exponent) like in [Koh22, p. 248]. The predicate's arguments can be specified concerning the respective sort so that the extension of a predicate is not a subset of a k-ary Cartesian power but a subset of a Cartesian product of k sorted universes (see Eq. (4.2), where the k arguments of some predicate p have the sorts s_1,..., s_k).

$$\mathcal{I}(p) \subseteq \prod_{i=1}^{k} \mathcal{D}_\iota^{(s_i)} \tag{4.2}$$

Another important definition addresses the sorted quantifiers, which do not quantify over the entire universe but only over the individuals of the respective sort. For each sort s, we define a monadic predicate $\mathcal{Q}_s$ whose domain is the overall universe $\mathcal{D}_\iota$. $\mathcal{Q}_s$ indicates whether its single argument is of type $s \in \mathcal{S}$ (see Eq. (4.3)).

$$\mathcal{I}(\mathcal{Q}_s) = \{x | x \in \mathcal{D}_\iota^{(s)}\} \tag{4.3}$$

Using this predicate and the unsorted quantifiers, we can define the sorted quantifiers as in Eqs. (4.4) and (4.5). [Obe90]

$$\forall_s X.\mathbf{A} \equiv \forall X.(\mathcal{Q}_s(X) \Rightarrow \mathbf{A}) \tag{4.4}$$

$$\exists_s X.\mathbf{A} \equiv \exists X.(\mathcal{Q}_s(X) \wedge \mathbf{A}) \tag{4.5}$$

In [End01, p. 295f.], the equality relation is only defined for individuals of the same sort. Therefore and due to the fact that the sorted universes are disjoint, two individuals of different sorts can be regarded as unequal.

By using many-sorted logic, we can be very precise concerning the definition of predicates and axioms with respect to the addressed domains—more precise than if we used first-order logic without sorts. Hence, a predicate dedicated for a specific purpose cannot be misused by taking an argument which is out of the predicate's scope of purpose. Not only is this advantageous on a formal level but also and especially on an implementation level, where an erroneous predicate application is automatically recognizable (concretely by TPTP provers). This helps finding mistakes in manually phrased axioms.

As the TFF dialect of TPTP supports sorted logic [Sut10], it is quite straightforward to realize the sort aspect of our formalization by using TFF.

Sorts for Director Application Having stated the primary features and advantages of many-sorted logic compared to unsorted first-order logic, we can now apply it to the director application. We depicted several entities, like the company, the application, and the shareholders, as classes with relations among each other in Fig. 2.2. These classes can be modeled as sorts in our many-sorted first-order model. We regard the *director* and *new director* boxes in Fig. 2.2 as the same sort and take into account that shareholders and directors are persons (with the possibility of personal unions). If we also consider rational numbers ($\mathbb{Q}$) and three sorts for the *representation power*, the *deed format*, and the *meeting format*, we obtain the fourteen sorts and universes in the Eqs. (4.6) and (4.7). Note that we abbreviated the sort names so that the sorted quantifications remain succinct and readable.

$$\begin{aligned} S = \{ \quad & com, sh, dir, pers, res, deed, meet, vot, app, ass, \\ & meetf, deedf, repr, \mathbb{Q} \quad \} \end{aligned} \tag{4.6}$$

$$\begin{aligned} \mathcal{D}_\iota = \; & \mathcal{D}_\iota^{(com)} \cup \mathcal{D}_\iota^{(sh)} \cup \mathcal{D}_\iota^{(dir)} \cup \mathcal{D}_\iota^{(pers)} \cup \\ & \mathcal{D}_\iota^{(res)} \cup \mathcal{D}_\iota^{(deed)} \cup \mathcal{D}_\iota^{(meet)} \cup \mathcal{D}_\iota^{(vot)} \cup \\ & \mathcal{D}_\iota^{(app)} \cup \mathcal{D}_\iota^{(ass)} \cup \\ & \mathcal{D}_\iota^{(meetf)} \cup \mathcal{D}_\iota^{(deedf)} \cup \mathcal{D}_\iota^{(repr)} \cup \mathbb{Q} \end{aligned} \tag{4.7}$$

As the domains of the representation powers, the deed formats, and the meeting formats are independent from the considered case, we call the individuals in these domains *static* data or individuals in the following.

4.2 Individuals & Predicates

4.2.1 Object Individualization

Propositional Approach Despite its limited ability compared to first-order logic [End01, p. 67], propositional logic would indeed also suffice to formalize parts of the requirements check in Scheme 2.1. Let us forget those limitations for a moment. If we implemented the requirements check up to a depth of 1 off the root, a rule formula could look like the one in Eq. (4.8).

$$\begin{aligned} (\quad & is\_resolution\_legal \\ \wedge \quad & is\_deed\_legal \\ \wedge \quad & is\_assurance\_legal \\ \wedge \quad & are\_applicants\_authorized \\ ) \Rightarrow \quad & is\_application\_legal \end{aligned} \tag{4.8}$$

This propositional expression is easy to comprehend. Understandably, the deeper we would implement the scheme tree, the more atomic propositions would be necessary. They would be available as a large cluttered set of atoms without any structure. Not only does the usage of first-order logic provide more expressiveness [End01, p. 67], but it also allows us to bring structure into the set of propositions. The propositions are expressed as predicates and can therefore be related to the individuals about which they make a statement.

Introducing Individuals A trivial translation of Eq. (4.8) into first-order logic is the conversion of every proposition into a monadic predicate. Taking up on the sorts from Eq. (4.6), we furthermore introduce for every entity in our use case one individual with the appropriate sort and set the monadic predicates' domains to the particular sort which it concerns with respect to the intended legal meaning. The resulting first-order formula is written in Eq. (4.9).

$$\begin{aligned} (\quad & is\_resolution\_legal(res1) \\ \wedge \quad & is\_deed\_legal(deed1) \\ \wedge \quad & is\_assurance\_legal(ass1) \\ \wedge \quad & are\_applicants\_authorized(app1) \\ ) \Rightarrow \quad & is\_application\_legal(app1) \end{aligned} \tag{4.9}$$

In a very extensive formalization, the decomposition of the scenario into several entities is beneficial in assuring a clear overview and also helps to facilitate the communication between the logician who conducts the formalization and the legal expert.

Non-Existence of Individuals In order to represent the real world accurately, we constitute the convention that an individual is only introduced for real-world objects about which we actually know. An object which does not exist, e.g. a meeting that has not been held, has no representation in the formalization.

4.2.2 Binary Relation Predicates

Motivation Eq. (4.9) evaluates the properties of the four individuals $app1$, $res1$, $deed1$, and $ass1$. Let us assume a restricted environment where there are no other individuals than those four. We could without any difficulty neglect that there is no explicit mention of some facts which are actually relevant in our scenario:

- Is it the resolution $res1$ which is attested by $deed1$?
- Is $res1$ the subject of the application $app1$?
- Is the signer of the assurance $ass1$ the new director appointed in $res1$?

We do not want to restrict ourselves to an environment where these relations between the individuals can just be disregarded. Such an environment would not enable the treatment of a more complex application process or the treatment of two different processes at once. Understandably, the associations in Fig. 2.2 have to be modeled, too.

Defining Relations They will be represented by binary predicates that take the two linked individuals as arguments. This enables navigating from one individual to another, which will be necessary to define the legal rules like the one in Eq. (4.9). For example, the relation between the application and the resolution has the domain as indicated in Eq. (4.10).

$$\mathcal{I}(application\_resolution) \subseteq \mathcal{D}_\iota^{(app)} \times \mathcal{D}_\iota^{(res)} \tag{4.10}$$

We only have relations between at most two individuals in our formalization. Thus, binary predicates are sufficient.

Predicate Reification However, one could imagine an alternative version of the model where there does not exist a dedicated sort for the shareholder. Instead, there would exist a direct relation between the company and the person. The relation would also embed the shareholder's voting rights as a natural number (denoted by $\mathbb{N}$ as a subset of $\mathbb{Q}$). See Eq. (4.11) for the structure of that ternary relation predicate.

$$\mathcal{I}(company\_shareholder) \subseteq \mathcal{D}_\iota^{(com)} \times \mathcal{D}_\iota^{(pers)} \times \mathbb{N} \tag{4.11}$$

We decided to avoid predicates with more than two arguments in favor of a *reification*-based approach called *binary representation*. The n-ary predicate corresponding to the relation is turned into a term. This "reified" predicate corresponds to the now existing sort for the shareholder. The relation partners are declared using separate predicates that take the reified term as an argument so that we end up with binary predicates again.

The binary representation pattern is advisable because the relation can subsequently be extended with new properties without having to modify any predicate. Instead of that, it is only necessary to define a new predicate for the new property. [Kow78, p. 79f.] Furthermore, navigating through the

object relations in a quantified expression is way easier because one does not need to quantify over all the arguments that do not contribute to the information which just affects the relation between two individuals.

Applying the pattern to the predicate from Eq. (4.11) introduces the sort *sh* (shareholder) and the three predicates in Eq. (4.12).

$$\begin{aligned} \mathcal{I}(shareholder\_company) &\subseteq \mathcal{D}_\iota^{(sh)} \times \mathcal{D}_\iota^{(com)} \\ \mathcal{I}(shareholder\_person) &\subseteq \mathcal{D}_\iota^{(sh)} \times \mathcal{D}_\iota^{(pers)} \\ \mathcal{I}(shareholder\_votes) &\subseteq \mathcal{D}_\iota^{(sh)} \times \mathbb{N} \end{aligned} \tag{4.12}$$

In general, reification is a helpful approach to extend some statement with context information, e.g. a time point (like in event calculus) [Mue15] or the observer of the statement. If we had to model such context information, reification would be the way to go.

4.2.3 Case Facts

We already mentioned the distinction between law-related and case-related axioms. Since the interrelated individuals are specific to the considered application case, the relational assertions belong to the case-related axioms. Apart from that, there exist case facts which do not refer to a connection between two objects but to a single object's property, e.g. the format of the resolution deed or a shareholder's voting rights. We will generally state such properties using binary predicates—just like the inter-object relations.

As an exception, boolean properties can be represented by a monadic predicate with the affected individual as its argument. For example, the application can specify whether the new director is allowed to do self-dealings. A monadic predicate is sufficient for this yes-or-no property.

4.3 Rule Axioms

Motivation In Eq. (4.9), we created an initial approach for the definition of a legal rule concerning the validity of the director application. In order to make this more general in a way that does not refer to explicitly named individuals, we have to replace these individuals by variables over which is universally quantified. Furthermore, the questions concerning the object relations, which we listed in Section 4.2.2, still have to be incorporated. The relations can simply be added as conjunction operands on the left side of the implication. The resulting formula for the rule *is_application_legal* is formulated in Eq. (4.13).

$$
\begin{aligned}
&\forall_{app} A.\ \forall_{res} R.\ \forall_{deed} D.\ \forall_{ass} S.\ (\\
&\quad (\quad is\_resolution\_legal(R)\\
&\qquad \wedge\ is\_deed\_legal(D)\\
&\qquad \wedge\ is\_assurance\_legal(S)\\
&\qquad \wedge\ are\_applicants\_authorized(A)\\
&\qquad \wedge\ application\_resolution(A, R)\\
&\qquad \wedge\ resolution\_deed(R, D)\\
&\qquad \wedge\ application\_assurance(A, S)\\
&\quad) \Rightarrow is\_application\_legal(A)\\
&)
\end{aligned}
\tag{4.13}
$$

As a universally quantified implication with a left side consisting of a conjunction of atomic formulas and an atomic right side, it directly corresponds to Horn clauses, which are used to define rules in Prolog. We will take up this observation later when implementing the formalized process in Prolog.

On the legal interpretation level, the rule from Eq. (4.13) concerns a special property of the application—its *validity*. Since it is the application whose property the validity is, it might appear a bit counterintuitive that the universal quantification embraces other variables than only the application.

Coarse Rule Pattern The formula can be reformulated so that the other variables are handled by an existential quantification which does not embrace the right side of the implication—$is\_application\_legal(A)$. The resulting Eq. (4.14) can be derived according to the transformation in Appendix A. The formalized rule is equally understandable in this form but focuses more on the application individual.

$$
\begin{aligned}
&\forall_{app} A.\ (\\
&\quad \exists_{res} R.\ \exists_{deed} D.\ \exists_{ass} S.\ (\\
&\qquad\quad is\_resolution\_legal(R)\\
&\qquad \wedge\ is\_deed\_legal(D)\\
&\qquad \wedge\ is\_assurance\_legal(S)\\
&\qquad \wedge\ are\_applicants\_authorized(A)\\
&\qquad \wedge\ application\_resolution(A, R)\\
&\qquad \wedge\ resolution\_deed(R, D)\\
&\qquad \wedge\ application\_assurance(A, S)\\
&\quad) \Rightarrow is\_application\_legal(A)\\
&)
\end{aligned}
\tag{4.14}
$$

Let us phrase the rule in natural language: An application is legal if there exist a related legal resolution, an attached deed for this resolution, as well

as a related signed assurance, and if the application is carried out by an authorized set of applicants.

We can observe a coarse general pattern that is applicable for rules which are not a leaf node in the scheme: The formula consists of a *universal quantification* whose *body is an implication*. The implication's right side is a *monadic predicate* that takes the *quantification variable as its argument*. The left side is an *existentially quantified conjunction* of monadic *rule predicates* and binary *relation predicates*. When reaching a scheme's leaf node, an operand of the conjunction can also be a case fact or any other first-order proposition.

Rule Pattern Refinement Let us refine our pattern by taking the example of $is\_resolution\_legal$, which we already used in Eqs. (4.8) and (4.14). According to item 1 and its two direct children in Scheme 2.1, a resolution is legal if it is formally legal (1.1) and if it is made with a majority vote (1.2). For this simple rule, the implication's left side is a conjunction of two other rules. These take the resolution variable as an argument. Thus, as navigating to other individuals via the relations is not necessary, the existential quantification around the conjunction can be omitted, see also Eq. (4.15).

$$\begin{aligned}
&\forall_{res} R.\ (\\
&\quad (\quad is_resolution_formally_legal(R) \\
&\quad\ \wedge\ \ has_resolution_majority(R) \\
&\quad) \Rightarrow is_resolution_legal(R) \\
&)
\end{aligned} \tag{4.15}$$

Apart from rules with two or more necessary conditions, Scheme 2.1 also has rules where the fulfillment of one subcondition is sufficient. For example, the attached resolution deed is legal if it is either the original document (2.1) or a certified copy (2.2). Such a rule can be equivalently formulated as in Eq. (4.15) but using a disjunction instead of a conjunction.

Summarized Pattern In addition to the coarse general pattern for rule definitions, we can fix that the implication's left is either

- a conjunction of other rules,
- a disjunction of other rules, or
- one or more existential quantifications whose combined body is a conjunction of others rules and relations.

Due to its monadicity, a rule predicate can be applied isolatedly to the corresponding individual. Later, this property and the restricted pattern for rule definitions will be beneficial for the reasoning visualization in Chapter 8.

Rule Definitions for Scheme Leaf Nodes We came to the realization that the examination scheme's inner nodes consist predominantly of nested conjunctions and disjunctions and can therefore be modeled with the quite restricted pattern that we just established.

However, the rules in the scheme leaves need more flexibility. For instance, item 1.2.2.1 in Scheme 2.1 involves an arithmetic comparison between the AoA's majority requirement and the voting's yes-no-ratio. For this purpose, the case facts are retrieved and evaluated. So, we do not fix a general pattern for leaf nodes but allow the usage of all first-order logic expressions.

4.4 World-Closing

4.4.1 Predicate Completion

First-order logic's standard semantics make the *open-world assumption*. If a proposition is neither an axiom nor inferable from these, then it may be true or may be false without certainty. In contrast, the closed world of the so-called *database semantics* regards unprovable propositions as false. However, by using first-order logic for the formalization, we "live" in the open world. [Rei78; RN09, p. 300] Related to this is the *monotonicity* property. Conclusions from known facts cannot be invalidated by extending the knowledge. The set of conclusions can only increase along with the given axioms. [RN09, p. 251f.]

Let us have a look at item 1.1.2.1 in Scheme 2.1, which is the proposition that all company shareholders consent to the resolution's determination. Given that the shareholders are related to their company in the case-related axioms via the predicate *shareholder_company* and their consent to the determination is stated via *shareholder_consents*, we have the rule axiom as defined in Eq. (4.16).

$$
\begin{aligned}
&\forall_{res} R.\ (\\
&\quad \exists_{com} C.\ (\\
&\qquad\quad company\_resolution(C, R)\\
&\qquad \wedge\ \ \forall_{sh} S.\ (\\
&\qquad\qquad\quad shareholder\_company(S, C)\\
&\qquad\qquad\quad \Rightarrow\\
&\qquad\qquad\quad shareholder\_consents(S, R)\\
&\qquad\qquad)\\
&\quad) \Rightarrow all\_shareholders\_consent(R)\\
&)
\end{aligned}
\tag{4.16}
$$

If all three shareholders Alice, Ben, and Chris consent, the axioms which are relevant for the rule could be stated as in Eq. (4.17).

$$\begin{aligned}
&company_resolution(com1, res1).\\
&shareholder_company(shareholder_alice, com1).\\
&shareholder_company(shareholder_ben, com1).\\
&shareholder_company(shareholder_chris, com1).\\
&shareholder_consents(shareholder_alice, res1).\\
&shareholder_consents(shareholder_ben, res1).\\
&shareholder_consents(shareholder_chris, res1).
\end{aligned} \tag{4.17}$$

Even if all company shareholders seemingly consent according to these axioms, the proposition $all\_shareholders\_consent(res1)$ cannot be inferred. The axioms make statements about the minimum extension of the corresponding predicates. Nevertheless, the open world does not prescribe that the extension does not go beyond that. The axioms in Eq. (4.17) do not exclude that there might exist another shareholder which perhaps does not consent—a situation where the proposition $all\_shareholders\_consent(res1)$ is obviously not true anymore. If the axioms in Eq. (4.17) were sufficient for the inference, it would be at odds with the monotonicity property of first-order logic considering that an additional axiom for another company shareholder would invalidate the inference.

In order to make particular inferences, the open world coerces us into stating not only what we know but also that there is nothing beyond our knowledge. In [Rei82], a "world-closing" approach for axiomatic assertions is proposed and named using the convincing term *predicate completion*. The extension of a predicate is specified conclusively via a bi-implication, which is exemplarily indicated for $shareholder\_company$ in Eq. (4.18).

$$\begin{aligned}
&\forall_{sh} S.\ \forall_{com} C.\ (\\
&\quad shareholder_company(S, C)\\
&\quad \iff\\
&\quad (\quad (S = shareholder_alice \quad \wedge\ C = com1)\\
&\quad\ \vee \quad (S = shareholder_ben \quad \wedge\ C = com1)\\
&\quad\ \vee \quad (S = shareholder_chris \quad \wedge\ C = com1)\\
&\quad)\\
&)
\end{aligned} \tag{4.18}$$

The extension of the predicate is stated as a disjunction of conjunctions. Each disjunction operand corresponds to an applicable tuple and each conjunction operand corresponds to a single item of the tuple. In the present

example, we neglect the possibility to extract $C = com1$ from the brackets. This allows to immediately perceive the tuples in the predicate's extension.

If the extension of a predicate is the empty set, then the right side of the bi-implication is simply the trivially unsatisfiable proposition F.

Understandably, legal reasoning can only be performed based on known facts. Therefore, all case-related axioms in our formalization are preferably formulated using predicate completion so that unknown facts are safely ignored.

4.4.2 Conclusive Sort Domains

Another reasonable world-closing approach is the conclusive enumeration of static data instances. A meeting can only happen in personal or as a teleconference. So, it can be explicitly stated that there exist no other formats than those two. Similar to predicate completion, the domain-concluding axiom in Eq. (4.19) makes a restriction to the otherwise open world.

$$\begin{aligned} &\forall_{meetf} X.\ (\\ &\qquad X = personal \\ &\ \vee \quad X = teleconference \\ &) \end{aligned} \tag{4.19}$$

4.4.3 Inequality Declarations

In addition to the previously explained aspects, the open world also implicates the possibility that two named individuals are actually the same symbol even if they are named with two different labels.

Director Kate from our example has the modified representation power. Hence, she can represent the company together with another director. If there are not any inequality declarations, a proof for the existence of the representation authority cannot be provided. The representation authority's requirements are not fulfilled in every model because there exist models where the two executing directors are actually the same individual—despite being described by different labels. Therefore, those models have to be excluded by declaring all directors to be mutually "unsame".

In order to prevent situations as just mentioned, we constitute the convention to explicitly declare the inequality of all same-sort individuals. The only exception are scenarios where the possibility of equality should actually be left open. Individuals of different sorts need not be considered in the inequality declarations because they are unequal anyway in many-sorted logic (see Section 4.1.3).

Eq. (4.20) contains an exemplary axiom that declares the mutual inequality for the three directors. Given n individuals of the same sort, $\frac{n \cdot (n-1)}{2}$ inequality declarations must be established.

$$\begin{aligned} & \neg(director\_jacob = director\_kate) \\ \wedge\ & \neg(director\_jacob = director\_luca) \\ \wedge\ & \neg(director\_kate = director\_luca) \end{aligned} \tag{4.20}$$

4.5 Default Behavior

4.5.1 Default Values

In the legal domain, it is frequently the case that the law defines a general regulation which can be overridden by a special one. As already mentioned, a company's AoA can stipulate a special majority requirement for resolutions. The lack of such a specification in the AoA leads to a fallback to the law-given norm, which is the simple majority, i.e. 50 percent of the votes.

According to its AoA, the Exemplum GmbH requires 60 percent. Let us consider a predicate *company_majreq_AoA* which represents a company's majority requirement from the AoA and another predicate *company_majreq* which provides a company's actual majority requirement having regard to both the AoA and the fallback. While the former predicate is just a case fact, the latter requires an implicational rule axiom which checks the existence of a corresponding *company_majreq_AoA* assertion.

The non-existence of an assertion can only be proved in the open world if the extension of the respective predicate is conclusively stated. Consequently, *company_majreq_AoA* must be asserted via predicate completion. Of course, we want *company_majreq* to be conclusively stated, too.

$$\begin{aligned} & \forall_{com} C.\ \forall_{\mathbb{Q}} X.\ (\\ & \quad company\_majreq(C, X) \\ & \quad \Longleftrightarrow \\ & \quad (\quad company\_majreq\_AoA(C, X) \\ & \quad\ \vee\ (\\ & \qquad\qquad (\neg\exists Y.\ company\_majreq\_AoA(C, Y)) \\ & \qquad\ \wedge\ (X = 0.5) \\ & \qquad) \\ & \quad) \\ &) \end{aligned} \tag{4.21}$$

Looking at Eq. (4.21), the bi-implication ensures the conclusiveness of the predicate's extension. The disjunction on the bi-implication's right side constitutes the distinction of the two mutually exclusive cases. The first one adopts the AoA's value given that there is a specification in the AoA. The second case applies if there is no special regulation. It determines the majority requirement to be equal to the default value.

4.5.2 Assumed Subconditions

Another practical issue of the formalization is the eventuality that we do not have any information about some requirement that is actually necessary for the application's overall validity. This could occur, for example, because the information does not definitely arise from the input data.

In order to perform the requirements check nevertheless, it is inevitable to simply assume that the subconditions which we do not surely know are actually given. So, we have to extend the case-related axioms with that assumption.

For example, according to item 1.1.1 in Scheme 2.1, a legal meeting is necessary. Given that we do not know about any meeting, we just make the default assumption that there was some legal meeting addressing the resolution $res1$. An appropriate axiom formula is written in Eq. (4.22).

$$\begin{aligned} &\exists_{meet} M.\ (\\ &\quad resolution\_meeting(res1, M) \\ &\ \wedge\ is\_meeting\_legal(M) \\ &) \end{aligned} \tag{4.22}$$

As we have no information about any meeting, there does also not exist any meeting individual (according to the convention in Section 4.2.1). Thus, the set of knowledge-based case axioms does not contain any relational statement establishing a connection between the resolution and a meeting.

In Eq. (4.22), it is stated that, nonetheless, there exists some meeting related to the resolution $res1$ and it was legal. Starting from the resolution, it is navigated to that meeting via the relation predicate $resolution\_meeting$ and it is made a statement about the meeting. Theoretically, the navigation can go over more than only one relation. This option of multi-stage navigation was already made use of in the rule in Eq. (4.14).

The assumptive axiom in Eq. (4.22) is sufficient to fill the gap that results from the lack of knowledge and therefore allows to satisfy the requirements check for the application.

4.6 Evaluation

Successful Realization The undertaking of formalizing the requirements check for the considered use case, namely the application of a director appointment, was successful on the whole.

Many-Sorted Conciseness Applying many-sorted logic allows a concise notation that supports the distinction between different sorts of individuals. Explicit sort-indicating predicates, which would be necessary in one-sorted logic, are therefore dispensable. This allows for avoiding verbosity.

General Rules vs. Case-Related Facts The juridical syllogism's implicational character allows a comprehensible distinction between general rules and case-related facts. Together with the tree-shaped structure of complex norm schemes, the juridical syllogism provides a convenient basis for a straightforward translation into first-order logic.

Conventions for Rule Definition Furthermore, it enables a reasonable restrictiveness with respect to the definition of rule axioms. As their structure is very close to Horn clauses, a success in the pending implementation in Prolog (see Chapter 6) appears imaginable as well.

Deformation due to World-Closing The formalization suffers a little from the open-world assumption of first-order logic's standard semantics. Constructs like predicate completion and the inequality declarations make the open world manageable. However, it is an unaesthetic deformation, which contributes to the verbosity of the formalization.

Weightiness due to World-Closing Moreover, the bloated axioms might be an issue concerning the complexity and thus runtime of a corresponding logic program. The universally quantified bi-implication of the predicate completion corresponds to way more primitive axioms than a single atomic statement.

High Manual Effort A last practically relevant observation is the effort of the manual conversion of the examination scheme into first-order logic axioms. Despite being straightforward, the formalization by hand turns out to be a quite time-consuming task.

Chapter 5

Implementation in TPTP

Now where the director application has been formalized in first-order logic, we want to practically implement the axioms as well as the conjecture using the TPTP infrastructure. As already justified, the TPTP dialect TFF will be used for the implementation.

5.1 Program Assembling

For structuring the axioms in a comprehensible way, they are split into several files, which are externally included in a main program file `program.p`. When speaking of "references" or "dependencies" (e.g. in Fig. 5.1) between the files in the following, we do not necessarily mean an explicit reference to the target file but the usage of symbols which are declared in the target file.

types.ax is the most basic file. It is referenced by all other files and consists exclusively of TFF type declarations regarding all sorts and all predicate signatures. These declarations have to be stated explicitly in TFF in order to build an executable program.

staticdata.ax contains the introduction of all individuals that belong to the static data (see Section 4.1.3). Such an introduction must be done for all typed individuals in TFF. Besides that, an inequality declaration for all the individuals of the same type is made. The static individuals are referenced by `rules.ax`, `casefacts.ax`, and `defaults.ax` because all those three may mention them in their axioms.

rules.ax states all the law-related axioms. These follow the conventions which were established in Section 4.3.

individuals.ax introduces all individuals which are not static (as the ones in `staticdata.ax`) but individual for the considered case.

casefacts.ax is the file that contains the axioms which assert the relations between the individuals from `individuals.ax` as well as the case facts. Both of these declarations are phrased via predicate completion.

defaults.ax contains firstly the default values according to Section 4.5.1. A default-valued predicate must not be asserted in `casefacts.ax` at the same time whereas the special value is allowed to be asserted there. Secondly, if there exist assumed subconditions (see Section 4.5.2), then they are specified in `defaults.ax`. This file therefore references `individuals.ax`.

The order of the six files' inclusion into the main file `program.p` is constrained by the interdependencies that are depicted in Fig. 5.1. The TFF-supporting provers require that types, predicates, and individuals have already been declared before they are used. The order in which the six files were just described is an option that satisfies the constraint.

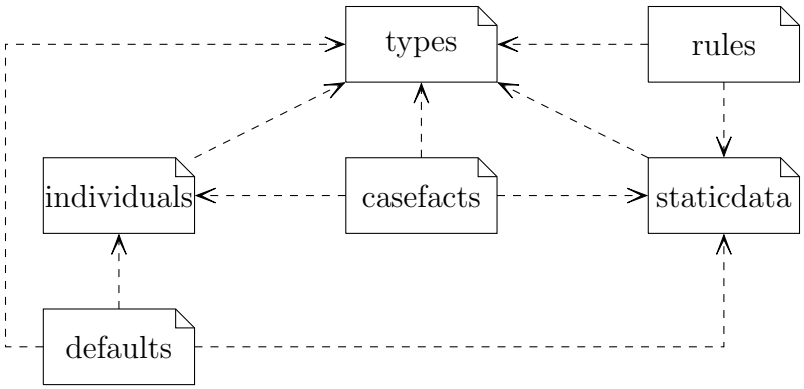

Figure 5.1: TFF files and their interdependencies

Besides the files' inclusion, the main program specifies the conjecture which is to be checked by the prover. Given that the individual that represents the application has the name `app1`, the program text for the requirements check corresponds to the one in Listing 5.1.

```
include('types.ax').
include('staticdata.ax').
include('rules.ax').
include('individuals.ax').
```

```
include('casefacts.ax').
include('defaults.ax').

tff(theorem_in_question, conjecture,
  is_application_legal(app1)
).
```

Listing 5.1: TFF main program for the director application

5.2 ATP Tool Invocation

Next, the assembled program file `program.p` is passed to the prover. In Section 3.1.3, we committed ourselves to the five tools cvc5, iProver, Leo-III, Princess, and Vampire. All of these can be executed via command line.

In order to simplify the prover call, it is convenient to install the considered tool in a way that it can be invoked from everywhere on the system via its name. For instance, on an Ubuntu system, storing the tool or an interposed shell script with the tool's name in a directory that is mentioned in the `$PATH` variable does the job.

A command for invoking Vampire on our `program.p` and a possible output is written in Listing 5.2. The additional option `-t` specifies the maximum time for the operation whereas `-p` determines in how much detail the proof is explicated in the output.

```
~$ vampire -t 5 -p off program.p
% Running in auto input_syntax mode. Trying TPTP
% Refutation found. Thanks to Tanya!
% SZS status Theorem for program
% ------------------------------
% Version: Vampire 4.7 (commit 2d02e4655 on
   2022-07-11 21:15:24 +0200)
% Linked with Z3 4.8.13.0
   f03d756e086f81f2596157241e0decfb1c982299 z3
   -4.8.4-5390-gf03d756e0
% Termination reason: Refutation

% Memory used [KB]: 5628
% Time elapsed: 0.010 s
% ------------------------------
% ------------------------------
```

Listing 5.2: Command for invoking Vampire prover and the output

The status "Theorem for program" (see output line 3) indicates that the proof is successful. There exist also other possible status, like "Counter-Satisfiable" for a non-performable proof or "ContradictoryAxioms" for an inconsistent axiom set. [SS01d]

Of course, for the four other prover tools, the command line arguments and the outputs are different. Apart from that, the invocation is mostly equivalent.

5.3 Tool-Dependent Limitations

During the tool examination, we observed on the one hand that different tools return different results for the same logic program. On the other hand, some tools have difficulties with particular constellations. In the following, for each noticed critical situation, we will

1. describe the circumstances and explain their *practical relevance* in our formalization,
2. *set up an experimental program* to test the tools' behavior in the particular situation,
3. evaluate the *test results*, and
4. propose a *remedy* for the detected problems as far as possible.

The experimental programs are mostly an ordered sequence of type declarations, axioms, and finally the conjecture. The execution is carried out on Ubuntu 22.04. A timeout of five seconds is configured for all provers.

5.3.1 Validity vs. Falsifiability

Practical Relevance A TPTP program contains either a conjecture that semantically holds or a conjecture whose validity is not necessarily entailed by the axioms. The provers must be able to handle both scenarios in order to be suitable for our purposes.

Experimental Set-Up The conjecture of the test program in Listing 5.3 is implied by the two propositions in the axioms `a02` and `a03`. The validity can be toggled by removing or negating one of those axioms. The program does not apply any potentially problematic TPTP features.

```
tff(t01, type, res: $tType).
tff(t02, type, is_legal: res > $o).
tff(t03, type, is_formally_legal: res > $o).
tff(t04, type, has_majority: res > $o).
tff(t05, type, res1: res).

tff(a01, axiom,
  ! [R: res] :
    ( is_legal(R)
      <=
      ( is_formally_legal(R)
      & has_majority(R)
      )
    )
).
tff(a02, axiom, is_formally_legal(res1)).
tff(a03, axiom, has_majority(res1)).

tff(tiq, conjecture, is_legal(res1)).
```

Listing 5.3: `trivial_valid.p`: TFF program without special features

Test Results All tools can prove the valid conjecture. However, the falsifiable conjecture cannot be identified as such by Princess because a timeout occurs (even for larger timeout values than five seconds).

Remedy There exists no obvious workaround for the observed deficit of Princess. At worst, one could accept that a timeout simply indicates the non-existence of a proof. Indeed, this is a quite unsatisfying solution.

5.3.2 File Inclusion & Existential Quantification

Practical Relevance As mentioned before, in the interest of a comprehensible structure, the extensive logic program is split into several files. The main program consists solely of `include` statements and some conjecture. This conjecture might be an existential quantification, as we will see in the course of the reasoning visualization in Chapter 8.

Experimental Set-Up The TPTP program in Listing 5.4 has a trivial conjecture which is phrased as an existential quantification. It includes the file from Listing 5.5, in which some type is declared.

```
include('included.ax').
tff(tiq, conjecture, ? [X: res] : ($true)).
```

Listing 5.4: `includes.p`: TFF program which includes another axiom file and whose conjecture is an existential quantification

```
tff(t01, type, res: $tType).
```

Listing 5.5: `included.ax`: The included axiom file

Test Results As the only one out of the five tools, iProver aborts the execution of `includes.p` due to an unexpected exception. The problem can be fixed by either replacing the `include` statement with the content of the included file or by adding some other statement to the program file. Apparently, iProver has a bug in consequence of which an existentially quantified conjecture cannot be handled if the program file contains no other statements than file inclusions.

Remedy A rather primitive remedy is the assertion of some dummy statement in the program file. Alternatively, one could dispense with the subdivision into several files and write all statements into one single large file.

5.3.3 Arithmetic Reasoning

Practical Relevance In Section 3.1.3, we restricted the selection of the provers to those which support arithmetic reasoning. The director application's resolution must satisfy a majority requirement and the approval rating has to be computed and compared to that requirement. This is preferably specified in the axioms and performed by the provers.

Given that the ratio of yes votes is already available via the predicate *voting_yesratio*, the rule axiom looks like in Eq. (5.1).

$$\begin{aligned}
&\forall_{res} R.\ (\\
&\quad (\quad \exists_{com} C.\ \exists_{\mathbb{R}} M.\ \exists_{vot} V.\ \exists_{\mathbb{R}} Y.\ (\\
&\qquad\qquad company_resolution(C, R)\\
&\qquad\quad \wedge\quad resolution_voting(R, V)\\
&\qquad\quad \wedge\quad company_majreq(C, M)\\
&\qquad\quad \wedge\quad voting_yesratio(V, Y)\\
&\qquad\quad \wedge\quad Y > M\\
&\qquad)\\
&\quad) \Rightarrow is_resolution_passed_via_voting(R)\\
&)
\end{aligned} \tag{5.1}$$

Experimental Set-Up Truncating the director application's whole TPTP program to a minimal version that is suitable for the experiment results in the program in Listing 5.6. It uses the predefined TPTP predicate `$greater`.

```
tff(t01, type, vot: $tType).
tff(t02, type, voting_yesratio: (vot * $real) > $o).
tff(t03, type, has_majority: vot > $o).
tff(t04, type, vot1: vot).

tff(a01, axiom,
  ! [V: vot] :
    ( has_majority(V)
      <=
      ( ? [YesRatio: $real] :
          ( voting_yesratio(V, YesRatio)
          & $greater(YesRatio, 0.5)
          )
      )
    )
).
tff(a02, axiom, voting_yesratio(vot1, 0.6)).

tff(tiq, conjecture, has_majority(vot1)).
```

Listing 5.6: `realcmp_valid.p`: TFF program for real number comparison

The conjecture is valid in the example. The validity can be toggled by changing the 0.6 in axiom `a02` to 0.4, for example.

Test Results The valid conjecture can be proved by all tools but Leo-III, which, on the contrary, states in its output that the symbol `$greater` is unknown or unsupported. Other than specified in [SS01b], arithmetic reasoning is apparently not or not fully supported by Leo-III.

Given the falsifiable conjecture, iProver, Princess, and Vampire time out. Only cvc5 directly recognizes it as not provable. So, disregarding Leo-III, we observe that arithmetic reasoning on a conjecture which is not valid seems to be a non-trivial task for three out of four provers.

Remedy If the case-related axioms are somehow generated automatically, then this generation process could be adjusted so that it precomputes the conclusions which are entailed from arithmetic axioms. In turn, the arithmetic-based implications are removed from the TPTP program. Indeed, this would

be a rather crude workaround for the issue as well as a significant restriction with respect to the formalization's expressiveness.

5.3.4 Predicate Completion & Quantifications

Practical Relevance In the course of the formalization, we justified the utilization of predicate completion. Furthermore, many rules make use of existential quantifications in order to navigate to related individuals. So, the provers are supposed to handle those two conventions in combination.

Experimental Set-Up Listing 5.7 displays an example program in which a predicate's extension is conclusively asserted via predicate completion. The conjecture is an existential quantification whose body is a predicate statement with one constant argument and one quantified variable argument. Since the quantified variable `C` can match to `com1` due to axiom `a01`, the conjecture is obviously semantically valid.

```
tff(t01, type, dir: $tType).
tff(t02, type, com: $tType).
tff(t03, type, director_company: (dir * com) > $o).
tff(t04, type, dir1: dir).
tff(t05, type, com1: com).

tff(a01, axiom,
  ! [D: dir, C: com] :
    ( director_company(D, C)
      <=>
      (D = dir1 & C = com1)
    )
).

tff(tiq, conjecture,
  ? [C: com] : (director_company(dir1, C))
).
```

Listing 5.7: `predcomplquant.p`: TFF test program for predicate completion in combination with an existential quantification

Test Results All provers but cvc5 are capable of conducting the proof. However, cvc5 is at least able to prove the quantification-free conjecture `director_company(dir1, com1)`. Interestingly, after having proved

this easier conjecture, the tool gains the capability to prove the conjecture from Listing 5.7. This can be reproduced by turning the conjecture into a conjunction with `director_company(dir1, com1)` as the other operand.

This observation can be explained as follows: During the proof procedure, several intermediate conclusions arise, are added as valid theorems to the knowledge base, and can therefore be further used equivalently to axioms. If the atomic statement `director_company(dir1, com1)` was in the axiom set, then the proof could be conducted by cvc5. So, by asking for the easier sub-conjecture, we pave a way that cannot be found by the prover without that assistance.

Remedy For every assertion which is made using predicate completion, one could additionally (and indeed, redundantly) assert the predicates with all the affected tuples of individuals again without predicate completion. The information content of the axiom set is not increased by that step. Though this workaround exacerbates the verbosity of the case-related axioms, it makes cvc5 cope with the existential quantifications.

Special Case: Default Values Particular attention must be paid to the handling of default values. Let us recall the axiom for the company's majority requirement from Eq. (4.21) in Section 4.5.1. The general predicate *company_majreq* receives the value in question from the specifying predicate *company_majreq_AoA* or subsidiarily from the default value in the disjunction's second operand. Both the general and the specifying predicate's extension is concluded via predicate completion.

If there is no stipulation via the specifying predicate, then cvc5 does not have the ability to derive from the default rule axiom that the extension of *company_majreq* comprises a tuple which consists of the company and the default value, as we just observed it for the predicate completion in general.

One could now apply the above-mentioned remedy, i.e. the explicit assertion of the affected combinations of individuals as atomic statements in the case-related axioms, in particular a *company_majreq* assertion with the default value. However, this would give rise to a redundancy which is not only limited to the case data but—recalling where the default value actually comes from—extends to the default rules, which is in contrast to the previous situation.

Given that the case-related axioms are automatically generated, then the generating code should definitely somehow extract the default value for the atomic re-assertions from the corresponding rule axioms. This would allow the code to avoid the redundant knowledge of the default value.

5.4 Runtime Duration

Having examined a handful of issues separately and isolatedly, the five tools are now applied to the implemented logic program for the director application. This experiment is performed on two case constellations. On the one hand, the case axioms entail the director application's validity. On the other hand, one necessary condition for the validity is left unfulfilled (e.g. the voting majority or the representation power). So, for each tool, the test is conducted with one valid and one falsifiable conjecture.

The axiom set of the implemented requirements check consists of

- 24 rule axioms,
- 5 inequality assertions for 7 static individuals,
- 27 assertions for relations and case facts (with predicate completion)
- 21 inequality assertions for 20 non-static individuals (for the running example whereas it differs for other cases).

Of course, since the implementation is done in TFF, the type attributions further enlarge the number of actual axioms.

If the remedies which were supposed above are conceded to the affected tools, then cvc5, iProver, and Vampire are able to return a result within less than a second. In contrast, both Leo-III and Princess do not terminate even if they are granted a timeout of ten minutes. This finding is indifferent to the conjecture's validity.

5.5 Summarized Tool Support

The findings obtained from the experiments are summarized in Table 5.1. None of the five tools can handle the implemented TFF program without limitations. Nevertheless, if we settle for the noticed limitations and alleviate them via the remedies, then cvc5, iProver, and Vampire are capable of inferring on the formalization. In contrast, Leo-III and Princess are not able to make a conclusion within a reasonable time.

In the event that a decision for one tool was necessary, then one would have to prioritize with respect to the acceptable cutback. If arithmetic-based axioms are indispensable but the predicate completion's verbosity in the case axioms is acceptable, then cvc5 is the best choice. If the precomputation workaround is admissible, then iProver and Vampire are viable options, too.

Of course, there exist a handful of more TPTP provers which we did not examine and, therefore, are a basis for further studies.

	cvc5	iProver	Leo-III	Princess	Vampire
Trivial (valid)	Can a valid conjecture be proved?				
	✓	✓	✓	✓	✓
Trivial (falsifiable)	Can a non-valid conjecture be recognized as non-valid?				
	✓	✓	✓	✗	✓
File Inclusion	Can all statements be outsourced to other files and, at the same time, the conjecture be an existential quantification?				
	✓	✗	✓	✓	✓
Arithmetic (valid)	Can a valid conjecture that is entailed by arithmetic propositions be proved?				
	✓	✓	✗	✓	✓
Arithmetic (falsifiable)	Can a non-valid conjecture that is entailed by arithmetic propositions be recognized as non-valid?				
	✓	✗	✗	✗	✗
Predicate Completion	Can the mere existence of appropriate individuals be derived from a predicate completion statement?				
	✗	✓	✓	✓	✓
Runtime	Is the logic program for the director application processed within a reasonable time?				
	✓	✓	✗	✗	✓

Table 5.1: The test results of the prover tools

5.6 Evaluation

Qualified Implementation Success The implementation using the infrastructure of TPTP was successful. Nevertheless, the success is reduced by the fact that none of the five considered prover tools are capable to process the logic programs instantly without further adjustments.

Rejected Prover Tools The provers cvc5, iProver, and Vampire turned out to be suitable for the formalized register process. In contrast, Leo-III and Princess must be rejected as not qualified.

Chapter 6

Implementation in Prolog

Having formalized and implemented the director application in pure first-order logic, we now aim at implementing the scenario using the logic programming language Prolog. When translating the first-order axioms into Prolog, we intend to maintain the described formalization conventions as these were established for good reasons. Nevertheless, we will call the conventions into question where appropriate.

6.1 General Equivalents in Prolog

In Section 3.2.4, we have ascertained that Prolog's expressiveness is in some measure limited compared to first-order logic. For this reason and though we translate from first-order logic to Prolog, we want to start off by taking the target perspective. From this, let us briefly recap Prolog's basic features and the reference to their respective equivalents in first-order logic.

The Prolog terms *atom* and *relation* are directly correlated to the *individual* and the *predicate* in first-order logic. The *fact* and *rule* clauses are equivalent to first-order *axioms*. The rules are Horn clause axioms (expressible as an implication) whereas the facts have the further restriction that they correspond to a simple atomic formula.

A striking syntactical difference can be observed when operating with variables. As already mentioned, the introduction of a Prolog variable would be translated to a wrapping universal quantification in first-order logic. This is the only method to perform some quantification since there exists no dedicated explicit expression for quantifications in Prolog.

6.2 Implementing the Formalization Conventions

6.2.1 Trivial Transformations

The translation of some formalization conventions is fairly trivial.

Law-Related vs. Case-Related Axioms Prolog's distinction between rules and facts is an appropriate means for the distinction between the law-related rule axioms and the case-related fact axioms. The rule axioms describe the general norms which apply under specific conditions whereas the case facts hold unconditionally.

Object Individualization The instantiation of objects which are mutually related can be adopted without restrictions. The instantiation is performed implicitly by just mentioning the individual.

Predicates for Case Facts & Relations The monadic and binary case fact predicates as well as the binary relation predicates are representable as Prolog relations with the corresponding arity.

6.2.2 Many-Sorted Logic

In Prolog, there exists no built-in support for sorts like in TFF. We have the option to simply translate the first-order axioms neglecting the sorts and therefore treating the TPTP axioms as if they were formulated in FOF. By doing this, we lose the benefits of many-sorted logic.

Therefore, we aim for maintaining the sorts in Prolog. As stated in Section 4.1.3, many-sorted logic can be implemented using unsorted first-order logic. So, we can emulate the sorts in Prolog.

We introduce a binary relation `is_of_sort`, whose first argument is some (sorted) individual and whose second argument is the information about the sort. This sort information can be one of the sorts in Eq. (4.6) except the rational number, which is a built-in type in Prolog.

In the Prolog clauses, we use the `is_of_sort` relation to map all individuals to their type. This is similar to the mandatory introduction of sorted individuals in TFF. In the rule clauses' body, `is_of_sort` is used once again to ensure that the rule only applies to individuals of a specific sort.

Regrettably, this workaround does not compensate the lack of interpreter-supported type checking, which facilitates mistakes in the implementation.

6.2.3 Rule Definitions

As stated in Section 3.2, rule clauses in Prolog are equivalent to universally quantified restrictedly structured implications in first-order logic, namely Horn clauses. The right side of the implication is an atomic statement whereas the left side is a conjunction of atomic statements.

Non-Leaf Rules This fits pretty well with the convention for validity rules with conjunctions from Section 4.3. If we consider the validity rules as initially formulated in Eq. (4.13), where all variables are universally quantified, then our first-order validity rules can be directly translated into Prolog. Including the sort emulation, the mentioned equation would be translated to the Prolog clause in Listing 6.1.

```
is_application_legal(A) :-
  is_of_sort(A, app), is_of_sort(R, res),
  is_of_sort(D, deed), is_of_sort(S, ass),

  is_resolution_legal(R),
  is_deed_legal(D),
  is_assurance_legal(S),
  are_applicants_authorized(A),

  application_resolution(A, R),
  resolution_deed(R, D),
  application_assurance(A, S).
```

Listing 6.1: Prolog clause of the root rule

Validity rules consisting of a disjunction can be translated to rule clauses which are split. For example, the requirement concerning the deed format (item 2 from Scheme 2.1) can be implemented as in Listing 6.2.

```
is_deed_legal(D) :-
  is_of_sort(D, deed),
  deed_format(D, original).

is_deed_legal(D) :-
  is_of_sort(D, deed),
  deed_format(D, certifiedcopy).
```

Listing 6.2: Prolog clauses for the validity check of the deed

Leaf Rules The leaf rules in the requirements check are free from any restriction. Therefore, the question whether their translation into Prolog is feasible cannot be answered in general but has to be examined individually. However, the limited expressiveness of Prolog compared to first-order logic suggests that restrictions are probable.

In the course of the practically performed translation of the law-related axioms, we encountered exactly one problem with leaf rules. Some items from Scheme 2.1, e.g. item 1.1.2.1, which we formalized in Eq. (4.16), require that all shareholders or directors of the company have a specific property. In proposition 1.1.2.1, it is the shareholders' consent to the resolution's subject. The first-order formulation in Eq. (4.16) is a nested universal quantification which cannot be expressed in Prolog and therefore marks one practical limitation for implementing the director application in Prolog.

However, as Prolog supports list representations and operations, we can implement a workaround in order to realize such a rule. Let us assume that we have the Prolog case-related fact clauses as in Listing 6.3.

```
company_resolution(com1, res1).
company_shareholderlist(com1, [alice, ben, chris]).
shareholder_consents(alice, res1).
shareholder_consents(ben, res1).
shareholder_consents(chris, res1).
```

Listing 6.3: Prolog case facts for the shareholders' consent to the resolution

The company `com1` is connected to the resolution `res1` via the relation `company_resolution` as well as to a list containing our three shareholders via `company_shareholderlist`. This latter predicate deviates from the approach in the first-order logic formalization, where each shareholder is individually connected to the company. Next, for every of those three shareholders, the consent to the resolution is asserted. In Listing 6.4, the list-based implementation of our rule for the consent of all company shareholders is written. For the purpose of conciseness, we omit the sort relations.

```
all_shareholders_consent(R) :-
  company_resolution(C, R),
  company_shareholderlist(C, ShList),
  all_shareholders_consent(R, ShList).
all_shareholders_consent(R, []).
all_shareholders_consent(R, [Sh|Others]) :-
  shareholder_consents(Sh, R),
  all_shareholders_consent(R, Others).
```

Listing 6.4: Prolog rules for the shareholders' consent to the resolution

The head of the first clause in Listing 6.4 is a monadic relation called `all_shareholders_consent`, whose argument is a resolution. The appropriate company as well as the list of shareholders are retrieved. The rule passes this list of shareholders to a binary relation with the same name.

This binary relation is the second and third clause's head. A statement `all_shareholders_consent` is trivially true if the passed list is empty. This makes sense at the commonsense level. If there is no shareholder in the list, then the consent is given by each of "them". The last clause checks the individual consent for the first shareholder in the list and passes the remaining shareholders to the same predicate call, which constitutes a recursion. This recursion finds its end as soon as every list item has been examined and the second clause has eventually been reached.

By applying that pattern, a proposition can be checked for each item in a list in Prolog.

6.2.4 Open-World Assumption

Predicate completion, which was necessary due to the open world in first-order logic, cannot be expressed in Prolog. This is because Prolog does not allow bi-implications, which are required for the predicate completion (see Section 4.4.1). However, since Prolog uses database semantics [RN09, p. 343] and, hence, there exists no open-world assumption in Prolog [Bra86, p. 135], predicate completion is not required and can be omitted.

The same goes for the conclusive sort domains. Introducing the individuals along with their corresponding sorts is sufficient in the closed world of Prolog.

6.2.5 Unique Name Assumption

Besides the closed-world assumption, there exists the unique name assumption in Prolog. This means that two atoms with different names are assumed to be different individuals. [RN09, p. 344] This makes the inequality declarations from Section 4.4.3 dispensable, too. Admittedly, the limited expressiveness would not have allowed such declarations anyway.

6.2.6 Default Values

The default value pattern, as exemplified in Eq. (4.21), is made up of a bi-implication and a not-exists expression, which seem to be problematic for the translation. Since the bi-implication is for predicate completion purposes, it can be simply ignored in the closed world of Prolog. The non-existence, on

the other hand, can be phrased using a special operator \+ in Prolog. The disjunction can simply be expressed by splitting the rule into one clause per disjunction operand. The two resulting clauses are written in Listing 6.5 and look even more intuitive than the first-order formula in Eq. (4.21).

```
company_majreq(C, X) :-
    is_of_sort(C, com),
    company_majreq_AoA(C, X).
company_majreq(C, X) :-
    is_of_sort(C, com),
    \+ company_majreq_AoA(C, _),
    X is 0.5.
```

Listing 6.5: Prolog rules for the majority requirement's default value

The first clause applies if there exists a special majority requirement in the company's AoA, which is asserted via the `company_majreq_AoA` relation. This special requirement is adopted for `company_majreq`. The second clause applies if there is no AoA specification. In this case, the default value 0.5 is taken.

6.2.7 Assumed Subconditions

The assumed subconditions from Section 4.5.2 are based on a positive existential quantification. Such a construct goes beyond the expressiveness of Prolog. There is only one obvious alternative solution for this situation. We have to deviate from the convention that real-world objects that do not evidently exist have no explicit formal representation as an individual, which we established in Section 4.2.1. Instead, some dummy individual is introduced, connected, and specified. The translated Prolog implementation of Eq. (4.22) consists of three fact clauses, as written in Listing 6.6.

```
is_of_sort(meet_dummy, meet).
resolution_meeting(res1, meet_dummy).
is_meeting_legal(meet_dummy).
```

Listing 6.6: Assumed subconditions using a dummy individual

6.3 Program Assembling

Similar to the TPTP implementation, the Prolog implementation's main logic program solely references a handful of files into which the knowledge base is divided. An imaginable division consists of

- `rules.pl` for the law-related rule clauses,
- `individuals.pl` for the `is_of_sort` assertions for both the static data and the case-related atoms,
- `casefacts.pl` for the case-related fact clauses, and
- `defaults.pl` for the default behavior.

Other than in TFF, there exist no mandatory type declarations and the atoms do not need to be introduced explicitly. We only do this for the purpose of the sort specifications in `individuals.pl`.

The Prolog program is displayed in Listing 6.7.

```
:- [rules].
:- [individuals].
:- [casefacts].
:- [defaults].
```

Listing 6.7: Prolog main program for the director application

The square brackets are a shorthand notation for Prolog's built-in predicate `consult`, which reads a program from a file. [Bra86, p. 152f.] Note that the file extension must not be stated.

6.4 Interpreter Invocation

As one can notice, the conjecture is not part of the logic program. Instead, the interpreter software SWI-Prolog first of all scans the logic program and can afterwards accept the queries.

SWI-Prolog can on the one hand be used as an interactive tool, to which one query after another is manually passed via the command line. On the other hand, the non-interactive mode is of course more interesting because it allows an automated invocation from another program. This can be done using one `-g` parameter with the "goal" conjecture and another `-g` with the value `halt`, which terminates the execution. [SWI] If the proof succeeds, then the execution finishes without any output. If the proof fails, the corresponding conjecture is printed as in Listing 6.8.

```
~$ swipl -s program.pl
    -g "is_application_legal(app1)" -g halt
ERROR: -g is_application_legal(app1):  false
```

Listing 6.8: Exemplary SWI-Prolog invocation with a negative proof result

Unlike in TPTP, the Prolog interpreter has no hard time executing the implemented director application. Arithmetic expressions are smoothly processed and the overall reasoning process always terminates without any noticeable problems regarding the execution duration.

6.5 Evaluation

Feasibility of Implementation On the whole, the implementation in Prolog turns out to be successful even though some concessions with respect to the compliance with the formalization conventions have to be made. Prolog's restriction to Horn clauses is compensated by Prolog's support of list operations and by the rules' structure, which is very close to Horn clauses in the first place.

Indirect Axiom Translation Nevertheless, this means that the conversion of the formalized first-order axioms to Prolog is not entirely trivial. Instead, it requires some special translation rules. If we wanted to automate the translation, those rules would need to be implemented.

No World-Closing All the detours for closing the first-order world are not necessary in Prolog due to its database semantics. This convenience gives rise to a more concise notation, to a less bloated axiom set, and to non-monotonic reasoning.

Lack of Sorts The conciseness which we gain from the implicitly closed world is a little impaired by the lack of sorts. This lack can be tackled by emulating the sorts via a relation `is_of_sort`, which must be used explicitly. Nevertheless, in contrast to TPTP, we lose the automated type checking by the interpreter so that mistakes during the implementation might occur more frequently.

Excellent Tool Support In contrast to the several issues that we had when utilizing TPTP, the reasoning execution via Prolog does not cause any striking problems. The runtime reliably terminates and also arithmetic operations and reasoning are well supported.

Chapter 7

Ontology Incorporation

Having accomplished the implementations, in this chapter, we will examine how ontological knowledge can be incorporated into the formalized process. Ontologies are a common means to represent domain-specific knowledge. Of course, this applies also to the legal domain, where formalized knowledge is crucial for implementations in the realm of AI & Law. [LDV20]

We will illustrate the ontology incorporation by no more than two examples. First, we deal with *LKIF Core*, which is a core ontology containing a manageable amount of concepts. After that, we briefly have a look at *LegalRuleML*, which is rather a markup language for legal knowledge.

The general approach that applies to both examples is to translate the ontology into our implementation language and, afterwards, to connect the translation with our implementation. For this chapter, we limit ourselves to the TPTP axioms and omit the Prolog-based approach.

7.1 Core Ontology: LKIF Core

7.1.1 Overview

LKIF Core is a legal core ontology which declares basic concepts on a very abstract level. The concepts originate from both legal and commonsense domains. [Hoe+07] Even though the ontology was developed in 2007 and has thus reached a considerable age, its great influence justifies choosing it as an example. For instance, its abstract concepts are referenced by more recent ontologies, like LOTED2 (2014)[1], the Data Protection Ontology (2015)[2], and

[1] https://code.google.com/archive/p/loted2/source

[2] https://bitbucket.org/guerret/lu.uni.eclipse.bpmn2/src/master/resources/dataprotection.owl

PrOnto (2018)[3]. [LDV20] LKIF Core was published on GitHub[4].

The approximately 200 concepts are divided into 13 basic modules, each of which contains a cluster of closely related terms. Eleven of those modules (all but "rules" and "modification") form the package "LKIF Core" as a first entry point. In contrast, the more comprehensive package "LKIF Extended" incorporates LKIF Core and expands it by adding concepts regarding the validity period of legal documents as well as a basis for legal argumentation. [Hoe07] For illustrative purposes, we limit ourselves to incorporating the core package.

The modules are provided as separate interrelated files, which are written using the *Web Ontology Language (OWL)*. OWL is a language for representing knowledge in the Semantic Web in a specific XML format. An OWL document can define classes, their properties, individuals, and relations between these. As a W3C recommendation, it is a standard approach to phrase ontological knowledge. [W3C04] Consequently, our approach to incorporate LKIF Core into our formalization can be analogously transferred to other OWL ontologies.

7.1.2 OWL to TPTP Translation

The semantics of OWL are compatible with the description logic $\mathcal{SROIQ}$. [W3C12], which can be considered as a decidable fragment of first-order logic. [Sik17, p. 81] It may therefore be concluded that translating an OWL ontology to first-order axioms can be performed without any restrictions or the risk of losing information. Theoretically, it is therefore possible to convert the LKIF Core ontology into the first-order language of our formalization.

In addition, our practical implementation with the TPTP infrastructure requires the ontology to be available in a syntax that is compatible to our TFF program. A Python-based software solution for the translation from OWL into TPTP called *FOWL* was developed by [Flü+21] and published on GitHub[5]. A small modification is necessary to render the tool suitable for our purposes because it is not directly capable to resolve dependencies across several files. The single steps for executing the translation can be found in the GitHub repository that is related to this thesis.

The result is a set of about 1000 axioms, which declare and interrelate the LKIF Core concepts. Two exemplary axioms concerning the concept "Decision" are displayed in Listing 7.1 (with added indentation and shortened identifiers for readability).

[3] no data publication

[4] https://github.com/RinkeHoekstra/lkif-core/

[5] https://github.com/gavel-tool/python-gavel-owl

```
fof(axiom295,axiom,(
 ![X0]:(
  'lkif_core_legal_action_owl_Decision'(X0)
  =>
  'lkif_core_legal_action_owl_Legal_Speech_Act'(X0)
 )
)).
fof(axiom804,axiom,(
 ![X0]:(
  'lkif_core_legal_action_owl_Decision'(X0)
  =>
  ?[X1]:(
   'lkif_core_action_owl_actor'(X0,X1)
 & 'lkif_core_legal_action_owl_Public_Body'(X1)
  )
 )
)).
```

Listing 7.1: Two exemplary LKIF Core axioms translated by FOWL

The depicted axioms represent on the one hand the subclass relation between the concepts "Decision" and "Legal Speech Act". On the other hand, they assert that any "Decision" item has an "actor" property, which refers to a "Public Body" individual.

All the axioms which are generated by FOWL can be stored in a file called `lkif-core.ax`, which constitutes an includable TPTP axiom file.

7.1.3 Ontology Inclusion

In this chapter, we exemplify the ontology's inclusion into the target TPTP program by linking the object that represents the director application's resolution with the LKIF Core concept "Decision".

Incompatibility of FOF and TFF Predicates As can be seen in Listing 7.1, the automatically generated axioms are phrased in the one-sorted FOF dialect using predicates whose signature is not explicitly specified. In variation from this manner, the realization of our formalization is in TFF syntax with predicates which are explicitly declared with regard to their arguments' types. In general, the TPTP provers allow to merge FOF axioms with TFF axioms. Nevertheless, an individual with an explicit type specification must not be passed into a predicate if the corresponding argument declaration is untyped (and vice versa).

The exemplary program in Listing 7.2 is the naive approach to state that every resolution individual falls under the concept of "Decision".

```
include('./lkif-core.ax').
tff(type_res, type, res: $tType).
tff(axiom_res_is_lkif_Decision, axiom,
  ! [X: res]: lkif_core_legal_action_owl_Decision(X)
).
```

Listing 7.2: Exemplary TPTP program which causes a type error

However, the typed variable `X` is passed to a predicate whose argument is not typed. The input program causes Vampire 4.7 to refuse the execution. Instead, it outputs the error message in Listing 7.3, which points to the unavailable substitutability between typed and untyped terms.

```
Failed to create predicate application for
   lkif_core_legal_action_owl_Decision of type ($i) >
    $o
The sort res of the intended term argument X0 (at
   index 0) is not an instance of sort $i
```

Listing 7.3: Error message from Vampire 4.7 given a type error

Indeed, the problem of interconnecting the FOF axioms and the TFF axioms is rather a question of implementation than of formalization. Let us have a look at the possibilities to solve this practical problem.

Target Program in FOF A trivial solution is translating the target program from TFF into FOF. The type specifications can either be emulated—like we did it in Prolog—or simply be discarded. In both cases, all individuals in the director application program would become untyped and could be passed into the LKIF Core predicates without any restrictions.

Since there are good reasons for the use of TFF instead of FOF, as already depicted, this approach is mentioned only for the sake of completeness.

Ontology in TFF: Naive Approach The reverse way is the conversion of the generated ontology axioms into TFF. This would enable the further usage of the target program with types. As the concept-related information is only available in OWL and not anymore in the FOWL output, an upcast of the LKIF Core FOF axioms to TFF is infeasible. So, if every LKIF Core concept should result in one type, then an alternative tool which converts OWL directly into TFF would be the way to go. A general solution, however, cannot be provided because the resulting TFF predicates must refer to

the individual target application's types, e.g. the resolution of the director appointment. Hence, this approach is not adequate either.

Ontology in TFF: Polymorphism An alternative possibility is to introduce a TFF type that becomes the overall base type of all so far unsorted terms in the ontology translation and of all types in the target program. In addition, the LKIF Core predicates are adjusted to expect arguments of the newly introduced base type. The predicates henceforth accept the individuals from the director application because the types of these are more specific than the required type. Of course, such an approach necessitates the support of a polymorphic mechanism for passing arguments. The syntax specification of TFF mentions an operator for a subtype relation. [SS01a] It therefore insinuates that one can define multi-level type hierarchies and exploit polymorphic predicate calling mechanisms. Unfortunately, none of the ATP tools which were introduced in Section 3.1.3 actually support that. So, this approach also fails in practice.

Linking Adapters Another approach leaves both the LKIF Core FOF axioms and the target TFF program unchanged and instead establishes an adapter architecture. For each typed individual in the target program, one untyped adapter "twin" individual is declared. As the twin is untyped, it can be passed vicariously to the automatically generated ontology predicates. Reconsidering the resolution example, such a set-up is displayed in Listing 7.4. Note that `$i` indicates the pseudo-sort for an unsorted individual.

```
tff(type_res, type, res: $tType).
tff(pred_lkiflink_res, type,
  lkiflink_res: (res * $i) > $o
).
tff(lkiflink_res_adapting, axiom,
  ! [X: res, Y] : (
    lkiflink_res(X, Y)
    =>
    lkif_core_legal_action_owl_Decision(Y)
  )
).
```

Listing 7.4: Adapter individual and predicate

The predicate `lkiflink_res` serves as a connector for a `res`-typed individual and its corresponding untyped adapter twin. The concrete state-

ment that connects `res1` and the adapter `lkiflink_adapter_res1` using the predicate `lkiflink_res` is written in Listing 7.5.

```
tff(inst_res1, type, res1: res).
tff(lkiflink_res1_adapter, axiom,
  lkiflink_res(res1, lkiflink_adapter_res1)
).
```

Listing 7.5: Connecting the sorted individual to its adapter

This approach has the advantage that no conversion between FOF and TFF is necessary. However, it is an unbeautiful workaround which brings in some kind of redundancy. Accessing the ontology axioms is also a little cumbersome because the detour using the adapter has to be taken.

7.1.4 LKIF Core Evaluation

As already mentioned, the incorporation of LKIF Core into the TFF program of the director application can analogously be transferred to any OWL ontology. This is convenient because OWL is a pretty popular ontology language. So, more comprehensive ontologies with valuable domain knowledge can be included into the legal reasoning process.

The obstacles in the practical realization of the ontology incorporation are admittedly somewhat sobering. It is to be hoped that our polymorphic approach will gain relevance if the established TPTP tools support multi-level inheritance someday.

7.2 Rule Markup Language: LegalRuleML

7.2.1 Overview

We want to pay attention to another ontology-like technology called *LegalRuleML*, which was published in [Pal+11]. Actually, LegalRuleML is not an ontology in the narrow sense but an XML markup language, which provides a standard for the machine-readable representation and exchange of legal knowledge. [LDV20, p. 11] Considering its strong expressiveness, it is supposed to model normative rules in a sufficiently suited manner for the requirements in the legal domain. As an extension of *RuleML*, which is a markup language for Semantic Web rules, it avoids reinventing the wheel but reuses an established technology to build the formalized rules. [Pal+11]

A LegalRuleML document consists of three main parts. The *metadata* section contains references to the original legal and non-legal source docu-

ments, details about the geographic and functional area of application, the norms' temporal validity period, and information about their provenance and authorship.

In the *context* section, the metadata, especially the sources, and the modeled rules, are connected with each other. Such connections are of particular significance if a norm can be legally interpreted in different ways and therefore results in different sets of formalized rules. As a side note, this example of a feature demonstrates how extensive LegalRuleML's expressiveness actually is. Relations between elements in the LegalRuleML document are made via *Internationalized Resource Identifiers (IRIs)*. In the example files from LegalRuleML's web page (see [Pal+21]), those identifiers are also used for external references, e.g. to legal norms.

The last and usually most substantial part of the document are the *statements*. Legal rules which consist of preconditions and some conclusion can be phrased as so-called *legal statements*, which are distinguished between *constitutive statements* and *prescriptive statements*. While constitutive statements serve to just define concepts, prescriptive statements have a normative character. This is reflected in the syntax by the fact that a prescriptive statement's conclusion—other than in a constitutive statement—can be a deontic expression, like an *obligation*, a *permission*, a *prohibition*, or a *right*. Aside from those legal statements, there exist a couple of more types. A sanction can be stated using a *penalty statement*, which is in turn connected to a violable prescriptive statement via a *reparation statement*. In a self-explanatory way, a *factual statement* enables the assertion of case-related data.

Another interesting feature is the *override statement*, which makes a relation between two legal statements in view of the fact that one takes precedence over the other. This behavior is related to the term *defeasibility*. Defeasible reasoning allows the existence of two contradictory conclusions without giving rise to *ex falso quodlibet* like in classical logic. Instead, the conflict is resolved via the preference relation of two rules. [Pal+21]

7.2.2 Conceivable Use

After this very rough overview of the extensive possibilities offered by LegalRuleML, let us reflect on a conceivable use of this technology. As we did not practically realize any linking between LegalRuleML and our TPTP programs in the course of this thesis, the following paragraphs should only be interpreted as theoretical considerations and motivations.

Our formalization of the legal regulations, more precisely the ones from Scheme 2.1, was done for the purposes of this thesis. The scope of all predicates for both the legal regulations and the case facts is limited to this

thesis as well. Consequently, all legal formalizations which are created in this manner are self-contained units and cannot be interconnected directly. If we want to perform some kind of interconnection with another formalization, then an explicit harmonization is necessary to state which elements are mutually related or correspond to the same real-world entity.

In the Semantic Web, entities are generally identified by a *Universal Resource Identifier (URI)*. [BHL01] Two entities can be recognized as same or different according to their identifier. So, if two documents use those identifiers to specify the objects and relations, then they can be combined more smoothly. As LegalRuleML claims to comply with the Semantic Web technologies [Pal+11, p. 2] and also uses those URIs (or more precisely IRIs), it is a very interesting approach for phrasing machine-readable legal norms considering that the resulting artifact becomes a participant in the Semantic Web.

Another conceivable advantage would arise from a database with several LegalRuleML documents. One legal norm is usually relevant for more than only one use case. So, a LegalRuleML database would supply legal knowledge which can be reused by several use cases.

7.2.3 Automated Reasoning

LegalRuleML was published as a markup language without any reasoning engine or the like. However, there exist at least two publications which intend to enable automated reasoning based on LegalRuleML documents and are worth being briefly mentioned.

[SF22] provides a bridge between LegalRuleML and TPTP by proposing a fragmentary translation logic. Focusing on TPTP, the main technology we utilized as the target language, the paper offers a very promising approach for some future research beyond this thesis. By the way, TPTP supports deontic operators, whose concrete use is a result of the translation from LegalRuleML into TPTP. It must be noted that deontic operators indeed go beyond the TPTP features which we employed in the course of this thesis.

The second noteworthy publication is [LH18], which addresses the transformation between LegalRuleML and Defeasible Logic (DL). Other than classical logic [RN09, p. 251], DL implies non-monotonic reasoning and is therefore not straightaway compatible with our first-order formalization. Instead, the publication utilizes a DL reasoning tool called *SPINdle*. Even though DL is a quite different approach to our formalization, it is still worth mentioning in the context of LegalRuleML.

7.3 Evaluation

Knowledge Reuse Incorporating ontologies into a set of legal axioms is an important topic in general. It enables to outsource, concentrate, and release knowledge in a way that many applications can benefit from the particular knowledge.

Minor Practical Issues The LKIF Core incorporation suffers a little from practical issues. Of course, being on the implementation level, the issues do not derogate the general idea of using ontologies and can be considered as limited to the TFF language.

Participation in the Semantic Web Examining LegalRuleML encourages to formalize knowledge and rules in a way that these can become a part of the Semantic Web. The unique identification via URIs allows a smooth compatibility with other resources which participate in the Semantic Web.

Chapter 8

Reasoning Visualization

8.1 Motivation

The great advantage of symbolic AI, in contrast to statistical AI, is the fact that conclusions can be reconstructed without any uncertainty. [Gru18]

Now as the director application has been implemented and prover tools can decide about its validity, there remains one open question: In the event that the validity of the application cannot be proved, what is the reason for this failure? This question is the motivating factor of this chapter.

The application's validity depends on the fulfillment of its nested subconditions, as depicted in Scheme 2.1. First and foremost, the provers output a simple yes or no concerning the validity. However, not only does this one final conclusion for a specific case matter, but it is also of special interest which particular item in the examination scheme does not hold and consequently negates the whole decision. In order to supply this need, we will develop a tool that makes the reasoning traceable and identifies the failing point.

Our presented approach will provide its result in a machine-readable structure for a further automated processing. In addition, we will display the result in a human-readable form. This is why we refer to the program that is to be developed as the *reasoning visualization*.

In Chapter 3, we asserted that both TPTP and Prolog allow an unambiguous reproduction of the way that led to a proof. However, our approach to make the reasoning traceable is not based on those reproduction possibilities. We will justify this in the next section.

In the following, we will examine the single steps of the program and the way how a comprehensible outcome is constructed for some use case. The explanations are primarily based on the TPTP implementation, but, of course, they can be analogously transferred to the Prolog implementation.

8.2 General Approach

As just mentioned, both TPTP and Prolog allow the user to reproduce how a proof was created. The TPTP provers support an execution mode which outputs for a successful proof all the applied first-order inference rules as well as the involved axioms and intermediate theorems. In contrast, Prolog has a debugging mode which allows running step by step through the backtracking process. [Bra86, p. 187]

By applying these proof reproduction features, an end-user-addressed proof demonstration could be implemented, at least in theory. However, the transformation of the chain of proof into a user-friendly output appears to be a non-trivial undertaking. Moreover, such an extensive task would go far beyond the scope of this thesis. Especially in the case of TPTP, there exists also the drawback that, understandably, the proof steps are only provided for successful proofs whereas there does not exist such an output in the case of a non-provable conjecture.

That is why we decided to neglect those reproduction features for illustrating the proof. Instead, we go an easier way. Scheme 2.1 and its itemization into a nested tree of subordinate prerequisites is a comprehensible illustration for legal experts, logicians, and also for laypersons. Similar to the construction of our formalized rules from the scheme, we can walk the reverse path and recursively subdivide the main conjecture—the question whether the application is legal—in a way that we obtain a tree-like structure consisting of sub-conjectures. This proof tree resembles our given examination scheme. One can now execute the proof check for the single tree nodes. Thus, for each tree node, there exists the information whether the corresponding subcondition is provable on its own or not. This enables a comprehensible illustration for both successful and non-performable proofs. If the conditions for the director application's legality are not satisfied, one can easily identify the deficient element of the current case.

Admittedly, the presented approach has a flaw by design because it entails redundant proof checks. Proving some proposition involves that the ATP tool is implicitly concerned with subordinate propositions that imply the considered one. As all tree items are recursively checked, those subordinate propositions are proved once again in the deeper levels of the recursion, which causes the redundancy.

8.3 Case Data File

Before we look into the structure and behavior of the program, we introduce a general machine-readable data format in which all the data of a single director application case can be represented. This is necessary because the program is intended to work for different director application cases and therefore needs a standardized format in which the case data is written. A file in our format can be considered as an input for the desired program.

We decided to define a JSON-based format whose structure is specified by a JSON Schema[1]. Unlike a space-saving binary format, a JSON file is human-readable and its structure is subsequently modifiable. Compared to XML, the schema definition is easier and the deserialization is mostly straightforward in many programming languages.

The input file provides a means of storing the case details in a compact way. The case-related axioms themselves are relatively chaotic and scattered and therefore not suitable for being a standard format. Instead, they will be a product automatically generated from the case data file (see Section 8.4.5).

This leads us to another advantage. The JSON file is independent from the language in which the axioms are formulated. It is usable as a source for generating both TPTP and Prolog code. Moreover, the data format can serve as an interchange format for processes which supply the case data in advance. Being readable by both humans and machines, JSON can also be both written manually by a person and generated by some automated process. The automated acquisition of case data for our application process is briefly discussed in Chapter 10.

The JSON structure for the director application is inspired by the arrangement in Fig. 2.2, which displays the involved objects. Roughly, the JSON file is divided into details about the *company*, the *resolution*, and the *application*. An example for a case data file for our running example application is printed in Appendix B.

8.4 Program Components

In the following, the functionality of the desired program is described. As we actually implemented it only for TPTP and its dialect TFF, the explanations are limited to the formalization using TFF. Nevertheless, the insights can be transferred analogously to Prolog and other TPTP dialects.

[1] https://json-schema.org/

8.4.1 Overview

The program consists of several steps, as depicted in the UML activity diagram in Fig. 8.1.

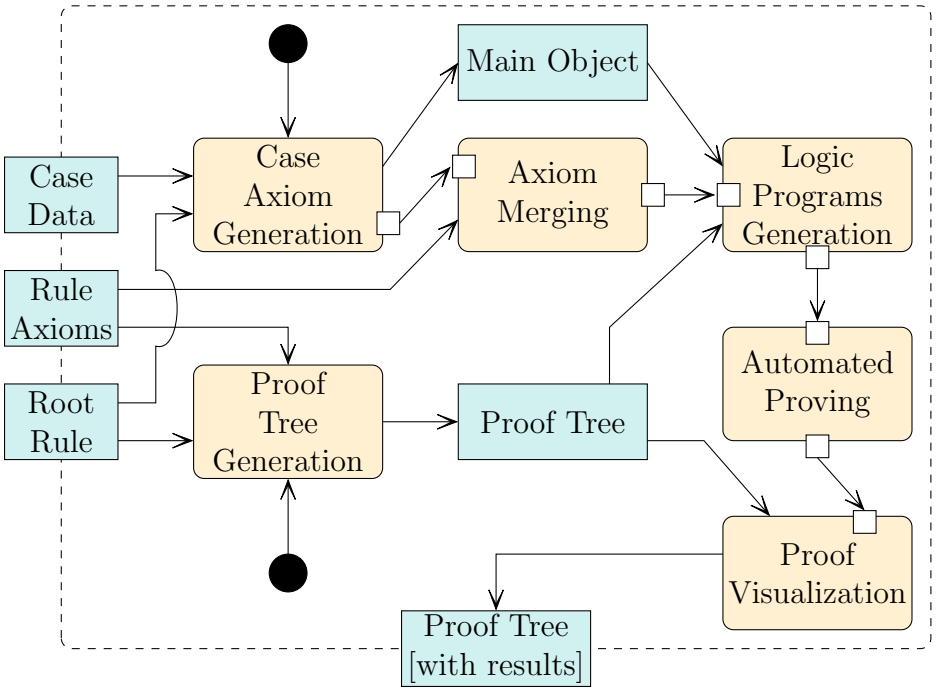

Figure 8.1: Activity diagram for the reasoning visualization

The inputs are a set of universal rule axioms, an individual director application case, and the rule which is checked. The case data is available in the established JSON format, from which the convention-compliant case-related axioms are automatically generated (see Section 8.4.2). The rule axioms and the case-related axioms are merged to the complete TPTP axiom set for the considered case (see Section 8.4.3).

Independently from these first steps, a tree-shaped proof scheme is recursively computed beginning from the root rule (see Section 8.4.4). In the case of our director application, the root rule is *is_application_legal*. For every item in the proof tree, a logic program with an individual conjecture and the previously assembled axiom set is created (see Section 8.4.5) and passed to the ATP tool (see Section 8.4.6).

The final step is described in Section 8.4.7. The particular results which are returned from the prover tool—positive or negative—are attached to the corresponding proof tree item. This enriched tree is visualized to the user. Its nodes are highlighted depending on the corresponding ATP result.

8.4.2 Case Axiom Generation

The *case axiom generation* is a component whose input are the case data in the established format and the root rule. It returns two outputs: The first result is the case-related axiom set as plain text. Depending on the language for which the component is aimed, the resulting axioms are either in TPTP or in Prolog. The second result is the name of the individual that corresponds to the root rule. In the case of the director application, where the root rule is *is_application_legal*, it is the name of the individual that represents the application.

As the input case data file is in JSON format, the deserialization into a corresponding nested object to be further processed is usually trivial in every programming environment. The object properties and their subordinate data are worked through and the corresponding axioms are generated according to the formalization conventions from Chapter 4. The axioms consist of assertions concerning the relation between the individuals on the one hand and of assertions concerning the case facts on the other hand. In the case of TPTP and its open world, the axioms are followed by the inequality declarations (see Section 4.4.3). Moreover, as already mentioned in Chapter 5, the many-sorted TPTP dialect TFF requires the incipiently expressed explicit introduction of all typed individuals. Following these guidelines, the concrete implementation is a straightforward task.

Every individual receives a name which is unique within the case axiom file. An important role is played by the name of the individual that corresponds to the root rule. This name identifies the "main object"—in our scenario the individual representing the application. The main object's name will later be relevant for the generation of the single logic programs.

The implementation of the case axiom generation is not only individual for the output language but also for the considered application type and thus the corresponding case data format. This entails that a separate case axiom generation has to be developed for a scenario other than the director application.

8.4.3 Axiom Merging

The generated case axioms and the static rule axioms together constitute the whole axiom set for the director application case and, therefore, must now be assembled somehow. Like in the Chapters 5 and 6, a logic program that includes the axiom files is automatically generated. Up to now, this logic program does not yet contain any conjecture.

8.4.4 Proof Tree Generation

The *proof tree generation* component takes as input the rule axioms and the name of the root rule. It outputs a tree-shaped proof scheme, which is a vital intermediate result of the tool. This proof tree correlates to the general rules and is still independent from any concrete case.

Tree Structure An exemplary item of the proof tree is printed as JSON in Listing 8.1 (with truncated nesting). It is an object which consists of the four properties *label*, *proposition*, *necessary*, and *children*.

```
{
  "label": "is_application_legal",
  "proposition":
    "? [A: app] : (
         is_application_legal(A)
     )",
  "necessary": true,
  "children": [ ...  ]
}
```

Listing 8.1: Exemplary JSON structure of a TPTP proof tree item

label contains the item's readable caption, which is printed when being visualized somehow. The *proposition* is a TPTP proposition that corresponds to the respective subcondition in the scheme. It generally consists of an existential quantification, which will be explained in more detail below.

The *necessary* flag indicates whether the subcondition's validity is indispensable for the validity of its parent condition. It is true for subconditions that are arranged in a conjunction with their siblings. Subconditions in a disjunction are not essential for the parent's validity so that the flag is not set. The purpose of that boolean property will be apparent later during the visualization.

The property *children* is a collection which contains other proof scheme items and therefore constitutes the recursive tree structure.

Axioms Text Parsing At the beginning, the rule axioms are available in TPTP syntax and therefore in textual form. For being processible, the text must be parsed into a structured object. We can use Leo-III's parser[2] for this purpose.

[2] https://github.com/leoprover/tptp-parser

Rule Axiom Examination A necessary subcomponent for the proof tree generation is the examination of potential rule axioms. Its input is some (deserialized) TPTP axiom statement. This statement is examined in terms of whether it is a rule axiom and if it complies with the convention from Section 4.3. If this is the case, then the subcomponent outputs some rule-related metadata which has been collected during the compliance check.

The object which contains this metadata is depicted as a UML class *RuleAxiom* in Fig. 8.2. If the axiom deviates from the rigid convention, then it is apparently a leaf rule or some other kind of axiom and cannot be decomposed according to Fig. 8.2.

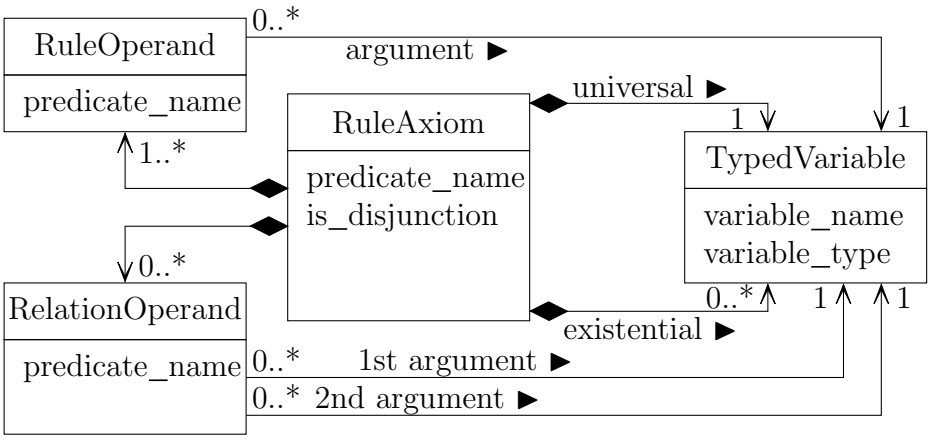

Figure 8.2: Class diagram of a rule axiom and its constituents

As stated in Section 4.3, the axiom formula is a universal quantification with only one variable, which corresponds to the 1-to-1-association between *RuleAxiom* and *TypedVariable*. The former's attribute *predicate_name* comes from the rule head, i.e. the implication's right side. The rule body, however, must be a conjunction, a disjunction, or a conjunction inside an existential quantification. This constitutes the childrens' *is_disjunction* attribute as well as the 0-to-*-association between *RuleAxiom* and *TypedVariable*.

The conjunction's or disjunction's operands can be divided into two categories:

- *Rule operands*: a unary predicate for which there exists a rule axiom. The predicate's single argument is either the universal variable or one of the existential variables.
- *Relation operands*: a binary relation predicate whose arguments are respectively one of the just mentioned variables.

If an operand is neither of them, then the whole axiom is not a compliant rule and not representable as a *RuleAxiom* object.

When analyzing the rule formula, special attention must be paid to one particular factor. Since the TPTP parser of Leo-III handles logical connectives as a binary operation, conjunctions or disjunctions with more than two operands are not deserialized as a flat list of operands. Instead, the parsed expression is represented as a binary tree with a strong imbalance to the right. In order to deal with all operands uniformly, a recursive function must be implemented for flattening the binary tree to a list.

Tree Construction The actual tree construction is performed as depicted in Fig. 8.3. We assume that the just described rule axiom examination has already been executed for the whole set of law-related axioms and that the resulting *RuleAxiom* objects are available in a dictionary as context information. This dictionary is not explicitly mentioned in Fig. 8.3.

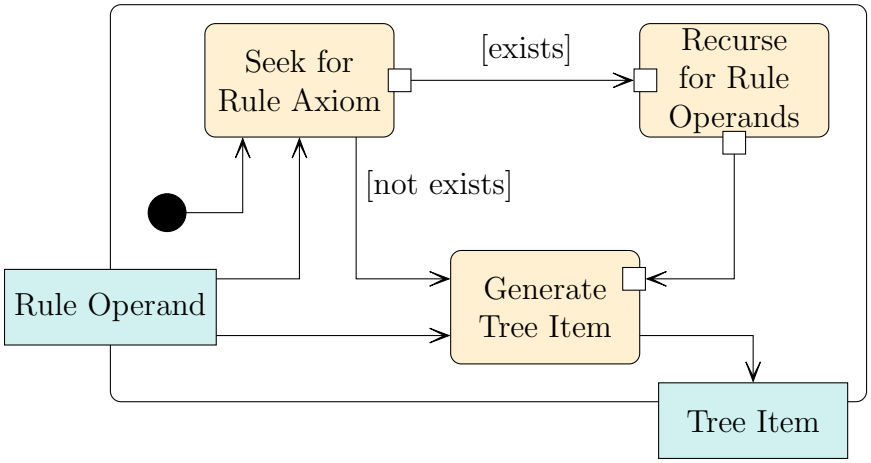

Figure 8.3: Activity diagram for the tree construction

The input of the tree construction is some rule operand. Note that, for this purpose, the root conjecture is treated as an operand even though it actually has no parent rule. First of all, it is checked whether the dictionary of rule axioms contains an entry that corresponds to the input rule operand. If this is the case, then the subordinate rule operands of the found rule axiom are passed to a recursive tree construction call. In this way, the recursive tree structure is established. The termination condition is fulfilled if there exists no convention-compliant rule axiom for the considered rule operand.

The finally generated tree item is structured like in Listing 8.1. The label is simply set to the rule operand's predicate name. The proposition is

built as explained below in "Conjecture Construction". The necessary flag is set depending on the *is_disjunction* attribute of the parent rule's metadata. Eventually, the children are filled with the results from the recursive calls.

As another side note, the currently handled *RuleAxiom* object is pushed onto a context stack before the recursive call and removed afterwards. This allows the tree generation to access the parent rule's metadata, e.g. for determining the *is_disjunction* value.

Conjecture Construction The subcomponent that generates the conjecture for a proof tree item will be sketched only very roughly. We exemplify it using the proposition 1.1.1 in Scheme 2.1. The corresponding rule predicate and the ancestors' rule predicates along with their respective argument sort are displayed in Scheme 8.1.

root	*app*	*is_application_legal*
1.	*res*	*is_resolution_legal*
1.1.	*res*	*is_resolution_formally_legal*
1.1.1.	*res*	*is_resolution_with_meeting_formally_legal*
1.1.1.	*meet*	*is_meeting_legal*

Scheme 8.1: Rule predicate for item 1.1.1 and its ancestors

If the individual for the application, i.e. the main object, is called $app1$, then the root conjecture can be expressed as in Eq. (8.1).

$$\exists_{app} A.\ (is\_application\_legal(A) \wedge A = app1) \tag{8.1}$$

The seemingly cumbersome formulation with its existential quantification separates the individual-specific part ($app1$) from the general part of the conjecture. This has the advantage that our conjecture construction can be general and independent from any connection to a concrete object. Instead, the concretization will be performed subsequently as soon as the case axiom generation from Section 8.4.2 has chosen a name for the main object and the logic programs have been built (as described in Section 8.4.5).

For the scheme proposition 1.1.1, the general individual-independent conjecture is shown in Eq. (8.2).

$$\begin{aligned} &\exists_{meet} M.\ \exists_{res} R.\ \exists_{app} A.\ (\\ &\quad is\_meeting\_legal(M) \\ &\wedge\ resolution\_meeting(R, M) \\ &\wedge\ application\_resolution(A, R) \\ &) \end{aligned} \tag{8.2}$$

It consists of the associated proposition itself as well as two relation predicates that lead from the meeting variable M over the resolution variable R to the variable A, which corresponds to the main object. This path can also be read from Scheme 8.1 by reading the predicates' argument types from bottom to top (*meet*, *res*, *app*).

The procedure which generates this path starts from the considered proposition within the parent rule formula. From there it walks rule by rule along the relation operands towards the root rule. The universally quantified variables of every rule on the way are regarded as intermediate targets. Having reached the universally quantified variable, the transition to the parent rule is performed with the support of the mentioned context stack.

This path-finding procedure can be implemented with a recursive search algorithm. As already mentioned, note that the constructed conjecture does not contain any reference to a concrete individual up to now.

8.4.5 Logic Programs Generation

Now that we have the entire axiom set, the name of the main object, and the general proof tree with its conjectures, we can continue the journey towards the automated execution of the proofs.

Every conjecture in the proof tree has a similar form as in Eq. (8.2). It is an existential quantification whose body is a conjunction. So far, the conjecture is general and has no connection to an individual. To establish the connection to the concrete main object, the conjunction is extended by another operand which is an equality check between the last existential variable and the main object. Given *app*1 is the main object, Eq. (8.2) would be transformed into Eq. (8.3).

$$\begin{aligned}
&\exists_{meet} M.\ \exists_{res} R.\ \exists_{app} A.\ (\\
&\quad\ \ is_meeting_legal(M)\\
&\ \wedge\ \ resolution_meeting(R, M)\\
&\ \wedge\ \ application_resolution(A, R)\\
&\ \wedge\ \ A = app1\\
&)
\end{aligned} \tag{8.3}$$

Eventually, the logic program without conjecture from Section 8.4.3 is taken as a pattern and equipped with the completed conjectures. In the case of TPTP, a statement with the type "conjecture" is appended to the end of the program, which in turn consists only of include statements.

For every node in the proof tree, there exists now one logic program whose conjecture asks for whether the corresponding subcondition of the scheme is valid on its own.

8.4.6 Prover Invocation

In the next step, all generated logic programs in the proof tree are given to the prover tool for being checked. As the single programs are independent from each other, the proofs can be performed in parallel.

In the Chapters 5 and 6, we already showed how the TPTP tools as well as SWI-Prolog can be invoked via the command line and how the printed data can be interpreted. This output is now parsed into an object with a fixed structure that can be processed further.

8.4.7 Proof Visualization

After the execution of the ATP tools, there exists now one result for every proof tree item. Disregarding the details of the concrete proof output, for simplification purposes, the result can simply be classified as one of the three categories *positive* (proof conducted), *negative* (no proof found), and *failed* (e.g. due to a timeout). This result is attached to the corresponding item in the proof tree. This enriched proof tree can now be visualized to the user.

```
~$ python3 test.py --process A96 --prover cvc5
   --reassert_predicate_completion ./example.json
✗ is_application_legal?
  ✗ is_resolution_legal?
    ✓ is_resolution_formally_legal?
      ✓ is_resolution_with_meeting_formally_legal?
        ✓ is_meeting_legal?
          ✗ is_personal_meeting_legal?
          ✓ is_teleconference_meeting_legal?
      ✗ is_resolution_without_meeting_formally_legal?
    ✗ has_resolution_majority?
      ✗ do_all_shareholders_consent_to_the_determi...
      ✗ is_resolution_passed_via_voting?
  ✓ is_deed_legal?
  ✓ is_assurance_legal?
  ✓ are_applicants_authorized?
    ✗ are_applicants_authorized_via_general_sole_r...
    ✗ are_applicants_authorized_via_individual_sol...
    ✗ are_applicants_authorized_via_general_modifi...
    ✓ are_applicants_authorized_via_individual_mod...
    ✗ are_all_directors_present?
```

Listing 8.2: Proof visualization for our example case

Listing 8.2 shows an imaginable proof visualization of our running example. In order to provoke some coloring, we modified the example so that Chris abstains from the voting, which derogates the resolution's majority.

The items' labels are printed line by line performing a depth-first traversal through the tree. The hierarchy is visualized using a depth-dependent indentation. The individual proof results are indicated with an appropriate text color, background color, and a preceding icon.

The striking red highlighting for a non-provided proof is only activated if the *necessary* flag is set. Since a non-valid operand of a disjunction does not necessarily cause the parent proposition to fail, it would be inappropriate to draw the attention to the operand using a red coloring.

All in all, the visualization allows an end-user to easily recognize the cause of the failure. It can be seen that the depth of the tree is limited so that only convention-compliant rules and their immediate descendants are displayed.

8.5 Evaluation

Traceability of Reasoning Examination schemes like Scheme 2.1 are a frequently utilized means for requirements checks by jurists and therefore provide a "familiar sight". As the visualization output is strongly inspired by the scheme, it is easily understandable for jurists. The remarkable highlighting of a failure-causing item contributes to the comprehensibility because it allows to immediately identify the conjecture's point of failure.

Redundant Proofs Of course, one downside of the presented approach is the redundancy of proof executions. In the case of a failing proof, we trade this additional computation effort for the information where exactly the crucial non-holding subcondition in the scheme is. In the director application's requirements check, the redundancy did not cause a perceivable loss of performance. Admittedly, this statement cannot be made universally for all axiom sets.

Reusability for Other Processes Even though the visualization was developed based on the formalized rule set, the general approach can be reused for any other scenario. Of course, the axioms must comply with the formalization conventions, especially with the one for the rule definitions. Furthermore, besides the rule axioms, an individual case data file format as well as a corresponding case axiom generation have to be implemented because the new scenario involves its own characteristic data.

Chapter 9

Adaption to Other Processes

The goal of this thesis is to make a universal statement about the suitability of logic for the formalization and automation of register processes in general. Until now, we have obtained several important insights using the example of the director application process. However, the question remains open whether these insights are not restricted to the considered register process but can be regarded as universal and are hence appropriate to make general conclusions.

Having limited ourselves to the director appointment so far, we will now apply our developments to two other application processes from [GBM22], namely A115 (*Company Dissolution by Resolution*) and A108 (*Capital Increase by Contribution*). A successful adaption would substantiate the generality of our insights.

Unlike the director appointment in Chapter 2, the two processes and their prerequisites will be discussed in a little less detail, but we will throw a coarse glance at the application's requirements and by all means provide the examination schemes on which the formalization is based.

9.1 Company Dissolution by Resolution

9.1.1 Legal Description

The process A115 from [GBM22] is the dissolution of an LLC. It has a slightly higher complexity than the director appointment. This is because it is not only concerned with the directors, who perform the application, but also with the liquidators, who responsibly execute the company's liquidation.

The liquidation is entailed by the dissolution resolution, which is made by the shareholders with a majority of three quarters of the votes cast. This

requirement may be tightened by the AoA. The resolution does not directly terminate the company but changes its intended purpose to liquidating and winding up the company.

§§ 60–67 GmbHG specify the requirements for the mandatory application for entry in the Commercial Register, as depicted in Scheme 9.1.

The application is legal.

1. The resolution is made legally.
   - 1.1. The resolution is formally legal. (see item 1.1 in Scheme 2.1)
   - 1.2. The resolution is made with a qualified majority. (see item 1.2 in Scheme 2.1)
   - 1.3. The resolution is notarized if the AoA are amended.
2. The liquidators are determined.
   - case 2.1. The liquidators are determined in the AoA.
   - case 2.2. The liquidators are determined in the resolution.
   - case 2.3. The liquidators are the directors (default case).
3. The deed for the appointment of the liquidators is attached. (see item 2 from Scheme 2.1)
4. The assurances of the liquidators are signed.
5. The applicants are authorized to represent the company.
   - case 5.1. The resolution entails an AoA amendment.
     - 5.1.1. Directors are applicants. (see item 4 in Scheme 2.1)
   - case 5.2. The resolution entails no AoA amendment.
     - 5.2.1. Liquidators are applicants. (see item 4 in Scheme 2.1)

Scheme 9.1: Requirements for the dissolution's application

Obviously, several requirements are quite similar to the application of the director appointment. For some of them, e.g. item 1.1 or 5.1.1, we did not further unfold them for the sake of clarity but refer to the director appointment's Scheme 2.1 instead.

Particular attention must be given to the rules which address the liquidators. Item 2 in Scheme 9.1 enumerates the different possible ways how the

liquidators are determined. Moreover, the persons who are responsible for the submission of the application—either the directors (item 5.1) or the liquidators (item 5.2)—are determined by whether the dissolution involves an amendment of the AoA. Such an amendment is necessary if the AoA specify the company's expiration date, which must be annulled in the event of a resolution-induced company dissolution.

9.1.2 Formalization

Without regard to the immediately reusable axioms from the director appointment, the formalization of the liquidator-related case distinctions under the items 2 and 5 in Scheme 9.1 is mentionable. The identification of the liquidators is conducted via a construct that is based on the default value pattern from Section 4.5.1. An individual of sort *liql* (list of liquidators) with relations to the single liquidators is optionally linked to the company (item 2.1) or to the resolution (item 2.2). If neither of the two cases applies, then the existence of the list is just assumed and its relation to the liquidators is adapted from the company's directors (item 2.3).

Fig. 9.1 illustrates connections with respect to how the extensions of the liquidator-specific predicates influence each other. A high level of intricacy can especially be noticed in the rule for *application_responsibleapplicant*, whose domain is $\mathcal{D}_\iota^{(app)} \times \mathcal{D}_\iota^{(dir)}$—assuming that the liquidators and directors are represented by the same sort *dir*.

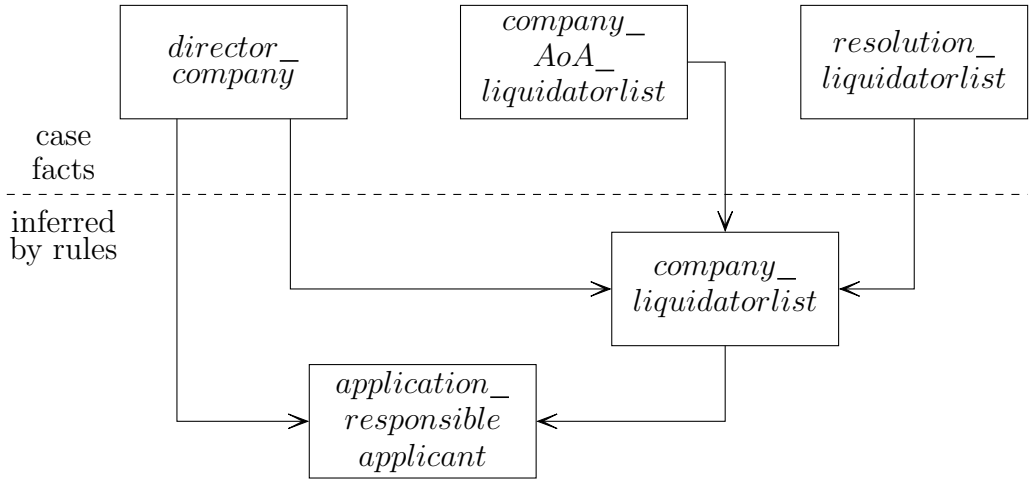

Figure 9.1: Determination of the predicates' extensions for the company dissolution

9.1.3 Implementation in TPTP

While we attached the first-order formula for the just mentioned rule in Appendix C, its TPTP implementation can be looked up in the dedicated GitHub repository if required. Having said that, the implementation of the whole company dissolution process consists of 28 TPTP rule axioms, which is only slightly larger than the 24 axioms of the director appointment.

Despite this seemingly small difference, the complex regulations for the liquidators give rise to practical problems. Passing the implemented TPTP program to cvc5, iProver, and Vampire—the tools which were capable of handling the director appointment—delivers the insight that both iProver and Vampire are to some degree overburdened with the logic program's complicacy. This might emerge from universal quantifications and the recurrent necessity to conclusively declare a predicate's extension via bi-implications.

Interestingly, the *semi-decidability*, which is a characteristic of first-order logic [Ert21], can be clearly perceived in the behavior of the ATP tools. Valid propositions—for both the main conjecture and the sub-conjectures in the proof tree—are proved without a hitch. In contrast, the examination of a conjecture which is not valid and is at the same time implied by rather complex rules does not terminate and leads to a timeout.

As already stated, we regard ourselves as "pure users" of the ATP tools and therefore treat them as black boxes. Their inner workings remain unconsidered since it is beyond the focus of this thesis, which is rather the general suitability of logic. This is why we leave the question unanswered why one does not encounter the above-mentioned issue when using cvc5 but when using iProver and Vampire.

9.1.4 Implementation in Prolog

Other than with TPTP, implementing the application process in Prolog does not create any mentionable difficulties with respect to programming the rules and to the automated reasoning. Once again, the list features in Prolog compensate very well the restricted expressiveness.

Even better, they allow a realization which is way more straightforward in both initially writing the program code and in retrospectively comprehending it. This mainly comes from the fact that one is not distracted by restraining any open-world semantics.

9.2 Capital Increase by Contribution

9.2.1 Legal Description

Process A108 from [GBM22] is the increase in the LLC's share capital, whose requirements are specified in §§ 53–57 GmbHG. In the interests of simplification, we limit the examination to the contribution in cash and exclude the even more complex contribution in kind.

Despite that, the process is noticeably more extensive than the director appointment, which manifests itself in the magnitude of the examination Scheme 9.2. Even though the scheme is not completely unfolded, like in the previous section, it takes up a whole page. Apart from that, it has no substantially greater deepness than the former schemes.

The complexity comes not only from the larger number of requirements on the top level—we can describe this as a greater *breadth* of the scheme—but in particular from the fact that there are many interdependencies between the conditions at the first level.

Similar to the director appointment and the company dissolution, the capital increase has its origin in a resolution by the shareholders. The resolution implies an amendment to the AoA because the now changing amount of shares is explicitly specified there. Due to this amendment, the resolution has to meet stricter requirements, in particular regarding the majority and the notarization.

The capital increase allows the entry of new shareholders, who in turn have to be admitted in a separate permit resolution. Furthermore, every shareholder who participates in the capital increase, i.e. acquires some of the newly generated shares, must sign a notarially recorded declaration for their subscription. This exemplary excerpt from the prerequisites illustrates the above-mentioned complexity of the interdependencies.

The regulations which are depicted in Scheme 9.2 have their origin not only in the legislation. The plausibility checks under item 9 do not result from legal norms but from settled case-law. [GBM22]

Finally, another interesting information, which is only a side note, is the fact that the application has not just a declarative but a constitutive effect. According to § 54 III GmbHG, as an AoA amendment, the capital increase has no legal force until the corresponding application has been processed by the register court.

The application is legal.

1. The amended AoA full text is attached.
2. Match of old AoA, amendments, and resolution are notarially certified.
3. The capital increase resolution is legal.
   - 3.1. Resolution is formally legal. (see item 1.1 from Scheme 2.1)
   - 3.2. Resolution fulfills material prerequisites.
     - 3.2.1. Resolution has majority of three quarters of the votes.
     - 3.2.2. Resolution satisfies additional AoA requirements.
   - 3.3. Resolution is notarized.
4. Requirements concerning a permit resolution (if necessary) are fulfilled.

   case 4.1. There are no newly joining shareholders.

   case 4.2. There are newly joining shareholders.
     - 4.2.1. All newly joining shareholders are mentioned in resolution.
     - 4.2.2. The resolution is legal. (see item 1.1 from Scheme 2.1)
5. List of the subscribers of the new shares is attached.
6. Declaration of each subscriber is attached.
   - 6.1 There is one declaration for each subscriber.
   - 6.2 Declarations are notarized.
   - 6.3 Declarations are available as original or certified copy.
7. Increased capital has been covered.
8. The assurance concerning the effectiveness and disposability of the contributions is attached.
9. The amended AoA are plausible.
   - 9.1. Former capital, increase, and new capital match.
   - 9.2. New capital and the new sum of nominal values match.
10. All directors perform the application.

Scheme 9.2: Requirements for the capital increase's application

9.2.2 Formalization

Most of the prerequisites from Scheme 9.2 can be formalized without any serious problems. Despite the scheme's considerable breadth, the individual subconditions do not reach the complicacy of the liquidator-related specifics which we formalized for the company dissolution process. Instead, the formalization's effort in a way "linearly" increases with the amount of subconditions compared to the director application.

Rules that are based on the pattern "all X must Y" (and equivalently "no X may Y") are strongly represented in the capital increase's prerequisites, e.g. in the items 4.1, 6.1, and 10. The structure of similar axioms that have been formulated in Chapter 4 can mostly be reused. However, the frequent occurrence of this pattern reinforces the repeatedly mentioned necessity for the closed world.

The expressiveness of first-order logic is brought to its limit by the attempt to formalize item 9.2 from the examination scheme, which compares the accumulated individual shares with the new capital information in the amended AoA. The sum of all values within a predicate's domain cannot be represented in first-order logic. Even though the considered rule does not originate from the GmbHG but "only" from a court order, such requirements can of course occur in practice.

9.2.3 Implementation in TPTP

The count of rule axioms for the TPTP implementation of the capital increase application is not less than 38, which correlates to the magnitude of the examination scheme.

In contrast to the company dissolution, the implementation did not create any difficulties, apart from the limitations in the antecedent formalization.

The outcomes of the experiments concerning the TPTP tool support (see Chapter 5) are equivalent to those of the director application process. Hence, the reasoning visualization can also be performed, which produces the satisfactory result that the extensive examination scheme can be clearly recognized in the proof tree and the visualized output.

9.2.4 Implementation in Prolog

Not surprisingly, the implementation in Prolog was successful and straightforward. Summing up the capital shares can be executed once again by means of Prolog's list operations.

9.3 Evaluation

Adaptability of Developments In spite of some minor difficulties, we are able to adapt the formalization, the implementation, and the visualization to two other register processes. We can therefore confirm that the developments from the previous chapters mainly still persist in new situations.

Restricted Expressiveness of First-Order Logic However, in spite of first-order logic's strong expressiveness, which allowed the formalization so far, we encountered the practical limitation that sums over a predicate's domain cannot be represented. Admittedly, this is a rather rare special case.

Semi-Decidability The semi-decidability of first-order logic has not become apparent until the implementation of the company dissolution. Considering the imaginable complexity of legal requirements, the semi-decidability, turns out to be a severe general problem when using first-order logic.

Generality of Conclusions Apart from those sobering realizations, we ascertained that most of our findings are not restricted to the director appointment. Instead, we have a clear indication of the fact that the intermediate results from the individual evaluation sections come with a certain degree of generality. We can thus use the intermediate findings to make universal conclusions for this thesis.

Necessity of Ontologies We noticed that a couple of rules occur in several register processes over and over again. This is the case for both explicit requirements (e.g. the formal aspects of a resolution) and implicit requirements which we blindly assumed (e.g. the legal capacity of the involved persons). This observation confirms the necessity to incorporate reusable ontological knowledge into the register process logic.

Benefits for More Complex Processes It is also noteworthy that dealing with the extensive examination schemes reveals that especially the automated rule decomposition gains in importance for more complex rule sets. The larger a scheme's breadth and depth is, the more time-consuming it is to identify by hand the single circumstance that disables the overall validity.

Tool Support Revision Finally, as cvc5 does not collapse under the complexity of the implemented company dissolution process, let us hereby emphasize its respectable powerfulness.

Chapter 10

Case Data Retrieval

Before we conclude this thesis, let us touch on a topic that is related to the application processes at the business process level. We are talking about the retrieval of structured case data. This is actually out of this thesis' scope but nevertheless worth being mentioned due to its significant relevance for the practical realization.

As ascertained in Section 4.1.2, the case facts—in contrast to the universal rules—are individual for each considered director application case. The question remains from where this case data can be retrieved in a structured and machine-readable form so that it is usable for an automated generation of the case-related axioms. In this chapter, we will briefly present a couple of possible methods to obtain that data.

Having established the JSON format for the case data inputs in Section 8.3, we can consider this format as the machine-readable target structure for the task.

10.1 Text-Based Data Extraction

The six documents from Fig. 2.3 have been suggested as a possible data source in Section 2.2.2. We can assume them to be already available in a plain text format.

[RN09, pp. 873–879, 888–906] proposes a handful of approaches to extract data from such text. In the following, the approaches which are interesting for our purposes will be explained on a rather abstract level.

10.1.1 Regular Expressions

Besides the no further noteworthy trivial method of extracting the relevant case data from the source documents by hand, another quite simple approach is the application of *regular expressions*. For this purpose, a pattern that describes the textual structure of the required information or its surrounding text is defined. There exists a general syntax for defining the pattern, albeit with some variations depending on the implementation. If the defined pattern matches the input text, then the affected portion of text can be extracted.

Regular expressions are well-suited for texts whose domain is restricted and whose structure is rather unvarying. Despite being somewhat primitive, that approach might actually bear fruit for our purposes. In practice, the documents like the AoA or the list of shareholders follow a quite uniform pattern and a predictable wording. Therefore, it is imaginable that a manageable amount of static regular expressions has the capability of extracting the case data from the documents. Nevertheless, unexpected deviations of even a single character in the text can make the approach fail.

Being supported by many programming languages, regular expressions are in practice a very common method for such tasks. That is why many programmers are familiar with this technique.

10.1.2 Probabilistic Methods

If the structure and the wording of the documents are not sufficiently uniform for employing regular expressions, then probabilistic methods might be a sensible route to pursue.

Hidden Markov Model A very simple method is the *hidden Markov model (HMM)*, in which the single words of the text are considered as a sequence of *observed variables* to each of which belongs a corresponding *hidden state*. This hidden state specifies whether the word is part of the required information or whether it is at least inside the information's context. The HMM makes the strict assumption that a hidden state only depends on its directly preceding state and models the full joint probability distribution of that transition model as well as the distribution of a state's influence on its corresponding observed variable.

Using the Viterbi algorithm on a model with trained probabilities, the most probable hidden state sequence for the text and thus the required information's position can be computed.

Conditional Random Fields An alternative probabilistic method is the *conditional random field (CRF)*, which does not have the HMM's strong independence assumption. Furthermore, in contrast to the HMM, the CRF does not model any full joint probability but the conditional probability of a set of target variables given the observations.

Both the HMM and the CRF require labeled training data in order to optimize their model parameters. Despite their high flexibility compared to regular expressions, the requirement for training might be a practical drawback for our information extraction task.

10.1.3 Grammar-Based Methods

A further approach aims to interpret the text as sentences of a *probabilistic context-free grammar (PCFG)*. Applying a parsing algorithm on each sentence yields a tree that specifies the sentence's grammatical structure. A grammar and the probabilities of its rules can be learned from a corpus of correctly parsed sentences (e.g. Penn Treebank).

The grammatical rules can be connected to semantics so that parsing a sentence results in a tree whose nodes are enriched with semantic interpretations. Eventually, the sentence can be represented as a logical formula, on which further inference is applicable.

10.2 Structured Data Resources

In Section 2.2, we distinguished the application case details on the basis of their invariability with respect to the application operation instance given the same company. The operation-related details are individual for the application instance, e.g. the resolution or the new director's assurance, whereas the operation-independent details do not differ from instance to instance but are in a way static for some company, e.g. the shareholders or the company's regulation for the general representation power. We also applied this distinction to the documents in Fig. 2.3.

It stands to reason that the operation-independent details can be obtained not only from the text-based operation-independent documents. Understandably, those details might also be available somewhere else in an already structured format. In this case, they can be used to generate a case data file that is at least partially completed.

10.2.1 Notary's Internal Database

It is probable that automated reasoning on register processes is performed in an environment where the companies are stored in some electronic database as structured data records. One can for example imagine a notary's office as such an environment. So, the operation-independent data for some company can be directly retrieved from that database in theory. The practical accessibility naturally depends on the database's technical implementation.

10.2.2 Commercial Register

A second obvious source of structured company data is the Commercial Register itself. As mentioned in Section 2.1.1, there exists a central portal providing data of all German companies on www.handelsregister.de. For each record listed there, one can access printable copies of the current and historical company details as well as an XML document with this data.

This XML document is structured according to a publicly accessible schema called *XJustiz*[1] with straightforward naming conventions. So, at first glance, that downloadable document is a well-suited source for case-independent data about the company.

Unfortunately, this approach has two drawbacks. Firstly, the embodied information is neither complete nor fully structured. Other than the directors, the shareholders are not included in the document. Moreover, a director's representation power is only available as free text within the document. Consequently, in order to obtain that data, it is necessary to fall back on the text-based extraction methods.

The second drawback is the portal's lacking provision of an official programming interface to retrieve the XML document even though the data is already available in a machine-readable format. Understandably, this shortcoming impedes the full automation of the data retrieval. Interestingly, in answer to that drawback, an unofficial application interface was implemented by a nongovernmental organization[2].

[1] https://www.xjustiz.de/

[2] see https://github.com/bundesAPI/handelsregister

Chapter 11

Conclusion

As we now come to the end of this thesis, let us consult the results from the individual chapters in order to make the final conclusions. As we have shown in Chapter 9, the intermediate results are suitable for making statements that go beyond our exemplary application process, the director appointment, and can thus be analogously transferred to other company register processes.

11.1 Sufficiency of Logic

The first research objective was the question whether logic is sufficiently powerful for the formalization and automation of company register processes.

11.1.1 Pros

We concluded after both the formalization and the concrete implementations that the undertakings have mostly been successfully completed using first-order logic. If we restrict the research question to first-order logic with its quite strong expressiveness, we can give a positive answer to it, albeit with slight reservations.

Successful Formalization The effectiveness of the formalization has been well supported by the structure of the rules. As we have noticed, complex legal rules are often composed of a manageable amount of recurrent "building blocks", namely sets of necessary or sufficient conditions and case distinctions. These fit very well to the conjunction and disjunction in classical logic. Moreover, the analogy to the juridical syllogism contributed to the success in the formalization.

Successful Implementation The subsequent implementation also turned out to be rather uncomplicated since both TPTP and Prolog provide suitable infrastructures for that purpose. Furthermore, the limitations of Prolog compared to first-order logic could be well compensated by its list operations.

11.1.2 Cons

Admittedly, the path to the goal was not free of obstacles.

Automated Reasoning Issues Apart from the mostly conquerable open-world assumption of first-order logic, the semi-decidability is a severe limitation, even though it is not frequently observed according to our experience. The heterogeneous "cooperativeness" of the TPTP tools originates on the one hand from the semi-decidability, which is certainly specific to first-order logic. On the other hand, the occurred issues are implementation-related and could at least partially be alleviated with some practical workarounds.

Expressiveness-Decidability-Tradeoff If one considers that first-order logic is pretty expressive, the decidability issue appears somewhat forgivable. However, the adaption to the capital increase process revealed a single limitation of the expressiveness. Even though this sole incident does not allow general statements about the extent of the limitation in practice, it at least raises doubts concerning the existence of an acceptable tradeoff between expressiveness and decidability, which argues against the sufficiency of logic.

11.1.3 Conclusion

Applying logic-based approaches does the job on the whole. Despite some considerable obstacles, first-order logic in particular is sufficiently powerful to formalize at least the great majority of the aspects of company register processes.

11.2 Necessity of Logic

The second research question was if logic is necessary for the formalization and automation of register processes. This thesis focused on the utilization of logic. As it did not cover the alternatives to logic and their suitability, the question of necessity can understandably not be conclusively answered. We can only highlight the benefits which are specific to logic and provide them as a benchmark for research on the alternatives to logic-based approaches.

11.2.1 Pros

Apart from the general feasibility of the formalization and the infrastructural support for the implementation, especially the ontology incorporation in Chapter 7 and the traceability of reasoning in Chapter 8 revealed the advantages of sticking to logic.

Ontology Incorporation If legal reasoning is at least partially intended to compete with a human jurist, then it is has to include a lot of knowledge. This knowledge is provided by legal and perhaps also commonsense ontologies. As we have shown for OWL, common technologies for ontology representation are compatible with logic. So, alternatives to logic can only benefit from that knowledge if they ensure the compatibility to established ontologies.

Reasoning Traceability A crucial advantage of logic-based approaches is the traceability of conclusions. While a successful proof can be justified by the ATP tool itself, we also developed an approach to decompose the formalized rules and therefore enable to easily reproduce why exactly some conclusion is positive or negative. This also allows for a high preciseness of the results. In contrast, especially statistical approaches have the problem of nontransparent decision-makings whose results are not retraceable. [Gru18]

11.2.2 Cons

Formalization Effort We considered the examination schemes for the application processes as preparatory groundworks of this thesis. The translation of the schemes to the logic axioms by hand is a quite time-consuming task, which constitutes a large drawback of our method of proceeding. The idea that alternative techniques, like probabilistic or learning approaches, can solve or at least alleviate the problem of the high formalization effort gives reason to hope.

11.2.3 Conclusion

The question concerning the necessity of logic cannot be ultimately answered. We highlighted the benefits of logic as a benchmark on which research into the alternatives can build.

11.3 Suitability of First-Order Logic

The research question in respect of the best-suited kind of logic can only be answered with a focus on first-order logic because it was the only logic that we used for the formalization.

11.3.1 Pros

The advantages of first-order logic have already been stated during the discussion of the sufficiency and necessity. They possibly also apply to other kinds of logic, but, in the case of first-order logic, we actually proved them.

11.3.2 Cons

Monotonicity First-order logic's monotonicity and the related open-world assumption were recurrent problems in the undertaking. Since legal reasoning must be based on given knowledge and the assumption that the knowledge is complete, the open-world assumption is largely impedimental.

Semi-Decidability The semi-decidability is an even more severe problem in practice. It may lead to the unconquerable situation that a non-valid proposition cannot be checked, which constitutes a great drawback of first-order logic.

Expressiveness Despite the strong expressiveness of first-order logic, we reached its limits when we formalized the capital increase process. Interestingly, Prolog's expressiveness was powerful enough to cope with the issue.

11.3.3 Conclusion

First-order logic is probably not the most reasonable option for formalizing company register processes and performing legal reasoning. It can be assumed that a non-monotonic logic might be the better choice.

11.4 Secondary Achievements

The outcome of this thesis is not limited to the answers to the research questions. Beyond this, our work created a couple of by-products. Let us have a brief look at the two mentionable secondary achievements.

Reasoning Traceability The reasoning visualization, which we developed in Chapter 8, in combination with the formalization conventions is a useful groundwork for some further practical implementation that strives to make automated legal reasoning traceable. The tool is not limited to being used in the scope of register law but can be applied to every formalization that complies with the established conventions.

OWL Ontology Inclusion Both TPTP and OWL are widespread and well-established technologies. By building on the open source translation from OWL into TPTP, we performed the conversion of the multi-file ontology LKIF Core in Section 7.1. Based on this, we documented the practical issues of including the conversion's output into a logic program in TFF syntax.

11.5 Future Work

Finally, let us enumerate a handful of topics which have only been slightly touched but constitute interesting subjects for research beyond this thesis.

Non-Monotonic Logics Having limited ourselves to classical first-order logic, one could beyond this examine other formal logics. Apart from the obvious switch to first-order logic with database semantics, there is also defeasible logic, into which we received a very tiny insight when we discussed LegalRuleML. Defeasible logic might be more suitable for legal reasoning than first-order logic because the concept of rules and exceptions to these rules might possibly better meet the legal reality.

Exploring Alternatives to Logic Since the research question concerning the necessity of logic could not be fully answered, an obvious idea for further research is the examination of AI techniques other than logic for dealing with the register processes. As already mentioned in the introduction, there also exist probabilistic and learning approaches. To be complete, it is furthermore imaginable to implement a legal requirements check in classical imperative programming in order to get rid of logic's limitations of expressiveness. Examining the suitability of those approaches for automated reasoning in the area of register processes would be the perfect complement to this thesis.

Extensions for Reasoning Visualization Of course, the reasoning visualization from Chapter 8 offers room for extension. Since it has only been

implemented for TPTP programs, an adaption to Prolog would be a possibility. Along with a so far missing graphical user interface, one could also implement a feature for "switching off" single requirements in the examination scheme according to the assumed subconditions from Section 4.5.2.

Case Data Retrieval We motivated the automated retrieval of structured case data for the generation of case-related axioms in Chapter 10. The otherwise manual collection of the data is impractical and therefore necessitates automation. Due to the fact that the documents in the field of register law are often uniformly structured, the automated case data retrieval from these documents shows great promise for success.

Automated Formalization As already mentioned, the manual formalization of the applications' prerequisites from the examination schemes is a laborious undertaking. This motivates the automation of the task. For many standard scenarios, especially in criminal law, there already exist such overviews, which have once been carried together from all the related norms. Alternatively, they can be constructed by a jurist with reasonable effort. It is imaginable that the approaches for the case data retrieval from Chapter 10 are suitable for the automated generation of law axioms as well. The nested scheme structure consisting of necessary and sufficient conditions as well as case distinctions furthermore already provides a coarse pattern.

Examination of TPTP Prover Limitations During the implementation in TPTP, we encountered different limitations of the utilized provers. As we treated these as black boxes, this thesis leaves open the question due to which reasons the particular failures occurred. Both our formalization and the developers of the prover tools could profit from such information.

Ontology Incorporation for Assumptions The examination schemes for the application processes are based on a couple of implicit and explicit assumptions. Of course, this restricts the schemes' scope of application. However, in the event that an assumption does not hold, a jurist with implicit knowledge has no difficulty to recognize that and to come up with a legal remedy. In contrast, artificial intelligence cannot deliver what it was not taught (externally or by itself). Our implicit assumptions from commonsense or the civil code, e.g. an agent's legal capacity, affect countless legal situations and are therefore predestinated for being formalized as reusable ontological knowledge. This motivates further research into making legal reasoning stronger and more human-like via ontology incorporation.

Bibliography

[Adr10] Axel Adrian. "Wie wissenschaftlich ist die Rechtswissenschaft?: Gibt es eine bindende Methodenlehre?" In: *Rechtstheorie* 41.4 (2010), pp. 521–548.

[Ash17] Kevin Ashley. *Artificial Intelligence and Legal Analytics: New Tools for Law Practice in the Digital Age*. Cambridge: Cambridge University Press, 2017, p. 8.

[BHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web: A New Form of Web Content That is Meaningful to Computers Will Unleash a Revolution of New Possibilities". In: *ScientificAmerican.com* 284.5 (2001), pp. 34–43.

[Bra86] Ivan Bratko. *Prolog Programming for Artificial Intelligence*. Wokingham: Addison-Wesley, 1986.

[End01] Herbert B. Enderton. *A Mathematical Introduction to Logic*. 2nd ed. San Diego: Harcourt/Academic Press, 2001.

[Ert21] Wolfgang Ertel. *Grundkurs Künstliche Intelligenz*. 5th ed. Wiesbaden: Springer Vieweg, 2021, pp. 77–78.

[Fit96] Melvin Fitting. *First-order logic and automated theorem proving*. 2nd ed. New York: Springer-Verlag, 1996, p. 28.

[Fle22] Holger Fleischer. "Einleitung". In: *Münchener Kommentar GmbHG*. Ed. by Holger Fleischer and Wulff Goette. 4th ed. Vol. 1. Rn. 202-204. München: C.H.Beck, 2022.

[Flü+21] Simon Flügel et al. "FOWL - An OWL to FOL Translator". In: *The Joint Ontology Workshops Episode VII (JOWO 2021)*. 2021.

[GBM22] Eckhardt Gustavus, Walter Böhringer, and Robin Melchior. *Handelsregisteranmeldungen*. 11th ed. Köln: Otto Schmidt, 2022.

[Gru18] Michael Grupp. “Verfügbare und machbare Rechtsautomation: Induktiv vs. Deduktiv”. In: *Legal Tech*. Ed. by Markus Hartung, Micha-Manuel Bues, and Gernot Halbleib. München: C.H.Beck, 2018, pp. 260–263.

[Gug19] Leonid Guggenberger. “Einsatz künstlicher Intelligenz in der Verwaltung”. In: *Neue Zeitschrift für Verwaltungsrecht* 38.12 (2019), pp. 844–850.

[Häh+20] Susanne Hähnchen et al. “Legal Tech: Rechtsanwendung durch Menschen als Auslaufmodell?” In: *Juristische Schulung* 60.7 (2020), pp. 625–635.

[Hoe+07] Rinke Hoekstra et al. “The LKIF Core Ontology of Basic Legal Concepts”. In: *2nd Workshop on Legal Ontologies and Artificial Intelligence Techniques (LOAIT 2007)*. 2007, pp. 43–63.

[Hoe07] Rinke Hoekstra. *The LKIF Core Ontology of Basic Legal Concepts*. Web Page. 2007. URL: https://github.com/RinkeHoekstra/lkif-core (visited on Feb. 14, 2023).

[Koh22] Michael Kohlhase. *Artificial Intelligence I, Winter Semester 2021/2022, Lecture Notes*. Erlangen, 2022. URL: https://kwarc.info/teaching/AI/notes/notes-2021-22.pdf (visited on Feb. 27, 2023).

[Kow78] Robert Kowalski. “Logic for Data Description”. In: *Logic and Data Bases*. Ed. by Hervé Gallaire and Jack Minker. 1st ed. Boston: Springer US, 1978, pp. 77–103.

[LDV20] Valentina Leone, Luigi Di Caro, and Serena Villata. “Taking Stock of Legal Ontologies: a Feature-based Comparative Analysis”. In: *Artificial Intelligence and Law* 28.2 (2020), pp. 207–235.

[LH18] Ho-Pun Lam and Mustafa Hashmi. “Enabling Reasoning with LegalRuleML”. In: *Theory and Practice of Logic Programming* 19.1 (2018).

[Mue15] Erik T. Mueller. *Commonsense Reasoning: An Event Calculus Based Approach*. 2nd ed. Amsterdam: Morgan Kaufman, 2015.

[Obe90] Arnold Oberschelp. “Order Sorted Predicate Logic”. In: *Sorts and Types in Artificial Intelligence*. Ed. by Karl Hans Bläsius, Ulrich Hedtstück, and Claus-Rainer Rollinger. Springer, 1990, pp. 7–17.

[Pal+11] Monica Palmirani et al. “LegalRuleML: XML-Based Rules and Norms”. In: *Rule-Based Modeling and Computing on the Semantic Web*. Ed. by Frank Olken, Monica Palmirani, and Davide Sottara. Springer, 2011, pp. 298–312.

[Pal+21] Monica Palmirani et al. *LegalRuleML Core Specification Version 1.0*. Web Page. 2021. URL: https://docs.oasis-open.org/legalruleml/legalruleml-core-spec/v1.0/legalruleml-core-spec-v1.0.html (visited on Feb. 14, 2023).

[Rei78] Raymond Reiter. “On Closed World Data Bases”. In: *Logic and Data Bases*. Ed. by Hervé Gallaire and Jack Minker. Boston: Springer US, 1978, pp. 55–76.

[Rei82] Raymond Reiter. “Circumscription Implies Predicate Completion (Sometimes)”. In: *The Second National Conference on Artificial Intelligence (AAAI-82)*. Ed. by David. L Waltz. 1982, pp. 418–420.

[RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Upper Saddle River: Pearson Education, 2009.

[SF22] Alexander Steen and David Fuenmayor. “Bridging Between LegalRuleML and TPTP for Automated Normative Reasoning”. In: *Rules and Reasoning*. Ed. by Guido Governatori and Anni-Yasmin Turhan. Springer International Publishing, 2022, pp. 244–260.

[Sie20] Jan-Reinard Sieckmann. *Logik juristischer Argumentation*. 1st ed. Vol. 76. Studien zur Rechtsphilosophie und Rechtstheorie. Baden-Baden: Nomos, 2020, p. 24.

[Sik17] Leslie Sikos. *Description Logics in Multimedia Reasoning*. Cham: Springer, 2017, p. 81.

[SS01a] Geoff Sutcliffe and Christian Suttner. *SyntaxBNF*. Web Page. 2001. URL: https://www.tptp.org/TPTP/SyntaxBNF.html#tff_subtype (visited on Feb. 5, 2023).

[SS01b] Geoff Sutcliffe and Christian Suttner. *System on TPTP*. Web Page. 2001. URL: https://www.tptp.org/cgi-bin/SystemOnTPTP (visited on Feb. 5, 2023).

[SS01c] Geoff Sutcliffe and Christian Suttner. *The TPTP Problem Library for Automated Theorem Proving*. Web Page. 2001. URL: https://www.tptp.org/ (visited on Feb. 5, 2023).

[SS01d] Geoff; Sutcliffe and Christian Suttner. *TPTP Documents File: SZSOntology*. Web Page. 2001. URL: https://www.tptp.org/cgi-bin/SeeTPTP?Category=Documents&File=SZSOntology (visited on Feb. 5, 2023).

[Ste22] Peter Stelmaszczyk. "§ 37 Das Gesellschaftsrecht heute und morgen – Auswirkungen der Digitalisierung auf das Gesellschaftsrecht und die Rolle des Notars". In: *Notarhandbuch Gesellschafts- und Unternehmensrecht*. Ed. by Hauschild, Kallrath, and Wachter. 3rd ed. Rn. 6-9. München: C.H.Beck, 2022, pp. 1990–1991.

[Sut10] Geoff Sutcliffe. "The TPTP World – Infrastructure for Automated Reasoning". In: *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-16)*. Ed. by Edmund M. Clarke and Andrei Voronkov. Springer, 2010, pp. 1–12.

[Sut16] Geoff Sutcliffe. "The CADE ATP System Competition - CASC". In: 37.2 (2016), pp. 99–101. URL: https://www.tptp.org/CASC/.

[Sut17] Geoff Sutcliffe. "The TPTP Problem Library and Associated Infrastructure". In: *Journal of Automated Reasoning* 59.4 (2017), pp. 483–502.

[SWI] SWI-Prolog. *Running goals from the command line*. Web Page. URL: https://www.swi-prolog.org/pldoc/man?section=runoptions (visited on Feb. 28, 2023).

[Vil+22] Serena Villata et al. "Thirty Years of Artificial Intelligence and Law: The Third Decade". In: *Artificial Intelligence and Law* 30.4 (2022), pp. 561–591.

[W3C04] W3C. *OWL Web Ontology: Language Guide*. Web Page. 2004. URL: https://www.w3.org/TR/owl-guide/ (visited on Feb. 14, 2023).

[W3C12] W3C. *OWL 2 Web Ontology Language: Direct Semantics (Second Edition)*. Web Page. 2012. URL: https://www.w3.org/TR/owl2-direct-semantics/ (visited on Feb. 14, 2023).

Appendix A

Rule Formula Transformation

This transformation is referred by Section 4.3. It converts a Horn-shaped rule for the predicate *con* (conclusion) with variable A into a format where it is only universally quantified over A. The rule body consists of the predicates *rel* (relation) and *pre* (precondition). In each line, the constituent which is affected by the next transformation is highlighted.

$$
\begin{aligned}
& \forall A.\Big[\quad \forall B.\ \big[\ (rel(A,B) \wedge pre(B)) \Rightarrow con(A) \quad\big] \quad\Big] \\
\equiv\ & \forall A.\Big[\quad \neg\exists B.\ \big[\ \neg\big((rel(A,B) \wedge pre(B)) \Rightarrow con(A)\big) \quad\big] \quad\Big] \\
\equiv\ & \forall A.\Big[\quad \neg\exists B.\ \big[\ \neg\big(\neg(rel(A,B) \wedge pre(B)) \vee con(A)\big) \quad\big] \quad\Big] \\
\equiv\ & \forall A.\Big[\quad \neg\exists B.\ \big[\ \neg\big(\neg rel(A,B) \vee \neg pre(B) \vee con(A)\big) \quad\big] \quad\Big] \\
\equiv\ & \forall A.\Big[\quad \neg\exists B.\ \big[\ rel(A,B) \wedge pre(B) \wedge \neg con(A) \quad\big] \quad\Big] \\
\equiv\ & \forall A.\Big[\neg\Big(\quad \exists B.\ \big[\ rel(A,B) \wedge pre(B) \wedge \neg con(A) \quad\big]\Big) \quad\Big] \\
\equiv\ & \forall A.\Big[\neg\Big(\quad \exists B.\ \big[\ rel(A,B) \wedge pre(B) \quad\big] \wedge \neg con(A)\Big) \quad\Big] \\
\equiv\ & \forall A.\Big[\quad \neg\exists B.\ \big[\ rel(A,B) \wedge pre(B) \quad\big] \vee con(A) \quad\Big] \\
\equiv\ & \forall A.\Big[\quad \exists B.\ \big[\ rel(A,B) \wedge pre(B) \quad\big] \Rightarrow con(A) \quad\Big]
\end{aligned}
$$

Appendix B

Director Appointment: Case Data JSON

This is our running example from Section 2.3 in the format which we established in Section 8.3.

```
{
  "company": {
    "shareholders": [
      { "name": "alice", "votes": 50001 },
      { "name": "ben", "votes": 39999 },
      { "name": "chris", "votes": 10000 }
    ],
    "directors": [
      { "name": "jacob",
        "representationpower": "",
        "exemption181": false },
      { "name": "kate",
        "representationpower": "modified",
        "exemption181": false },
      { "name": "chris",
        "representationpower": "sole",
        "exemption181": true }
    ],
    "representationpower": "joint",
    "majorityrequirement": 0.60
  },
  "resolution": {
    "new_director": {
```

```
      "name": "luca",
      "representationpower": "modified",
      "exemption181": false
    },
    "meeting": {
      "occurred": true,
      "format": "teleconference",
      "consents_to_teleconference": [
        "alice",
        "ben",
        "chris"
      ]
    },
    "voting": {
      "yes_votes": 60001,
      "no_votes": 39999,
      "abstentions": 0
    },
    "written_consents": null,
    "consents_to_voting_in_writing": null
  },
  "application": {
    "assurance_signed": true,
    "deed_format": "original",
    "applicants": [
      "jacob",
      "kate"
    ]
  }
}
```

Appendix C

Company Dissolution: Rule for Responsible Applicants

This is the rule from Section 9.1 for determining the persons who are responsible for submitting the company dissolution to the register court.

$$
\begin{array}{l}
\forall_{app} A.\ \forall_{dir} D.\ (\\
\quad application\_responsibleapplicant(A, D) \\
\quad \Longleftrightarrow \\
\quad \exists_{res} E.\ \exists_{com} C.\ \exists_{liql} L.\ (\\
\qquad\quad application\_resolution(A, R) \\
\qquad \wedge \quad company\_resolution(C, R) \\
\qquad \wedge \quad company\_liquidatorlist(C, L) \\
\qquad \wedge \quad (\\
\qquad\qquad\quad (\quad are\_AoA\_amended(R) \\
\qquad\qquad\quad \wedge \quad \exists_{dir} X.\ (\\
\qquad\qquad\qquad\qquad\quad director\_company(X, C) \\
\qquad\qquad\qquad\qquad \wedge \quad D = X \\
\qquad\qquad\qquad\quad) \\
\qquad\qquad\quad) \\
\qquad\qquad \vee \quad (\quad \neg are\_AoA\_amended(R) \\
\qquad\qquad\quad \wedge \quad \exists_{dir} X.\ (\\
\qquad\qquad\qquad\qquad\quad liquidatorlist\_liquidator(L, X) \\
\qquad\qquad\qquad\qquad \wedge \quad D = X \\
\qquad\qquad\qquad\quad) \\
\qquad\qquad\quad) \\
\qquad\qquad) \\
\quad\quad) \\
)
\end{array}
$$