Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departmento de Informática e Matemática Aplicada
Programa de Pós-Graduação em Sistemas e Computação
Mestrado Acadêmico em Sistemas e Computação

A method for automated generation of proof exercises with a comparable level of complexity

João Mendes Lopes Neto

Natal-RN
July 2025

João Mendes Lopes Neto

A method for automated generation of proof exercises with a comparable level of complexity

Master's thesis presented to the Programa de Pós-Graduação em Sistemas e Computação of the Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte as a partial requirement for obtaining the Master's degree in Systems and Computing.

Research area:
Computing Fundamentals

Supervisor

João Marcos

Cosupervisor

Patrick Terrematte

PPgSC – Programa de Pós-Graduação em Sistemas e Computação
DIMAp – Departamento de Informática e Matemática Aplicada
CCET – Centro de Ciências Exatas e da Terra
UFRN – Universidade Federal do Rio Grande do Norte

Natal-RN

July 2025

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Setorial Prof. Ronaldo Xavier de Arruda - CCET

Lopes Neto, João Mendes.
A method for automated generation of proof exercises with a comparable level of complexity / João Mendes Lopes Neto. - 2025.
112 f.: il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Norte, Centro de Ciências Exatas e da Terra, Programa de Pós-Graduação em Sistemas e Computação. Natal, RN, 2025.
Orientação: João Marcos.
Coorientação: Patrick Terrematte.

1. Computação - Dissertação. 2. Automatic question generation - Dissertação. 3. Teaching logic - Dissertação. 4. Cut-based tableaux - Dissertação. I. Marcos, João. II. Terrematte, Patrick. III. Título.

RN/UF/CCET CDU 004(043.3)

Elaborado por Joseneide Ferreira Dantas - CRB-15/324

JOÃO MENDES LOPES NETO

"*Um método para geração automatizada de exercícios de demonstração com nível de complexidade similar*"

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Sistemas e Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte.

Documento assinado digitalmente
EVERTON RANIELLY DE SOUSA CAVALCANTE
Data: 21/07/2025 15:36:52-0300
CPF: ***.194.874-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. EVERTON RANIELLY DE SOUSA CAVALCANTE

Coordenador do PPgSC

Banca Examinadora:

Documento assinado digitalmente
CLAUDIA NALON
Data: 21/07/2025 12:22:55-0300
CPF: ***.416.151-**
Verifique as assinaturas em https://v.ufsc.br

Examinadora Externa: **Dr.ª CLAUDIA NALON**

Documento assinado digitalmente
ADOLFO GUSTAVO SERRA SECA NETO
Data: 21/07/2025 12:24:34-0300
CPF: ***.351.324-**
Verifique as assinaturas em https://v.ufsc.br

Examinador Externo: **Dr. ADOLFO GUSTAVO SERRA SECA NETO**

Documento assinado digitalmente
PATRICK CESAR ALVES TERREMATTE
Data: 21/07/2025 14:00:02-0300
CPF: ***.330.094-**
Verifique as assinaturas em https://v.ufsc.br

Examinador Externo: **Dr. PATRICK CESAR ALVES TERREMATTE**

Documento assinado digitalmente
REGIVAN HUGO NUNES SANTIAGO
Data: 21/07/2025 18:14:05-0300
CPF: ***.805.812-**
Verifique as assinaturas em https://v.ufsc.br

Examinador Interno: **Dr. REGIVAN HUGO NUNES SANTIAGO**

Presidente: **Dr. JOÃO MARCOS DE ALMEIDA**

Documento assinado digitalmente
JOAO MARCOS DE ALMEIDA
Data: 21/07/2025 11:50:35-0300
CPF: ***.976.956-**
Verifique as assinaturas em https://v.ufsc.br

Documento assinado digitalmente
JOAO MENDES LOPES NETO
Data: 22/07/2025 09:43:20-0300
CPF: ***.975.634-**
Verifique as assinaturas em https://v.ufsc.br

Discente: **JOÃO MENDES LOPES NETO**

Natal, 21 de julho de 2025.

To my parents, Consuêlo and Lindomar.

Acknowledgements

I would not have made it here without the support and love my family has shared. I especially thank my parents for giving me the privilege of choosing my own renunciations.

During these years, I was very lucky to have so solicitous supervisors who cared about me not only as a supervisee. Here is my word of gratitude: to João Marcos, for always doing his best in trying to make me understand *doncovim*, *oncotô* and *proncovô*; and to Patrick, for showing me that there might always be an optimistic point of view.

Still on the members of the examination board, I thank professors Cláudia Nalon, Adolfo Neto and Regivan Santiago for kindly accepting to join it and help in the improvement of this work. I also thank professor Carlos Olarte for his fundamental contributions on the final version of this manuscript.

Being part of a community full of generous and welcoming people has been, and continues to be, a substantial source of motivation for me to work with Logic and, hence, write this thesis. I thank all those who somehow contribute for the Brazilian logic community to be what it is and inspire me in trying to do the same.

To finish this thesis, I have had the support of many amazing people. I will not risk trying to list them all, but I cannot help expressing how deeply grateful I am to all those who are indirectly present in this work.

Last but not least, I express my gratitude to all those who keep me believing in the collective act of dreaming.

This work was financially supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — Brasil (CAPES) — Finance Code 001.

"Eu não tenho hora pra morrer
Por isso sonho"

– Rita Lee, *Coisas da vida*

A method for automated generation of proof exercises with a comparable level of complexity

Author: João Mendes Lopes Neto
Supervisor: João Marcos
Cosupervisor: Patrick Terrematte

Abstract

The automated generation of exercises may benefit educators by significantly reducing the time they spend in manually creating exercises. However, an obstacle in making such automation more present in an educator's professional routine is controlling the level of complexity of mechanically generated exercises. In this work, we present a method for the automated generation of proof exercises with a comparable level of complexity. The inputs of this method are a proof exercise and a set of rules allowing to prove this exercise. The output is a set of proof exercises with a comparable complexity to that of given as input. The scope of exercises we work with are mathematical proof exercises described in first-order languages, covering topics such as Set Theory and Number Theory. In order to calculate the level of complexity of these exercises, we base our approach on the effort required to solve them via informal proofs. We argue that such an effort, in turn, may be captured by formal proofs in cut-based tableaux that do not contain logical symbols. The rules utilized in these proofs are extracted by a mechanizable procedure we provide. We use the analytic character of such rules and the formal structure tableau proofs have to provide a computational procedure of the method in question. As case studies, we demonstrate how our method works with fragments of Set Theory and Number Theory. A prototype implementation of the method was also developed and we present it here.

Keywords: Automatic Question Generation, Teaching Logic, Cut-based Tableaux

Um método para geração automatizada de exercícios de demonstração com um nível de complexidade similar

Autor: João Mendes Lopes Neto
Orientador: João Marcos
Coorientador: Patrick Terrematte

Resumo

A geração automatizada de exercícios pode beneficiar educadores ao reduzir significativamente o tempo que gastam na elaboração manual de exercícios. No entanto, um obstáculo para tornar essa automação mais presente na rotina profissional de um educador é o controle do nível de complexidade de exercícios gerados mecanicamente. Neste trabalho, apresentamos um método para a geração automatizada de exercícios de demonstração com um nível de complexidade similar. As entradas deste método são um exercício de demonstração e um conjunto de regras as quais permitem demonstrar este exercício. A saída é um conjunto de exercícios de demonstração com uma complexidade similar à do exercício passado como entrada. O escopo dos exercícios com os quais trabalhamos são exercícios de demonstração matemática descritos em linguagens de primeira ordem, cobrindo tópicos como Teoria dos Conjuntos e Teoria dos Números. Para calcular o nível de complexidade destes exercícios, fundamentamos a nossa abordagem no esforço necessário para os resolver através de demonstrações informais. Argumentamos que esse esforço, por sua vez, pode ser capturado por demonstrações formais em tablôs baseados em corte que não contêm símbolos lógicos. As regras utilizadas nestas demonstrações são extraídas por um procedimento mecanizável que fornecemos. Usamos o carácter analítico dessas regras e a estrutura formal que as demonstrações de tablô têm para fornecer um procedimento computacional do método em questão. Como casos de estudo, demonstramos como o nosso método funciona com fragmentos da Teoria dos Conjuntos e da Teoria dos Números. Uma implementação prototípica também foi desenvolvida e nós a apresentamos aqui.

Palavras-chave: Geração Automática de Questões, Ensino de Lógica, Tablôs baseados em corte

List of figures

List of tables

List of abbreviations and acronyms

AQG – Automatic Question Generation

QDET – Question Difficulty Estimation from Text

PNF – Prenex Normal Form

BFS – Breadth-First Search

Contents

1 Introduction

Besides the time in a classroom, the schedule of an educator may be filled with several other activities, such as preparing lessons, extraclass support to students and delivering exercises and assessments. In particular, the manual construction of exercises can require a significant space in this schedule. One solution to tackle this problem are Automatic Question Generation (AQG) tools. However, the lack of AQG tools that generate exercises with controlled level of complexity still poses a challenge for their incorporation in the pedagogical practice [3, 14, 1].

In this work, we propose a method for automated generation of proof exercises with a comparable level of complexity. The proving complexity of a proof exercise is calculated in terms of a tableau proof for it whose formulas contain no logical symbols. These proofs are a particular case of the cut-based tableaux system presented by D'Agostino and Mondadori [7, 8] and we refer to them as *theory-specific proofs*. The *theory-specific rules* used to construct them are extracted from first-order closed formulas, which we call *definitional axioms*, via a mechanizable procedure. Through the definitional axioms, the objective is to describe fragments of first-order mathematical theories of pedagogical interest, as Set Theory and Number Theory, in such a way that we can derive provable conjectures that might be used as proof exercises in a classroom.

In the diagram of Figure 1, we present a big picture of our method. After the extraction from the definitional axioms, the theory-specific rules are preprocessed so they are provided as input to the procedure the generates the proof exercises. This procedure has also as input a proof exercise and, as output, a set of proof exercises with a comparable complexity to that of given as input.

In summary, our method relies on three computational procedures that are theoretically developed throughout this work:

1. Extraction of theory-specific rules,

2. Preprocessing of theory-specific rules and

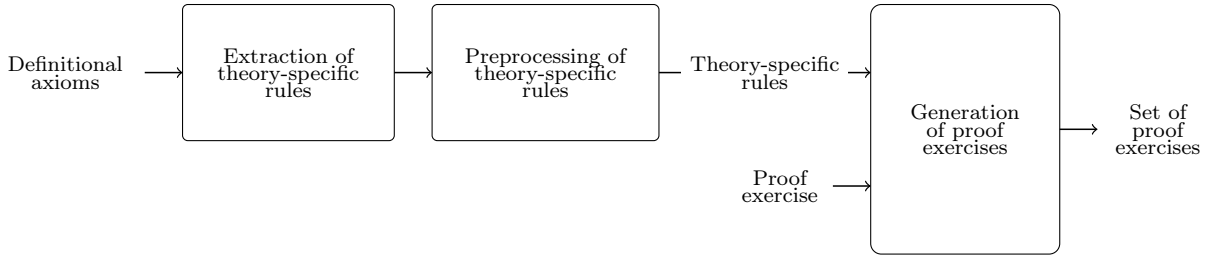


Figure 1: A diagram of the big picture of the method for automated generation of proof exercises with a comparable level of complexity.

3. Generation of proof exercises with a comparable level of complexity, which we regard as the main one.

The aim with the first and second procedures is that they generate a set of tableau expansion rules that do not contain logical symbols neither in their premises nor in their conclusions and with the same deductive strength than the definitional axioms initially specified. For the third procedure, we need first to formalize what it means two proof exercises having a "comparable level of complexity". With this formalization in hands, the procedure of generation of proof exercises is developed as a search procedure, which we also implement.

The pedagogical applicability of our method is demonstrated with two case studies: a fragment of Set Theory and a fragment of Number Theory. Through them, we aim to show how higher Mathematics courses in which students are taught how to prove theorems described in first-order languages may benefit from our work.

The present document is organized as follows: Chapter 2 introduces AQG tools used in education focusing on the problem of control of difficulty; Chapter 3 presents the theoretical background for the development of our method and presents the procedure of extraction of theory-specific rules; in Chapter 4, the notions of proving complexity concerning our educational purpose are formalized and, then, the preprocessing of theory-specific rules and the generation of proof exercises procedures are described; limitations we have found and perspectives for futures works are detailed in the Conclusion; in the Appendix A, the `C++` prototype implementation of generation of proof exercises is described.

2 Automatic Question Generation and control of difficulty

The start of the second half of the 20th century marks the period in which the first steps towards the inclusion of computers in education were taken [16]. In the beginning, the use was primarily focused on the power of computer for doing numerical calculations. The impact caused by this use, nevertheless, seemed to have stayed far below from the impact computer could cause in education: "Why then should computers in schools be confined to computing the sum of the squares of the first twenty odd numbers and so-called 'problem-solving' uses? Why not use them to produce some action?" [21].

Later on, the development of the system Plato [1] and the environment LOGO [2] represented milestones to this change of paradigm. Following this period of advances, in the 70's, AQG methods started to be developed [3]. The early efforts were devoted to automating the production of questions from texts so as to liberate educators from the laborious work of manually constructing questions for their students. Since then, meaningful progress has been made and, nowadays, there are AQG tools covering dozens of domains and generating questions of different types [14] [9]. Currently, AQG methods also find other applications than education, as for chatbots and fact-checking.

Considering the educational scope, an AQG method usually has two inputs:

1. A *knowledge source*, that can be either structured (e.g. databases and ontologies) or unstructured (e.g. continuous texts). This knowledge source provides the information that allows the method to find the solution(s) for the generated questions. In our method, the knowledge source is the set of rules of the cut-based tableaux system which we use to extract the proofs.

[1] Released in 1960 at the University of Illinois, Plato was a precursor for computer-assisted instruction systems. More information about its history may be found at `platohistory.org`.

[2] Designed for children, LOGO is a computational learning environment released in 1967. More information about its history may be found at `el.media.mit.edu/logo-foundation`.

2. A set of information with the *desired properties* for the generated questions (e.g. amount of distractors for multiple choice questions). In our method, we desire the generated exercises to have a comparable level of proving complexity to that of the input exercise. As we define in Chapter 4, this complexity is calculated in terms of a minimal proof for the input exercise. Then, our desired properties are encapsulated in the input exercise.

In spite of all the progress that has been done in the last decades, some gaps regarding question construction still remain. In [14], the authors conducted a review on 93 papers about AQG for educational purposes published from 2015 to 2019. From these 93 papers, only 14 presented methods that had a mechanism to control the difficulty of their outputs. As it is posed in [3], this challenge is not a novelty since the lack of approaches for the generation of questions with controlled levels of difficulty was also identified in a similar analysis of 81 papers published between 1969 and 2015. And a more recent study [1], comprising AQG tools that use deep learning for educational purposes, highlights the customization of these tools by level of difficulty as "a further research opportunity".

The mechanization purpose of AQG conflicts with subjective aspects of determining how complex a question is. In light of this conflict, we dedicate this chapter to answer two questions: *"Why control the level of difficulty of questions?"* and *"How to control the level of difficulty of questions?"*. In the first section, we present reasons why control of difficulty should be taken into account when preparing assessments. In the second section, we comment on approaches in the literature to deal with control of difficulty and highlight the gaps we aim our method to fill. Before proceeding to these sections, it is important to make a little elucidation regarding the relevance of our work into the field. In our work, what we generate are not precisely questions, but mathematical objects that lecturers use to make questions to their students. Let c be an output of the tool. We may assume that what the tool actually generates are questions in the format of "What could be a proof for c?". This allows our tool to be considered an AQG tool.

2.1 Why control the level of difficulty?

According to its objective, an assessment can be classified as formative, summative and evaluative [28]: formative assessments are focused on providing ongoing feedback; summative assessments are those delivered at the end of a course (or a module of it) to certify a student has reached the knowledge expected to finish the course; and evaluative

assessments aim to measure the quality of educational institutions. From different perspectives, all of them aim to obtain information to aid in distinct processes of decision making. So, control of difficulty is a crucial component for the success of assessments as these pieces of information guarantee the coherence of the results according to the proficiency expected from their test-takers. For example, in a summative test with uncontrolled difficulty, we may either harm students by delivering an assessment that it is harder than it should be or inappropriately approve a student on a course.

Especially concerning formative assessments, control of difficulty also helps in dealing with the heterogeneity intrinsic to learning environments, since students may come from different backgrounds and have different learning paces. AQG tools with mechanisms for difficulty control can help tutors both on the construction of personalized formative assessments that match the mastery level of each student and on the development of computerized adaptive tests [3].

On the topic of individualized assessments, it is worth mentioning that they are an effective way of mitigating plagiarism too [17]. In this case, the plagiarism is the traditional one, in which a student share their solutions with another students. By providing tools capable of generating exercises with a comparable difficulty, test designers are able to construct summative assessments that are not at risk of benefiting some students by assigning to them easier questions than to their colleagues.

2.2 How to control the level of difficulty?

In spite of the previously mentioned lack of AQG techniques with controlled level of difficulty, some domain-independent approaches have already been proposed in the literature. Regarding structured knowledge sources, [15] proposes an ontology-based measure with a technique that can be adapted to other ontology-based AQG tools. However, it does not fit our domain as Description Logics [4], the formalisms employed to reason about ontologies, are not as expressive as First-Order Logics.

For unstructured knowledge sources, Question Difficulty Estimation from Text (QDET) [5] [2] is a recent growing trend in Natural Language Processing in order to automate the process of estimating difficulty of questions. In QDET, the usual approach is to provide a machine learning model that predicts the difficulty of a previously classified question.

[3] Computerized adaptive tests are dynamic personalized assessments that pick the next question to be solved based on the performance of the test-taker in the previous questions.

A further strategy adopted in [13] and [12] is to define a set of parameters that affect the difficulty of the question (e.g. where the answer is located in the text in a reading-comprehension question) and provide this set to the model as an input too. Besides also not fitting our purposes, one drawback of QDET is the fact that the classification of difficulty done by these models may not be explained.

Given the aforementioned limitations for these domain-independent mechanisms, we comment on some contributions that are more closely related to our scope and that provide explainable mechanisms to control the difficulty of questions. A metric employed to classify the difficulty of a logic exercise is its syntactic structure, as in [25] and [22]. In both works, the authors present methods to generate formulas of Propositional Logics using the number of propositional atoms or connectives and the depth of the formulas considering their abstract representation [4] as some of the parameters to adjust the complexity of the exercises they produce. An argument to justify the adoption of these parameters can be their influence on the size of the truth-table of the exercise, but this does not hold for First-Order Logics. More importantly, two provable propositional formulas φ_1 and φ_2 may have the same syntactic structure, but the structure of the proofs for them may be very different. So, students might have more difficulty to prove φ_1 than to prove φ_2 or vice versa. Concrete examples of this are presented in Section 4.1. We say that tools such as [25] and [22] adopt a priori metrics of control of difficulty because they only take into account factors that precede and are not influenced by the solution(s) of the produced exercises.

Syntactic similarity is again the strategy employed in [23] to produce proof exercises with a comparable level of difficulty, but, in this case, the scope is Algebra. This system allows the user to add constraints during the process of generation as a mechanism to control of difficulty. For example, if the input exercise is an equality that has an irreducible fraction in its right-hand side, the user can constraint the method to generate only equality exercises whose right-hand sides are irreducible fractions. In some cases, the authors argue that the exercises generated by their method may be proved with "similar proof strategies". However, no further detail on this is provided.

In [29], the implemented tool works via a formal grammar so as to generate individualized proof exercises of equivalences for Classical Propositional Logic. In the beginning of the procedure, based on the personal information of the student, a hash is generated and

[4] As opposed to the concrete representation, which is usually done via sequences of symbols, the abstract representation does not hide the internal structure of the mathematical object being expressed. The abstract representation of a formula φ of propositional logic, for example, is a tree whose root is the main connective of φ.

it is used as a random seed along the procedure to decide which rule is going to be applied at each step of the production. The rules of the grammar are divided into three levels of difficulty (easy, medium and hard). Then, the user can choose how many rules of each level they want the tool to choose during the production of the exercises. The classification of the rules into levels of difficulty is still intuitive and the authors do not show any result proving this intuition is reflected in the performances of the students. Moreover, attributing a general classification of levels of difficulty may fall into two dilemmas: even experts may disagree on such classification; and the difficulty a student with a certain topic is affected by the way they are taught, so what is easy for a student might not be easy for another student from another context.

In [26], the authors present a method capable of producing Mathematical Word Problems[5] whose level of difficulty can be controlled by the amount of arithmetic operations in the equation that solves the problem. The results of an experiment conducted with a group of 30 students revealed that: adding the amount of operations in an exercise increases its error rate, indicating that it becomes more difficult; and exercises with the same amount of operations had indistinguishable error rates. Even being a different type of question, these results give us an impression that considering the size of the solution involved could be more appropriate to define the difficulty of an exercise under the scope of a formal subject such as Logic.

We summarize the main challenges in controlling difficulty of the tools described in this section into three kinds:

1. A priori metrics: The level of difficulty attributed to an exercise does not reflect the effort one puts into solving it.

2. Subjectivity: Classification of difficulty by experts may have inconsistencies.

3. Lack of adaptability: Providing a general classification of difficulty to exercises reduces the control the user has of the tool as the difficulty of an exercise cannot be adapted to the pedagogical context of usage. For example, a user might request exercises of a medium level of difficulty to an AQG tool and receive exercises that for their students are supposed to be easy.

The development of our method aims to meet these challenges. In Chapter 3, we set all the theoretical background necessary to, in Chapter 4, present the method.

[5] A Mathematical Word Problem is an exercise constituted by a short narrative containing unknown values to be calculated using only the information provided in such narrative.

3 Theoretical Background

For pedagogical motivations, the measurement of proving complexity we are going to explore in Chapter 4 relies on formal proofs that do not contain logical symbols. In this chapter, we present how we can produce an analytic proof system for Classical First-Order Logic that allows us to construct formal proofs in a first-order language containing no logical symbols and that enjoys some properties of interest for automated theorem proving.

This chapter is divided into two sections: the Section 3.1 presents some preliminary definitions employed throughout the work; and, in the Section 3.2, we present the procedure that extracts the rules of our envisaged proof system.

3.1 Preliminaries

In this section, we define basic concepts that are used throughout the work. The definitions are divided into three subsections: syntax, deduction and semantics.

3.1.1 Syntax

Recall that the goal of this chapter is to present a proof system for Classical First-Order Logic in which the proofs for a restricted language do not contain logical symbols. Such language restriction is based on what we will call, from the end of this subsection on, a *definitional theory*. We start this subsection by presenting some standard syntactic definitions. In the end, we present two case studies definitional theories for which the main results are exhibited in Section 3.2.

Definition 3.1 (First-order signature)**.** A *first-order signature* (or simply signature) Σ is defined as an ordered pair $\langle FS, PS\rangle$, where $FS := \{FS_i\}_{i\in\mathbb{N}}$ is the family of *function symbols* and $PS := \{PS_i\}_{i\in\mathbb{N}}$ is the family of *predicate symbols* of the language. For each $i \in \mathbb{N}$, FS_i represents the set of function symbols of arity i and PS_i represents the set of

predicate symbols of arity i. The set FS_0 is the set of *constant symbols* of the signature. In a first-order signature, there is no symbol that is both a function and a predicate symbol, that is, $\bigcup_{i\in\mathbb{N}} FS_i \cap \bigcup_{i\in\mathbb{N}} PS_i = \emptyset$, and $\bigcup_{i\in\mathbb{N}} PS_i$ is non-empty.

Definition 3.2 (Terms)**.** Given a denumerable set of variables X and a first-order signature $\Sigma := \langle FS, PS\rangle$, the set of *terms* $Tm_\Sigma(X)$ is inductively defined as:

$$t ::= x \mid c \mid f(t_1, \ldots, t_n)$$

where $x \in X$, $c \in FS_0$, $t_1, \ldots, t_n \in Tm_\Sigma(X)$ and $f \in FS_n$.

Definition 3.3 (First-order language)**.** Given a denumerable set of variables X and a first-order signature $\Sigma := \langle FS, PS\rangle$, the set of *formulas*, or the *language*, $L_\Sigma(X)$ is inductively defined as:

$$\varphi ::= p(t_1, \ldots, t_n) \mid \neg\varphi_1 \mid (\varphi_1 \copyright \varphi_2) \mid (Qx)\varphi_1$$

where $p \in PS_n$, $t_1, \ldots, t_n \in Tm_\Sigma(X)$, $\copyright \in \{\wedge, \vee, \rightarrow\}$, $\varphi_1, \varphi_2 \in L_\Sigma(X)$, $Q \in \{\forall, \exists\}$ and $x \in X$.

Based on its format, a formula $\varphi \in L_\Sigma(X)$ may fall into one of the four groups:

- *Atomic formula* if $\varphi := p(t_1, \ldots, t_n)$ for any $p \in PS_n$.
- *Negated formula* if $\varphi := \neg\varphi_1$ for any $\varphi_1 \in L_\Sigma(X)$.
- *Binary formula* if $\varphi := \varphi_1 \copyright \varphi_2$ for any $\varphi_1, \varphi_2 \in L_\Sigma(X)$ and $\copyright \in \{\wedge, \vee, \rightarrow\}$.
- *Quantified formula* if $\varphi := (Qx)\varphi_1$ for any $\varphi_1 \in \{L\}_\Sigma(X)$, $Q \in \{\forall, \exists\}$ and $x \in X$.

When there is no risk of ambiguity, we omit the first-order signature and the set of variables, denoting a language $L_\Sigma(X)$ simply by L.

In this study, we use two syntactic abbreviations:

- $\varphi_1 \leftrightarrow \varphi_2 := (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$.
- $(Qx_1 \ldots x_n)\varphi := (Qx_1) \ldots (Qx_n)\varphi$, where $Q \in \{\forall, \exists\}$.

The set $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ is the set of *connectives* and the set $\{\forall, \exists\}$ is the set of *quantifiers*. Connectives and quantifiers are the *logic symbols*.

Definition 3.4 (Theory-specific language)**.** Given a language $L_\Sigma(X)$, where $\Sigma := \langle FS, PS\rangle$, the *theory-specific language*, $tsl(L_\Sigma(X))$, is the set of atomic formulas of $L_\Sigma(X)$.

We, now, present three definitions involving variables on terms and formulas.

Definition 3.5 (Variables of a term)**.** Given a set of terms $Tm_\Sigma(X)$ and a $t \in Tm_\Sigma(X)$, the set of *variables* $Var(t)$ of t is inductively defined as:

- $Var(c) = \emptyset$ if c is a constant symbol.
- $Var(x) = \{x\}$ if $x \in X$.
- $Var(f(t_1, \ldots, t_n)) = \bigcup_{1 \leq i \leq n} Var(t_i)$ if f is a n-ary function symbol of the language for any $n > 0$.

A term t is called *closed* if $Var(t) = \emptyset$ and is called *open* otherwise.

Definition 3.6 (Variables of a formula)**.** Given a language $L_\Sigma(X)$ and a $\varphi \in L_\Sigma(X)$, we can inductively extend the definition of sets of variables to formulas via the following clauses:

- $Var(p) = \emptyset$ if p is a 0-ary predicate symbol of the language.
- $Var(p(t_1, \ldots, t_n)) = \bigcup_{1 \leq i \leq n} Var(t_i)$ if $p \in PS_n$ for any $n > 0$.
- $Var(\neg\varphi) = Var(\varphi)$.
- $Var(\varphi_1 © \varphi_2) = Var(\varphi_1) \cup Var(\varphi_2)$ if $© \in \{\wedge, \vee, \rightarrow\}$.
- $Var((Qx)\varphi) = Var(\varphi) \cup \{x\}$ if $Q \in \{\forall, \exists\}$.

Definition 3.7 (Free variables)**.** Given a language $L_\Sigma(X)$ and a $\varphi \in L_\Sigma(X)$, the set of *free variables*, $Free\_var(\varphi)$, of φ is inductively defined as:

- $Free\_var(p) = \emptyset$ if p is a 0-ary predicate symbol of the language.
- $Free\_var(p(t_1, \ldots, t_n)) = \bigcup_{1 \leq i \leq n} Var(t_i)$ if $p \in PS_n$ for any $n > 0$.
- $Free\_var(\neg\varphi) = Free\_var(\varphi)$.
- $Free\_var(\varphi_1 © \varphi_2) = Free\_var(\varphi_1) \cup Free\_var(\varphi_2)$ if $© \in \{\wedge, \vee, \rightarrow\}$.
- $Free\_var((Qx)\varphi) = Free\_var(\varphi) \setminus \{x\}$ if $Q \in \{\forall, \exists\}$.

A formula φ is called *closed* if $Free\_var(\varphi) = \emptyset$ and is called *open* otherwise.

Next, we define the notion of substitution of a variable for a term on a formula. We start by showing how a substitution of variables is defined, then we extend it to terms and formulas.

Definition 3.8 (Substitution of variables)**.** Given a set of terms $Tm_\Sigma(X)$, a *substitution* of $Tm_\Sigma(X)$ is any function $\sigma^{Tm} : X \to Tm_\Sigma(X)$.

Example 3.9. In this example, we define a substitution and a language that are used to illustrate some of the remaining concepts of this subsection. Let $\Sigma := \langle FS, PS \rangle$ be a signature defined as:

- The only constant symbol of Σ is "c_s".
- The only unary function symbol of Σ is "f_s".
- FS_i is empty for any $i \geq 2$.
- The only unary predicate symbol of Σ is "P_s".
- PS_i is empty for any $i \neq 1$.

In the remainder of this subsection, we refer to the language $L_\Sigma(X)$ as L_s.

Now, we define a substitution σ of $Tm_\Sigma(X)$ as:

- $\sigma(x) = c_s$.
- $\sigma(y) = f_s(c_s)$.
- $\sigma(z) = f_s(f_s(c_s))$ for any $z \in X \setminus \{x, y\}$.

Definition 3.10 (Substitution on terms)**.** Given a substitution σ^{Tm} on $Tm_\Sigma(X)$, we recursively extend the definition of substitution to terms by:

- $\sigma^{Tm}(c) = c$ if c is a constant symbol.
- $\sigma^{Tm}(f(t_1, \ldots, t_n)) = f(\sigma^{Tm}(t_1), \ldots, \sigma^{Tm}(t_n))$ if f is a n-ary function symbol of the language for any $n > 0$.

Definition 3.11. Given a substitution σ^{Tm} on $Tm_\Sigma(X)$ and a set $Var \subseteq X$, the substitution σ^{Tm}_{Var} on $Tm_\Sigma(X)$ is defined as $\sigma^{Tm}_{Var}(x) = x$ if $x \in Var$ and $\sigma^{Tm}_{Var}(y) = \sigma^{Tm}(y)$ otherwise.

Consider the substitution σ from Example 3.9. For a set $Var = \{x\}$, we have that $\sigma^{Tm}_{Var}(x) = x$ and $\sigma^{Tm}_{Var}(y) = \sigma^{Tm}(y) = f_s(c_s)$.

Definition 3.12 (Substitution on formulas)**.** Given a substitution σ^{Tm}_{Var} on $Tm_\Sigma(X)$, we recursively define $\sigma_L : L_\Sigma(X) \times 2^X \to L_\Sigma(X)$ by:

- $\sigma_L(p, Var) = p$ if p is a 0-ary predicate symbol of the language.
- $\sigma_L(p(t_1, \ldots, t_n), Var) = p(\sigma^{Tm}_{Var}(t_1), \ldots, \sigma^{Tm}_{Var}(t_n))$ if $p \in PS_n$ for any $n > 0$.
- $\sigma_L(\neg\varphi, Var) = \neg\sigma_L(\varphi, Var)$.
- $\sigma_L(\varphi_1 \copyright \varphi_2, Var) = \sigma_L(\varphi_1, Var) \copyright \sigma_L(\varphi_2, Var)$ if $\copyright \in \{\wedge, \vee, \to\}$.
- $\sigma_L((Qx)\varphi, Var) = (Qx)\sigma_L(\varphi, Var \cup \{x\})$ if $Q \in \{\forall, \exists\}$.

We may refer to σ_L simply by σ. Moreover, we can use the notation $\varphi(t_1, \ldots, t_n) := \sigma(\varphi, \emptyset)$, where x_i is the i-th free variable to appear in φ considering its concrete representation and $\sigma^{Tm}(x_i) = t_i$ for each $1 \leq i \leq n$. Recall that the concrete representation of a formula is the representation of its symbols in a sequence.

Using the substitution σ and the language L_s from Example 3.9, by the previous definition, we have that $\sigma_{L_s}((\forall x)(P_s(x) \to P_s(y)), \emptyset) = (\forall x)(\sigma_{L_s}(P_s(x) \to P_s(y), \{x\})) = (\forall x)(\sigma_{L_s}(P_s(x), \{x\}) \to \sigma_{L_s}(P_s(y), \{x\})) = (\forall x)\,(P_s(\sigma^{Tm}_{\{X\}}(x)) \to P_s(\sigma^{Tm}_{\{X\}}(y))) = (\forall x)(P_s(x) \to P_s(f_s(c_s)))$. Thus, $\sigma_{L_s}((\forall x)(P_s(x) \to P_s(y)), \emptyset) = (\forall x)(P_s(x) \to P_s(f_s(c_s)))$. In other words, substitutions do not affect quantifications.

Now, let σ be an arbitrary substitution and d be a constant symbol. If $\varphi := (\forall x)(P(x) \to P(y))$, then $\varphi(d) = (\forall x)(P(x) \to P(d))$.

Definition 3.13 (Unifiable closed formulas)**.** Given a language $L_\Sigma(X)$ and a formula $\varphi \in L_\Sigma(X)$, we define the set $Unif(\varphi)$ of *unifiable closed formulas* of φ as:

- $Unif(\varphi) := \{\varphi\}$ if φ contains no free variables.
- $Unif(\varphi) := \{\psi \in L_\Sigma(X) \mid \psi$ is a closed formula and there exists $t_1, \ldots, t_n \in Tm_\Sigma(X)$ such that $\psi = \varphi(t_1, \ldots, t_n)\}$ if φ contains n free variables.

For the language L_s from Example 3.9, we have that $Unif(P_s(y)) = \{P(c_s), P(f_s(c_s)), P(f_s(f_s(c_s))), \ldots\}$.

Definition 3.14 (Subformulas)**.** Given a language $L_\Sigma(X)$ and a formula $\varphi \in L_\Sigma(X)$, we recursively define the set of *subformulas* $Subf(\varphi)$ of φ as:

- $Subf(p) = \{p\}$ if p is a 0-ary predicate symbol of the language.
- $Subf(p(t_1, \ldots, t_n)) = Unif(p(t_1, \ldots, t_n))$ if $p \in PS_n$ for $n > 0$.
- $Subf(\neg\varphi) = Unif(\neg\varphi) \cup Subf(\varphi)$. In this case, we say that ψ is an *immediate subformula* of $\neg\varphi$ if $\psi \in Unif(\varphi)$.
- $Subf(\varphi_1 © \varphi_2) = Unif(\varphi_1 © \varphi_2) \cup Subf(\varphi_1) \cup Subf(\varphi_2)$ if $© \in \{\wedge, \vee, \rightarrow\}$. In this case, we say that ψ is an *immediate subformula* of $\varphi_1 © \varphi_2$ if $\psi \in Unif(\varphi_1) \cup Unif(\varphi_2)$.
- $Subf((Qx)\varphi) = Unif((Qx)\varphi) \cup Subf(\varphi)$ if $Q \in \{\forall, \exists\}$. In this case, we say that ψ is an *immediate subformula* of $(Qx)\varphi$ if $\psi \in Unif(\varphi)$.

If ψ is a subformula of φ but not equal to φ, then ψ is said to be a *proper subformula* of φ. Moreover, if ψ is a proper subformula of φ and is not a proper subformula of any other subformula of φ, then ψ is a *direct subformula* of φ. Finally, if ψ is a subformula of φ, we may also say that ψ *occurs in* φ.

Considering the language L_s from Example 3.9, we may say that $P_s(f_s(c_s))$ is a proper subformula, but not a direct subformula of $(P_s(f_s(c_s)) \wedge P_s(c_s)) \vee P_s(c_s)$. However, $P_s(f_s(c_s))$ is a direct subformula of $(P_s(f_s(c_s)) \wedge P_s(c_s)) \vee P_s(y)$ since $P_s(f_s(c_s)) \in Unif(P_s(y))$.

Definition 3.15 (Definitional theory)**.** A *definitional theory* is a tuple $Th := \langle L_\Sigma(X), Ax\rangle$, such that $L_\Sigma(X)$ is a first-order language and $Ax \subseteq L_\Sigma(X)$ is a finite set of closed formulas of $L_\Sigma(X)$ to which we refer as *definitional axioms*.

Now, we start presenting two examples of the definitional theories we are going to explore in this work as case studies of our method. The presentation of these two definitional theories is going to be finished only in Section 3.1.2, when we present a necessary adjustment for them to work correctly for our proof-theoretical purposes.

Example 3.16. The first definitional theory we work with is the *theory of sets*. The signature $\Sigma := \langle FS, PS\rangle$ is defined as:

- The only constant symbol of Σ is "$\emptyset$".

- The only unary function symbols of Σ are "$\complement$", "fst" and "snd".
- The only binary function symbols of Σ are "$\cup$", "$\cap$", "$\setminus$", "$\triangle$", "$\times$".
- FS_i is empty for any $i \geq 3$.
- The only binary predicate symbols of Σ are "$\in$", "$\subseteq$", and "$\between$".
- PS_i is empty for any $i \neq 2$.

For all binary symbols, we adopt the infix notation and omit parentheses when there is no chance of ambiguity. The set of definitional axioms Ax is given by the formulas presented below:

- $[ax\emptyset]$ Definitional axiom for $\emptyset$: $(\forall x)(\neg x \in \emptyset)$.
- $[ax\complement]$ Definitional axiom for $\complement$: $(\forall x, y)(x \in \complement(y) \leftrightarrow \neg x \in y)$.
- $[ax\cup]$ Definitional axiom for $\cup$: $(\forall x, y, z)(x \in y \cup z \leftrightarrow (x \in y \vee x \in z))$.
- $[ax\cap]$ Definitional axiom for $\cap$: $(\forall x, y, z)(x \in y \cap z \leftrightarrow (x \in y \wedge x \in z))$.
- $[ax\setminus]$ Definitional axiom for $\setminus$: $(\forall x, y, z)(x \in y \setminus z \leftrightarrow (x \in y \wedge \neg x \in z))$.
- $[ax\times]$ Definitional axiom for $\times$: $(\forall x, y, z)(x \in y \times z \leftrightarrow (fst(x) \in y \wedge snd(x) \in z))$.
- $[ax\triangle]$ Definitional axiom for $\triangle$: $(\forall x, y, z)(x \in y \triangle z \leftrightarrow ((\neg x \in y \rightarrow x \in z) \wedge (x \in y \rightarrow \neg x \in z)))$.
- $[ax \subseteq]$ Definitional axiom for $\subseteq$: $(\forall x, y)(x \subseteq y \leftrightarrow (\forall z)(z \in x \rightarrow z \in y))$.
- $[ax \between]$ Definitional axiom for $\between$: $(\forall x, y)(x \between y \leftrightarrow (\forall z)\neg(z \in x \wedge z \in y))$.

The only definitional axiom in which the unary symbols fst and snd are present is in $[ax\times]$. Intuitively, we want $x \in y \times z$ to be the case when x is a pair (a, b) such that $a \in y$ and $b \in z$. To avoid axioms concerning pairs, we use fst to represent the first element of a pair and snd to represent the second element of a pair.

Example 3.17. The second definitional theory is the definitional *theory of numbers*. The signature $\Sigma := \langle FS, PS \rangle$ is defined as:

- The only binary function symbols of Σ are "$+$" and "$\cdot$"
- FS_i is empty for any $i \neq 2$.

- The only binary predicate symbols of Σ is "$\approx$", "$|$" and "$\leq$".
- PS_i is empty for any $i \neq 2$.

Again, we adopt infix notation for binary symbols and omit parentheses when there is no chance of ambiguity. The set of definitional axioms Ax is given by the formulas presented below:

- $[ax{+}_{com}]$ Definitional axiom of the commutativity of $+$: $(\forall x, y, z)(x \approx y + z \rightarrow x \approx z + y)$.
- $[ax{+}_{inv}]$ Definitional axiom of the inverse of $+$: $(\forall x, y, z)(x + y \approx x + z \rightarrow y \approx z)$.
- $[ax{+}_{cong}]$ Definitional axiom of the congruentiality of $\approx$ with respect to $+$: $(\forall x, w, y, z)((x \approx w \wedge y \approx z) \rightarrow (x + y \approx w + z))$.
- $[ax{\cdot}_{com}]$ Definitional axiom on the commutativity of $\cdot$: $(\forall x, y, z)(x \approx y \cdot z \rightarrow x \approx z \cdot y)$.
- $[ax{\cdot}_{inv}]$ Definitional axiom of the inverse of $\cdot$: $(\forall x, y, z)(x \cdot y \approx x \cdot z \rightarrow y \approx z)$.
- $[ax{\cdot}_{cong}]$ Definitional axiom on the congruentiality of $\approx$ with respect to $\cdot$: $(\forall x, w, y, z)$ $((x \approx w \wedge y \approx z) \rightarrow (x \cdot y \approx w \cdot z))$.
- $[ax \leq]$ Definitional axiom for $\leq$: $(\forall x, y)(x \leq y \leftrightarrow (\exists z)(y \approx x + z))$.
- $[ax \mid]$ Definitional axiom for $|$: $(\forall x, y)(x \mid y \leftrightarrow (\exists z)(y \approx x \cdot z))$.

One could wonder where does these definitional axioms come from. They are formulated so we are able to derive a fragment of theorems from branches of Mathematics with educational interest, as Set Theory and Number Theory are. But why the definitional axiom for $\times$ is not the formula $(\forall x_1, x_2, y, z)((x_1, x_2) \in y \times z \leftrightarrow (x_1 \in y \wedge x_2 \in z))$? Or why the definitional axiom on the commutativity of $+$ is not defined as $(\forall x, y)(x + y \approx y + x)$? As we detail in Chapter 4, the notion of comparability between proofs we provide is based on the isomorphism of proofs that do not contain logical symbols. Such proofs, in turn, are constructed using rules extracted from these definitional axioms. By describing definitional axioms whose atomic subformulas are isomorphic, we are more likely to have isomorphisms between the proofs.

3.1.2 Deduction

In this study, the deductive system we work with is a system proposed by D'Agostino and Mondadori in [7, 8], which we refer to as *cut-based tableaux*. Cut-based tableaux were proposed as an alternative to Smullyan's tableaux [24] having one of the motivations the fact that the latter are not generally as computationally efficient as truth-tables. The witnesses of this inefficiency are what D'Agostino and Mondadori define as *truly fat* tautotlogies, formulas of size n whose propositional variables are of order $log(n)$.

To exemplify this, consider the set of propositional variables[1] $\mathbb{P}_k := \{P_1, \ldots, P_k\}$. A formula in the set $D(\mathbb{P}_k)$ is a disjunction of literals[2] such that either P_i or $\neg P_i$ is a subformula of $D(\mathbb{P}_k)$ for any $i \in \{1, \ldots, k\}$. Finally, the truly fat formula $F(\mathbb{P}_k)$ is defined as the conjuction of all 2^k possible formulas from $D(\mathbb{P}_k)$. For $\mathbb{P}_2$, the formula $F(\mathbb{P}_2)$ is $(P_1 \vee P_2) \wedge (P_1 \vee \neg P_2) \wedge (\neg P_1 \vee P_2) \wedge (\neg P_1 \vee \neg P_2)$. In this case, while one can verify that $\mathbb{P}_k$ is a tautology with a truth-table of 2^k lines, a Smullyan's tableau with $k!$ branches would be necessary to accomplish this verification. Such inefficiency was found to be caused by the redundancy that bifurcations introduce in the construction of a tableau proof. The solution proposed, as we present in Definition 3.21, was to reformulate the rules about the logical connectives and only allow bifurcations via the application of an analytic form of cut.

The approach used in cut-based tableaux to overcome the computational flaw in Smullyan's tableaux might be replicated in other deductive systems, such as natural deduction. However, we decided to adopt tableaux as the formalism to address the problem of generating conjectures with a comparable level of proving complexity due to its advantage in supporting the extraction of countermodels for refutable conjectures, as in [24]. Despite the scope of this work being provable conjectures, we aim to adapt our methodology also to refutable conjectures in a future work.

Before setting the definitions about tableaux, we first present an important definition concerning the connection between informal and formal proofs. In informal mathematical proof practice, we usually introduce names for objects that will no longer be necessary when a proof is finished and hence are not part of the formal language employed in that proof. For example, if the goal is to prove that a unary property P holds for every object of a context C, we start the proof by writing "Let x be an arbitrary object of C" and proceed by proving that P holds for x. In formal proofs we are about to present, the name

[1] In first-order logic, a propositional variable may be treated as a 0-ary predicate symbol.

[2] A literal is a formula that is either a propositional variable or the negation of a propositional variable.

of such an "x" is called a parameter and we treat it as a constant symbol so, in semantics, interpretation structures alone suffice to provide the meaning of formulas containing no quantifiers.

Definition 3.18 (Parameters)**.** Given a language $L_\Sigma(X)$, a set of *parameters* Par for $L_\Sigma(X)$ is a countable set of constant symbols that are not constant symbols of Σ. In the examples of formal proofs that we present, parameters are denoted by p_i, where $i \in \mathbb{N}$. We use $L_\Sigma^{Par}(X)$, where $\Sigma := \langle FS, PS \rangle$, to denote the language in which the parameters of Par are also considered as constant symbols. When there is no risk of ambiguity, we also denote $L_\Sigma^{Par}(X)$ simply by $L_\Sigma(X)$.

Now, the presentation of the definitional theory of sets and numbers that we started at the end of 3.1.1 may be finished. In both cases, for a set of parameters Par, the language of the definitional theories is $L_\Sigma^{Par}(X)$ instead of $L_\Sigma(X)$.

Definition 3.19 (Tableau nodes)**.** In our tableaux trees, the nodes should be of one of the two types:

- Signed formulas: First-order formulas preceded either by a "$+$" sign (positive node) or by a "$-$" sign (negative node). In [7, 8], "t" and "f" are used instead of, respectively, "$+$" and "$-$".
- Fresh parameter expressions: Expressions in the format "`Fresh` p_i" to emphasize that the parameter p_i is not used in any formula of any node above in that branch. We refer to these nodes as "fresh parameter nodes".

Definition 3.20 (Conjugate signed formulas)**.** Given a signed formula sf, we refer to the sign of sf by $sign(sf)$ and the formula of sf by $fmla(sf)$. Two signed formulas sf_1 and sf_2 are *conjugates* (or sf_1 is the conjugate of sf_2) if $sign(sf_1) \neq sign(sf_2)$ and $fmla(sf_1) = fmla(sf_2)$. The conjugate of a signed formula sf may be denoted as sf^c. A contradiction, then, will be raised in a branch whenever there are two conjugate signed formulas on it and such branch closes.

Let φ_1 and φ_2 be two first-order formulas. Then, the signed formulas $+\,\varphi_1 \wedge \varphi_2$ $-\,\varphi_1 \wedge \varphi_2$ are conjugate signed formulas.

Definition 3.21 (Expansion rules)**.** The following expansion rules are the ones presented in [7]:

Negation rules

$$\frac{+\,\neg\varphi}{-\,\varphi}\ +\neg\mathbf{E} \qquad \frac{-\,\neg\varphi}{+\,\varphi}\ -\neg\mathbf{E} \qquad \frac{+\,\varphi}{-\,\neg\varphi}\ +\neg\mathbf{I} \qquad \frac{-\,\varphi}{+\,\neg\varphi}\ -\neg\mathbf{I}$$

Disjunction rules

$$\frac{+\,\varphi_1 \vee \varphi_2 \quad -\,\varphi_1}{+\,\varphi_2}\ +\vee\mathbf{E}_1 \qquad \frac{+\,\varphi_1 \vee \varphi_2 \quad -\,\varphi_2}{+\,\varphi_1}\ +\vee\mathbf{E}_2 \qquad \frac{-\,\varphi_1 \vee \varphi_2}{-\,\varphi_1 \quad -\,\varphi_2}\ -\vee\mathbf{E}$$

$$\frac{+\,\varphi_1}{+\,\varphi_1 \vee \varphi_2}\ +\vee\mathbf{I}_1 \qquad \frac{+\,\varphi_2}{+\,\varphi_1 \vee \varphi_2}\ +\vee\mathbf{I}_2 \qquad \frac{-\,\varphi_1 \quad -\,\varphi_2}{-\,\varphi_1 \vee \varphi_2}\ -\vee\mathbf{I}$$

Conjunction rules

$$\frac{-\,\varphi_1 \wedge \varphi_2 \quad +\,\varphi_1}{-\,\varphi_2}\ -\wedge\mathbf{E}_1 \qquad \frac{-\,\varphi_1 \wedge \varphi_2 \quad +\,\varphi_2}{-\,\varphi_1}\ -\wedge\mathbf{E}_2 \qquad \frac{+\,\varphi_1 \wedge \varphi_2}{+\,\varphi_1 \quad +\,\varphi_2}\ +\wedge\mathbf{E}$$

$$\frac{-\,\varphi_1}{-\,\varphi_1 \wedge \varphi_1}\ -\wedge\mathbf{I}_1 \qquad \frac{-\,\varphi_2}{-\,\varphi_1 \wedge \varphi_1}\ -\wedge\mathbf{I}_2 \qquad \frac{+\,\varphi_1 \quad +\,\varphi_2}{+\,\varphi_1 \wedge \varphi_2}\ +\wedge\mathbf{I}$$

Implication rules

$$\frac{+\,\varphi_1 \to \varphi_2 \quad +\,\varphi_1}{+\,\varphi_2}\ +\to\mathbf{E}_1 \qquad \frac{+\,\varphi_1 \to \varphi_2 \quad -\,\varphi_2}{-\,\varphi_1}\ +\to\mathbf{E}_2 \qquad \frac{-\,\varphi_1 \to \varphi_2}{+\,\varphi_1 \quad -\,\varphi_2}\ -\to\mathbf{E}$$

$$\frac{-\,\varphi_1}{+\,\varphi_1 \to \varphi_2}\ +\to\mathbf{I}_1 \qquad \frac{+\,\varphi_2}{+\,\varphi_1 \to \varphi_2}\ +\to\mathbf{I}_2 \qquad \frac{+\,\varphi_1 \quad -\,\varphi_2}{-\,\varphi_1 \to \varphi_2}\ -\to\mathbf{I}$$

Quantifier rules

$$\frac{+\,(\forall x)\varphi}{+\,\varphi(y)}\ +\forall\mathbf{E} \qquad \frac{-\,(\forall x)\varphi}{\texttt{Fresh}\ \ p_i \quad -\,\varphi(p_i)}\ -\forall\mathbf{E} \qquad \frac{+\,(\exists x)\varphi}{\texttt{Fresh}\ \ p_i \quad +\,\varphi(p_i)}\ +\exists\mathbf{E} \qquad \frac{-\,(\exists x)\varphi}{-\,\varphi(y)}\ -\exists\mathbf{E}$$

Cut rule

$$\frac{}{+\,\varphi \mid -\,\varphi}\ \mathit{cut}$$

φ is a subformula of a formula in a node
above the application of the cut rule

We may refer to this set of rules as `CB`, which stands for "cut-based". For every rule, the nodes above the bar, when it is the case, are the *premises* of the rule and the nodes below the bar are the *conclusions* of the rule. If a rule r contains n premises, we may refer

to it as a *n-premise rule*. If r contains only one conclusion, we may refer to it as a *single conclusion rule*, and as *multiple conclusion rule* otherwise.

It is worth mentioning that every expansion rule r is actually a schema representing all expansion rules that are instances of r. For example, consider a language having P, Q and R as 0-ary predicate symbols. The two rules below are instances of $+\vee\mathbf{E}_1$:

$$\frac{\begin{array}{c}+\neg P \vee \neg Q \\ -\neg P\end{array}}{+\neg Q} \qquad \frac{\begin{array}{c}+P \vee (Q \wedge R) \\ -P\end{array}}{+Q \wedge R}$$

To finish this definition, we have two comments:

1. Usually the *cut* rule is referred as PB (Principle of Bivalence). We adopt the terminology *cut* instead due to is resemblance to the cut rule in Sequent Calculus. Besides, the formula in the conclusion of a *cut* rule is referred as *cut formula*.

2. The introduction rules, labeled with $\mathbf{I}$, are not necessary to reach completeness with respect to the semantic notion of consequence we define in Subsection 3.1.3. However, we consider these introduction rules to be part of `CB` because they play a fundamental role in the procedure presented in Section 3.2.

Definition 3.22 (Uniform notation)**.** Inspired by [24], we use *uniform notation* to refer to signed formulas according to the role they play in `CB` expansion rules. This notation is presented in Tables 1 to 4.

Unlike [24], we do not consider negated formulas as α-nodes. The reason for this is that we want to treat negated formulas differently in the procedure described in Section 3.2.

$\boldsymbol{\alpha}$	$\boldsymbol{\alpha_1}$	$\boldsymbol{\alpha_2}$
$-\,\varphi_1 \vee \varphi_2$	$-\,\varphi_1$	$-\,\varphi_2$
$+\,\varphi_1 \wedge \varphi_2$	$+\,\varphi_1$	$+\,\varphi_2$
$-\,\varphi_1 \rightarrow \varphi_2$	$+\,\varphi_1$	$-\,\varphi_2$

Table 1: Uniform notation for α-nodes

$\boldsymbol{\beta}$	$\boldsymbol{\beta_1}$	$\boldsymbol{\beta_2}$
$+\,\varphi_1 \vee \varphi_2$	$+\,\varphi_1$	$+\,\varphi_2$
$-\,\varphi_1 \wedge \varphi_2$	$-\,\varphi_1$	$-\,\varphi_2$
$+\,\varphi_1 \rightarrow \varphi_2$	$-\,\varphi_1$	$+\,\varphi_2$

Table 2: Uniform notation for β-nodes

γ	$\gamma(y)$
$+\,(\forall x)\varphi(x)$	$+\,\varphi(y)$
$-\,(\exists x)\varphi(x)$	$-\,\varphi(y)$

Table 3: Uniform notation for γ-nodes

δ	Fresh	$\delta(y)$
$-\,(\forall x)\varphi(x)$	Fresh p_i	$-\,\varphi(p_i)$
$+\,(\exists x)\varphi(x)$	Fresh p_i	$+\,\varphi(p_i)$

Table 4: Uniform notation for δ-nodes

Definition 3.23 (Positive and negative occurrences of a subformula on a signed formula)**.** Let sf be a signed formula having as formula φ and $(Qx)\psi$ be a quantified subformula of φ such that there is only a single occurrence of $(Qx)\psi$ in φ. We say that the occurrence of ψ in sf is *negative* when:

- $\varphi = (Qx)\psi$ and $sf = -\,\varphi$.
- $sf = +\,\neg\neg\Phi$ and the occurrence of $(Qx)\psi$ in $+\,\Phi$ is negative or, analogously, $sf = -\,\neg\neg\Phi$ and the occurrence of $(Qx)\psi$ in $-\,\Phi$ is negative.
- sf is an α signed formula and the occurrence of $(Qx)\psi$ in either α_1 or α_2 is negative.
- sf is a β signed formula and the occurrence of $(Qx)\psi$ in either β_1 or β_2 is negative.
- sf is a γ signed formula such that $fmla(sf) = (Qy)\Phi$ and the occurrence of $(Qx)\psi$ in $\Phi(y)$ is negative.
- sf is a δ signed formula such that $fmla(sf) = (Qy)\Phi$ and the occurrence of $(Qx)\psi$ in $\Phi(y)$ is negative.

If the occurrence of $(Qx)\psi$ in sf is not negative, then it is *positive*.

Using the language of the definitional theory of sets from Example 3.16, we may say that the occurrence of $(\forall z)(z \in x \to z \in y)$ in $+\,(\forall x, y)((\forall z)(z \in x \to z \in y) \to x \subseteq y)$ is negative. The reason for this is that $-\,(\forall z)(z \in x \to z \in y)$ is the β_1 for $+\,(\forall z)(z \in x \to z \in y) \to x \subseteq y$ and the occurrence of $(\forall z)(z \in x \to z \in y)$ in $-\,(\forall z)(z \in x \to z \in y)$ is negative.

Definition 3.24 (Application of a rule)**.** Let T be a tree of signed formulas. If there are nodes of T that are premises of a rule r of CB, then the application of r in T is the result of adding the conclusion of r as the leaf node of T. In the case of *cut*, a branch is added

to T and the two new leaf nodes are $+\varphi$ and $-\varphi$, where φ is the cut formula of this application.

Definition 3.25 (Tableaux)**.** Given a finite set $SF := \{s_1, \ldots, s_n\}$ of signed formulas, a CB-tableau for SF is inductively defined as follows:

- A branch containing exactly all nodes of SF is a CB-tableau for SF. We may refer to this CB-tableau as the *initial segment* for SF.
- If T is a CB-tableau for SF and $\mathtt{T}^*$ is the result of applying any of the rules of CB in T, then $\mathtt{T}^*$ is a CB-tableau for SF.

Definition 3.26 (Initial branch)**.** Every CB-tableau starts with a single branch and might be expanded to other branches via application of the *cut* rule. The nodes above the first application of an expansion rule that creates another branch in a CB-tableau form the *initial branch* of such tableau.

Definition 3.27 (Descendant nodes)**.** A node n in a CB-tableau T is a *descendant* of a node m if n was added to T as the result of applying a rule of CB having either m as premise or a node m' that is a descendant of m. In the first case, we may refer to n as a *direct descendant* of m.

Definition 3.28 (Closed/open tableaux)**.** A branch in a CB-tableau is *closed* if it contains two conjugate signed formulas, and it is *open* otherwise. We extend these definitions to CB-tableaux by stating that a CB-tableau is *closed* if all its branches are closed and it is *open* otherwise.

Definition 3.29 (Tableau proofs)**.** A CB-proof of a set of formulas $\Delta := \{\delta_1, \ldots, \delta_n\}$ from a set of formulas $\Gamma := \{\gamma_1, \ldots, \gamma_m\}$ is a closed CB-tableau for $P := \{+\gamma_1, \ldots, +\gamma_m, -\delta_1, \ldots, -\delta_n\}$. Then, Δ is CB-provable from Γ if there is a CB-proof of Δ from Γ. We denote this by $\Gamma \vdash_{\mathtt{CB}} \Delta$ and refer to it as the *consequence relation induced* by CB. Moreover, the set P may be called CB-provable.

For any CB-provability relation $\Gamma \vdash_{\mathtt{CB}} \Delta$, the set of signed formulas Γ is called the *antecedent* of the relation and the set of signed formulas Δ is the *succedent* of the relation.

In the antecedent and in the succedent of a CB-provability relation, we may replace union symbols by commas. So, instead of writing $\Gamma_1 \cup \Gamma_2 \vdash_{\mathtt{CB}} \Delta_1 \cup \Delta_2$ for sets of signed formulas $\Gamma_1, \Gamma_2, \Delta_1$ and Δ_2, we may write $\Gamma_1, \Gamma_2 \vdash_{\mathtt{CB}} \Delta_1, \Delta_2$.

Here is where the motivation to use the signed version of tableaux should be clear. In the unsigned version of tableaux, the definition of a closed branch depends either on the negation or on the atomic connectives $\top$ and $\bot$, which we are not working with. Then, by adopting unsigned tableaux, our objective to produce a proof system whose rules do not contain logical symbols neither in their premises nor in their conclusions would be infeasible.

During the process of constructing a tableau proof, we may apply some expansion rules that, after finishing its construction, are realized to be not necessary to close it. In what follows, we present some definitions about nodes, application of expansion rules and CB-proofs regarding such detours.

Definition 3.30 (Clean proofs)**.** Let P be a CB-proof of SF and n be a signed formula node in a branch θ of P. We say that n is *dispensable* in P (and *indispensable* otherwise) if there is no conjugate node of n in θ and no descendant node of n has a conjugate node in θ. If a rule was either applied in a closed branch or it has added only dispensable nodes in P, we say that this rule application is *eliminable* in P. Finally, if the set of dispensable nodes of P are all in the initial CB-tableau for SF, then we say that P is a *clean* CB-proof. For any non-clean CB-proof P, we refer to the result P' of removing all dispensable node of P that are not in the initial node as the *clean* CB*-proof* for P.

All the definitions from CB-trees to CB-provability depend on the set of expansion rules CB. This allows us to adapt these definitions to any other set of expansion rules ER so we can talk, for example, about ER-proofs.

In this study, every tableau proof is represented by a tree of 3-uples. The first component is the index of the node, whose content is the second component. The third component indicate the indices of the node(s) that justifies the addition of that node(s) in the tree. In the representation of a CB-tree T for SF, if a node was added in T either by 0-premise rule or because it is in SF, then the third component is left blank. If a branch is closed, we add the symbol "$\divideontimes$" at the end of the branch indicating the indices of the nodes that justify such closure.

In Figure 2, we illustrate a CB-tableaux for a conjecture involving a definitional theory whose language contains only the 0-ary predicate symbols P, Q, R and S. Both branches in the CB-tableau T_1 of Figure 2 are closed. Then, T_1 represents a CB-proof of $\{\neg S \to \neg Q\}$ from $\{(P \vee (Q \vee R)) \to (S \wedge P)\}$.

In the remainder of this subsection, we present some lemmas that are helpful in

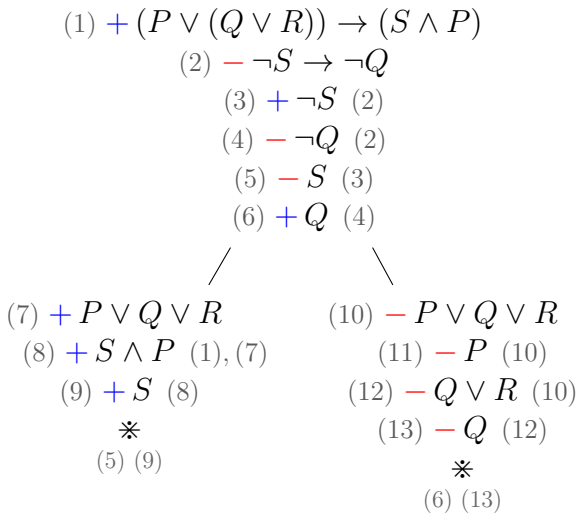


Figure 2: A CB-proof for $\{+\,(P \vee (Q \vee R)) \to (S \wedge P), -\,\neg S \to \neg Q\}$.

Section 3.2.

Definition 3.31 (Derivable rules)**.** Let r_1 the following rule, whose none of the conclusion is a fresh parameter node:

$$\dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \end{array}}{\begin{array}{c} conc_1 \\ \vdots \\ con_m \end{array}}\ r_1$$

We say that r_1 is *derivable* in a set of rules ER when the set $\{prem_1, \ldots, prem_n, conc_1^c, \ldots, conc_m^c\}$ is ER-provable (recall the notation for conjugate signed formulas in Definition 3.20). Moreover, for a rule r_2, if r_1 is derivable in $ER \cup \{r_2\}$ and r_2 is derivable in $ER \cup \{r_1\}$, we say that r_1 and r_2 are *ER-interderivable* or *interderivable* in ER.

These notions can be extended to sets of rules. A set or rules R is derivable in a set of rules ER if every rule $r \in R$ is derivable in ER. The notion of ER-interderivability is analogously extended. This concept of interderivability plays a central role in the procedure presented in Section 3.2.

If two rules r_1 and r_2 are ER-interderivable, the intuition behind is that we have the same deductive strength if we choose to add either r_1 or r_2 in ER. This is what we prove in the following lemma.

Lemma 3.32. For any set of rules ER, and single conclusion rules r_1 and r_2 containing no fresh parameter nodes, if r_1 and r_2 are ER-interderivable rules, then $\vdash_{ER \cup \{r_1\}} = \vdash_{ER \cup \{r_2\}}$

Proof. Let ER be a set of expansion rules and the rules r_1 and r_2 be single conclusion rules containing no fresh parameter nodes such that r_1 and r_2 are ER-interderivable. The rules r_1 and r_2 are specified as follows:

$$\frac{\begin{array}{c} n_1 \\ \vdots \\ n_n \end{array}}{p}\ r_1 \qquad \frac{\begin{array}{c} m_1 \\ \vdots \\ m_m \end{array}}{q}\ r_2$$

We refer to $ER \cup \{r_1\}$ as ER_1 and to $ER \cup \{r_2\}$ as ER_2.

$\subseteq$: Let L be a language and $R, S \subseteq L$ such that $R \vdash_{ER_1} S$, that is, there is an ER_1-proof P of S from R. By definition, there is a ER_2-proof Q of $\{n_1, \ldots, n_n, p^c\}$. We refer to the result of removing the nodes of the initial segment of Q as Q^*.

Now, we show how we can construct an ER_2-proof for S from R. If the rule r_1 was not applied in P, then P is already an ER_2-proof. Otherwise, assume the tree below represents a part of P in which the rule r_1 was applied:

$$\begin{array}{c} (1)\ n_1 \\ \vdots \\ (n)\ n_n \\ (n+1)\ p\ \ (1)\ldots(n) \end{array}$$

Then, we can do the following transformation:

$$\begin{array}{ccc} & (1)\ n_1 & \\ & \vdots & \\ & (n)\ n_n & \\ & (n+1)\ n_n \vee p\ \ (n) & \\ \swarrow & & \searrow \\ (n+2)\ p^c & & (n+3)\ p \\ Q^* & & \vdots \end{array}$$

The left branch is closed and, in the right branch, we replicate the steps of P. So, we have an ER_2-proof.

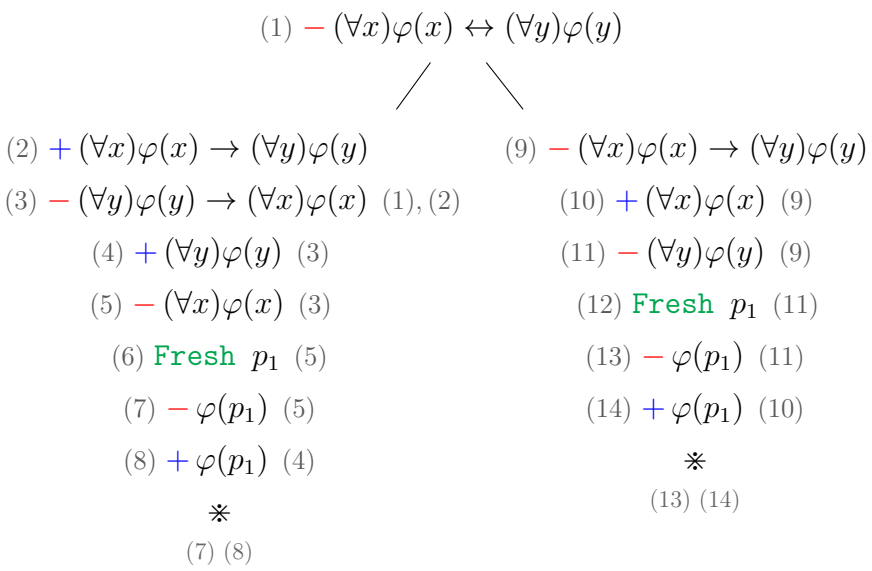
$\supseteq$: Similar. □

Some of the rules extracted in the process presented in Section 3.2 do not have deductive strength. To formally refer to them, we present the following definition.

Definition 3.33 (Redundant rule)**.** Let ER be a set of expansion rules and r be an expansion rule. If r is derivable in ER, then we say that r is a *redundant rule* with respect to ER. If r is derivable in any set of expansion rules, then we say that r is a *general redundant rule*.

Lemma 3.34. Let Q be a quantifier and φ be a formula containing exactly one free variable. Then, $\vdash_{\texttt{CB}} (Qx)\varphi(x) \leftrightarrow (Qy)\varphi(y)$.

Proof. We present the proof for the universal quantifier and the proof for the existential quantifier is similar.



□

3.1.3 Semantics

As usual, to provide an interpretation to a formula, we start by fixing a non-empty set, the domain, and we specify how every term of the language can be associated to an element of the domain. This association is called denotation. Before showing how this denotation can be constructed, we present three definitions.

Definition 3.35 (Interpretation structure)**.** Given a first-order signature $\Sigma := \langle FS, PS \rangle$, an *interpretation structure* IS for Σ is a pair $\langle D, I \rangle$, where D is a non-empty set called the domain of the structure and I is a function called interpretation that associates:

- Every n-ary function symbol $f \in FS$ to a function $I(f) : D^n \to D$.
- Every n-ary relation symbol $P \in PS$ to a relation $I(P) \subseteq D^n$.

For an interpretation function I, instead of using the infix notation, we adopt the postfix superscript notation f^I to represent the denotation of the function symbol f via I. For the case 0-ary predicate symbols, consider that $D^0 := \{\emptyset\}$.

Definition 3.36 (Assignment)**.** Given an interpretation structure $IS := \langle D, I\rangle$, for a set of variables X, an *assignment* for X over IS is defined as a function $\rho : X \to D$.

Definition 3.37 (Model)**.** Given a language $L_\Sigma(X)$, a *model* M for $L_\Sigma(X)$ is a pair $\langle IS, \rho\rangle$, where IS is an interpretation structure for Σ and ρ is an assignment for X over IS.

Definition 3.38 (Denotation of terms)**.** Given a language $L_\Sigma(X)$ and a model $M := \langle IS, \rho\rangle$ for $L_\Sigma(X)$, where $IS := \langle D, I\rangle$, we recursively define the *denotation* function $[\![\circ]\!]^M : Tm_{L_\Sigma(X)} \to D$ as follows:

- $[\![x]\!]^M = \rho(x)$ if $x \in X$.
- $[\![c]\!]^M = c^I$ if c is a constant symbol of Σ.
- $[\![f(t_1, \ldots, t_n)]\!]^M = f^I([\![t_1]\!]^M, \ldots, [\![t_n]\!]^M)$ if $t_1, \ldots, t_n \in Tm_{L_\Sigma(X)}$ and $f \in SF_n$, for $n > 0$.

Definition 3.39 (Assignment variant)**.** Given an interpretation structure $IS := \langle D, I\rangle$, a set of variables X and two assignments ρ and ρ^* for X over I, we say that ρ^* is an x-variant for ρ, where $x \in X$, if $\rho^*(y) = \rho(y)$ for every $y \in X \setminus \{x\}$.

With the definitions of denotation and assignment variant, we are able to provide meaning to formulas.

Definition 3.40 (Satisfiability relation)**.** Let $\mathcal{M}$ be the class of all models. Given a language $L_\Sigma(X)$, we recursively define the relation $\models \subseteq \mathcal{M} \times L_\Sigma(X)$ as:

- $M \models P$ iff $\emptyset \in P^I$, where $P \in PS_0$.
- $M \models P(t_1, \ldots, t_n)$ iff $([\![t_1]\!]^M, \ldots, [\![t_n]\!]^M) \in P^I$, where $P \in PS_n$ for any $n > 0$.
- $M \models \neg\varphi$ iff $M \models \varphi$ does not hold.
- $M \models \varphi_1 \wedge \varphi_2$ iff both $M \models \varphi_1$ and $M \models \varphi_2$.
- $M \models \varphi_1 \vee \varphi_2$ iff either $M \models \varphi_1$ or $M \models \varphi_2$.
- $M \models \varphi_1 \to \varphi_2$ iff $M \models \varphi_1$ implies that $M \models \varphi_2$.

- $\langle IS, \rho \rangle \models (\forall x)\varphi$ iff $\langle IS, \rho^* \rangle \models \varphi$ for every x-variant ρ^* of ρ.
- $\langle IS, \rho \rangle \models (\exists x)\varphi$ iff $\langle IS, \rho^* \rangle \models \varphi$ for some x-variant ρ^* of ρ.

When $M \models \varphi$, we say that "M satisfies φ" and, when $M \models \varphi$ does not hold, we write that $M \not\models \varphi$.

For a set of formulas S, we say that a model M *satisfies* S when M satisfies all signed formulas in S and this is denoted by $M \models S$. Then, we say that a set of signed formulas is *satisfiable* if there is a model that satisfies it.

Definition 3.41 (Semantic consequence relation)**.** Let L be a language and $R, S \subseteq L$. We say that S semantically follows from R if, for every model M of L, $M \models R$ implies that $M \models s$ for some $s \in S$. This is denoted as $R \models S$.

Fact 3.42 (See Section 3.6 of [7])**.** For every language L and $R, S \subseteq L$, $R \vdash_{\mathtt{CB}} S$ iff $R \models S$. In other words, $\vdash_{\mathtt{CB}}$ is sound and complete with respect to $\models$.

Fact 3.43 (See Section 2.5 of [11])**.** Let φ and ψ^* be formulas and ψ be a subformula of φ. If $\models \psi \leftrightarrow \psi^*$ and φ^* is the result of replacing ψ by ψ^* in φ, then $\models \varphi \leftrightarrow \varphi^*$.

3.2 First-order theory-specific cut-based tableaux

In this section, we aim to provide a set of rules that allows us to construct proofs whose nodes do not contain logical symbols. For this, we provide a procedure that extracts a set of what we call *theory-specific rules* TS_{Th} from the axioms of a given definitional theory Th. These rules (i) are linear, (ii) do not contain logical symbols and (iii) allow us to construct proofs for theory-specific formulas from theory-specific formulas such that they do not contain logical symbols in the formulas of their nodes.

The idea of providing inference rules that do not contain logical symbols is not a novelty and it was already explored in [18]. In this work, the authors present a way of translating closed first-order formulas in synthetic rules. However, besides not being analytic, these rules are not cut-based.

The procedure that extract the theory-specific rules is presented in Subsection 3.2.4. Before providing such procedure, we need to preprocess the definitional axioms. This is what we demonstrate in Subsections 3.2.1 to 3.2.3. In Subsection 3.2.5, we present an alternative notion of the cut rule of CB for the proofs constructed with the theory-specific rules.

3.2.1 Definitional axiom rules

Definition 3.44 (Definitional axiom rules)**.** Given a definitional theory $Th = \langle L, Ax \rangle$, for every $ax \in Ax$ we can define a rule $R(ax)$ as a 0-premise and single conclusion rule, whose signed formula is $+\,ax$. Then, the set of *definitional axiom rules* of Ax is defined as $R(Ax) = \bigcup_{ax \in Ax} R(ax)$.

The rule $R(ax)$ basically states that we can introduce the node $+\,ax$ in any place of a tableau tree. We use this fact to prove the following Lemma.

Lemma 3.45. Given a definitional theory $Th := \langle L, Ax \rangle$ and a set of expansion rules ER, $R \cup Ax \vdash_{ER} S$ iff $R \vdash_{ER \cup R(Ax)} S$ for any $R, S \subseteq L$.

Proof. Let ER be a set of expansion rules, $Th := \langle L, Ax \rangle$ be a definitional theory and $R, S \subseteq L$ such that $R = \{r_1, \ldots, r_n\}$, $S = \{s_1, \ldots, s_m\}$ and $Ax = \{ax_1, \ldots, ax_o\}$. In the proof, we use ER_1 to refer to the set $ER \cup R(Ax)$.

$\Rightarrow$: Let P be an ER-proof of S from $R \cup Ax$. The initial segment s_1 of P is

$$\begin{array}{r} (1)\ +\,r_1 \\ \vdots \\ (n)\ +\,r_n \\ (n+1)\ -\,s_1 \\ \vdots \\ (n+m)\ -\,s_m \\ (n+m+1)\ +\,ax_1 \\ \vdots \\ (n+m+o)\ +\,ax_o \end{array}$$

From this, we show how we can construct an ER_1-proof of S from R. We start with an initial segment s_2 that has the same first $n + m + o$ nodes of s_1. Since the rules of $R(Ax)$ are 0-premise, we can expand s_2 by applying all of its rules in such a way that the result of the expansion is a segment s_3 that is equal to s_1. Now, we replicate the same expansions of P in order to obtain the desired proof.

$\Leftarrow$: Let P be an ER_1-proof of S from R. We show how we can transform P so it becomes an ER-proof of S from $R \cup Ax$.

Given an arbitrary $ax \in Ax$, there are two cases to consider: either $R(ax)$ was used somewhere in P or $R(ax)$ was not used in P.

- Case 1: Assume the rule $R(ax)$ was used somewhere in P. Since $R(ax)$ is a 0-premise rule, it can be moved upwards and P remains an ER_1-proof of S from R. For every application of $R(ax)$ in P, we move its conclusion, which is $+ax$, until it is the node right below the node n in the tree. In case of multiple applications of $R(ax)$, say q, we may consider the direct descendants of $n+o$, where $1 < o \leq q$, to be the direct descendants of $n+1$. Since the node $n+1$ is the same as the node $n+o$, the expansions that added the former direct descendants of $n+o$ are still correct. In the end, we remove all nodes from $n+2$ to $n+q$.

- Case 2: If $R(ax)$ was not used in P, we simply add the conclusion of $R(ax)$ as the node right below the node n.

Repeating this procedure to all definitional axioms of Ax, we have an ER_1-proof Q for S that has as its first nodes $\{+r_1, \ldots, +r_n, -s_1, \ldots, -s_m, +ax_1, \ldots, +ax_o\}$ in which the only rules applied after these first nodes are from ER. By definition, Q is an ER-proof of S from $R \cup Ax$. □

3.2.2 Prenex normal form

Recall that our definition of definitional axiom only states that they are closed formulas. As it is going to be detailed in Theorem 3.74, the procedure of extraction lays on the elimination of logical symbols of definitional axioms. Moving all quantifiers of a definitional axiom to its beginning enables us to start by eliminating all quantifiers and not having to worry about them along the procedure. For this, we present the *prenex normal form*.

Definition 3.46 (Prenex normal form)**.** A first-order formula φ is said to be in *prenex normal form* (or PNF for short) if φ has the form $SQ\psi$, where SQ is a finite, and possibly empty, sequence of quantifications and ψ is a formula containing no quantifiers.

Definition 3.47 (Variables named apart)**.** A closed first-order formula φ has its *variables named apart* if every variable is bound by a unique quantifier.

Considering a language containing one unary predicate P. The formula $(\forall x)P(x) \vee (\exists x)\neg P(x)$ of this language does not have its variables named apart since the x is bound by the universal and the existential. But the formula $(\forall x)P(x) \vee (\exists y)\neg P(y)$ has its variables named apart.

Fact 3.48 (See page 209 of [11])**.** Let φ and $(Qx)\psi$ be formulas such that $Q \in \{\exists, \forall\}$, $(Qx)\psi$ is a direct subformula of φ and the variables of φ are named apart. We can convert φ into a formula φ^* by using the rules described in the Table 5 such that $\models \varphi \leftrightarrow \varphi^*$.

φ	φ^*
$\neg(\forall x)\psi$	$(\exists x)\neg\psi$
$\neg(\exists x)\psi$	$(\forall x)\neg\psi$
$(\forall x)\psi \wedge \phi$	$(\forall x)(\psi \wedge \phi)$
$\phi \wedge (\forall x)\psi$	$(\forall x)(\phi \wedge \psi)$
$(\exists x)\psi \wedge \phi$	$(\exists x)(\psi \wedge \phi)$
$\phi \wedge (\exists x)\psi$	$(\exists x)(\phi \wedge \psi)$
$(\forall x)\psi \vee \phi$	$(\forall x)(\psi \vee \phi)$
$\phi \vee (\forall x)\psi$	$(\forall x)(\phi \vee \psi)$
$(\exists x)\psi \vee \phi$	$(\exists x)(\psi \vee \phi)$
$\phi \vee (\exists x)\psi$	$(\exists x)(\phi \vee \psi)$
$(\forall x)\psi \rightarrow \phi$	$(\exists x)(\psi \rightarrow \phi)$
$\phi \rightarrow (\forall x)\psi$	$(\forall x)(\phi \rightarrow \psi)$
$(\exists x)\psi \rightarrow \phi$	$(\forall x)(\psi \rightarrow \phi)$
$\phi \rightarrow (\exists x)\psi$	$(\exists x)(\phi \rightarrow \psi)$

Table 5: Ways to convert a formula with a proper quantified subformula into an equivalent quantified formula

As a consequence of this fact and the Facts 3.42 and 3.43, we have the following corollary:

Corollary 3.49. We can convert every formula φ into a formula φ^{PNF} in PNF such that $\models \varphi \leftrightarrow \varphi^{PNF}$.

The conversion of a formula φ into PNF is non-deterministic, as φ may have more than one proper quantified subformula. When there are nested quantifications, we work from the outside in. Otherwise, we give preference to choosing either existentially quantified subformulas which occur positively or universally quantified subformulas which occur negatively in $+\varphi$.

Example 3.50. In this example, we detail every step of the conversion of the definitional axiom $[ax \subseteq]$ from Example 3.16 into prenex normal form.

The formula of the definitional axiom $[ax \subseteq]$ is $(\forall x, y)(x \subseteq y \leftrightarrow (\forall z)(z \in x \rightarrow z \in y))$.

Step 1: None of the conversions in Table 5 refer to formulas with bi-implications. This is because the treatment for the quantifier is different depending on the side of the implication it is. So, the first thing to do is to rewrite the bi-implication as the formula

it abbreviates, a conjunction of implications: $\varphi_1 := (\forall x, y)(((\forall z)(z \in x \to z \in y) \to x \subseteq y) \wedge (x \subseteq y \to (\forall z)(z \in x \to z \in y)))$.

Step 2: The resulting formula of Step 1 does not have its variables named apart, since z is bound by two different quantifications. So, what we do now is to pick one of the quantified subformulas of φ_1, say $(\forall z)(z \in x \to z \in y)$, and convert it into $(\forall w)(w \in x \to w \in y)$. The result is $\varphi_2 := (\forall x, y)(((\forall w)(w \in x \to w \in y) \to x \subseteq y) \wedge (x \subseteq y \to (\forall z)(z \in x \to z \in y)))$. The equivalence between φ_1 and φ_2 holds because of Lemma 3.34 and Facts 3.42 and 3.43.

Step 3: Now, we have two unnested quantifications. Then, the one we choose is $(\forall w)(w \in x \to w \in y)$ as it occurs negatively in $+\varphi_2$. By using the conversion in the 11th line and, after, the conversion in the 5th line of the Table 5, the result is $(\forall x, y)(\exists w)(((w \in x \to w \in y) \to x \subseteq y) \wedge (x \subseteq y \to (\forall z)(z \in x \to z \in y)))$.

Step 4: To finish, we use the conversion in the 12th line and, later, the conversion in the 4th line of the Table 5 so we have $(\forall x, y)(\exists w)(\forall z)(((w \in x \to w \in y) \to x \subseteq y) \wedge (x \subseteq y \to (z \in x \to z \in y)))$, which is in PNF.

Do note that if we had skipped the procedure of making the variables named apart, done in Step 2, we would have had two nested quantifiers binding the same variable in the end.

Lemma 3.51. Let r_1 and r_2 be two rules in the following format such that φ^{PNF} is the conversion of φ to PNF:

$$\frac{}{+\varphi}\ r_1 \qquad \frac{}{+\varphi^{PNF}}\ r_2$$

Then, $\vdash_{\mathtt{CB} \cup \{r_1\}} = \vdash_{\mathtt{CB} \cup \{r_2\}}$.

Proof. We prove that r_1 and r_2 are CB-interderivable. By Corollary 3.49, we have that $\models \varphi \leftrightarrow \varphi^{PNF}$. Using the completeness of $\vdash_{\mathtt{CB}}$ with respect to $\models$, it follows that $\vdash_{\mathtt{CB}} \varphi \leftrightarrow \varphi^{PNF}$. By definition, there is a CB-proof P for $\{-\varphi \leftrightarrow \varphi^{PNF}\}$. Let P^* be the result of removing the initial nodes of P. The tree below illustrates how r_2 can be derived in $ER \cup \{r_1\}$.

$$
\begin{array}{rl}
(1) & -\,\varphi^{PNF} \\
(2) & +\,\varphi \\
(3) & -\,\varphi \rightarrow \varphi^{PNF} \quad (1), (2) \\
(4) & -\,\varphi \leftrightarrow \varphi^{PNF} \quad (3) \\
 & P^{*}
\end{array}
$$

The other direction is similar. □

Corollary 3.52. Given a definitional theory $Th := \langle L, Ax \rangle$, $R \vdash_{\mathtt{CB} \cup R(Ax)} S$ iff $R \vdash_{\mathtt{CB} \cup R(Ax)^{PNF}} S$ for any $R, S \subseteq L$, where $R(Ax)^{PNF}$ is the result of replacing every rule $r \in R(Ax)$ having a conclusion $+\,\varphi$ by a 0-premise rule whose conclusion is $+\,\varphi^{PNF}$.

Proof. The proof follows immediately from Lemma 3.51 □

3.2.3 Skolemization

Different from all the rules in CB that are not 0-premise, rules for the elimination of δ-nodes (recall Table 4) have a global character since we need to guarantee that the term being instantiated is new in the proof. This character, in turn, may add complications to the automatization of the proof procedure. The conclusions of the rules in $R(Ax)^{PNF}$ may have positive occurrences of existentially quantified subformulas and this allows for the application of elimination δ-node rules. In order to assure that we do not face such an issue, we dedicate this subsection to showing how we can remove positive occurrences of existentially quantified subformulas from the conclusion of the rules of $R(Ax)^{PNF}$ via a process called *skolemization*.

Definition 3.53 (Skolemized signed formulas and rules)**.** Let sf be a signed formula. We say that sf is *skolemized* if it does not contain neither positive occurrences of existentially quantified subformulas nor negative occurrences of universally quantified subformulas. For a 0-premise single conclusion rule r_1, we say that r_1 is skolemized if its conclusion is skolemized.

Fact 3.54. (See page 206 of [11]) Let $\circ\,\varphi$ be a signed formula and $(\exists x)\psi$ be a formula with positive occurrence in $\circ\,\varphi$ such that $Free\_var((\exists x)\psi) = \{y_1, \ldots, y_n\}$. Let s be an n-ary function symbol that does not occur in φ. If φ^{*} is the result of replacing $(\exists x)\psi$ by $\psi[x \mapsto s(y_1, \ldots, y_n)]$ in φ, then $\circ\,\varphi^{*}$ is satisfiable if and only if $\circ\,\varphi$ is satisfiable.

The result of Fact 3.54 also applies to negative occurrences of universally quantified subformulas. However, we only need the result regarding positive occurrences of universally quantified subformulas since the formulas we treat are already in PNF. As an immediate consequence of Fact 3.54, we have the following corollary:

Corollary 3.55. Any signed formula sf can be converted into a skolemized singed formula sf^{skl} such that sf is satisfiable if and only if sf^{skl} is also satisfiable.

In order to guarantee that the skolemization works for any formula of every language L of finite signature Σ, we must extend Σ to an infinite signature as there can be non-skolemized formulas that contain all function symbols of Σ. For example, let be L be a language and $(\exists x)\varphi \in L$ such that φ contains all function symbols of L. Skolemization could not be applied in $+(\exists x)\varphi$ since there is no function symbol of L that does not occur in φ as it is required. Additionally, if we want to skolemize a signed formula sf, the arity of the first function symbols utilized in the procedure depends on the amount of universal quantifications before the first existential quantification in $fmla(sf)$. Thus, the signatures must be extended taking into account the need of symbols of any arity.

Definition 3.56 (Skolemized language)**.** Given a signature Σ, Σ^{skl} is the result of adding to Σ infinitely many skolemization function symbols skl_i^j such that $i, j \in \mathbb{N}$ and skl_i^j is an i-ary function symbol. We refer to the language $L_{\Sigma^{skl}}(X)$ as simply L^{skl}.

The existence of multiple skolemization symbols for a given arity plays a key role in the proof of Lemma 3.62.

Example 3.57. Let φ be the result of the PNF conversion of the definitional axiom $[ax \subseteq]$ provided in Example 3.50. The result of skolemizing $+\varphi$ is $+(\forall x, y, z)(((skl_2^1(x, y) \in x \rightarrow skl_2^1(x, y) \in y) \rightarrow x \subseteq y) \wedge (x \subseteq y \rightarrow (z \in x \rightarrow z \in y)))$.

Definition 3.58 (Skolemized correspondent)**.** We call two signed formulas sf_1 and sf_2 *skolemized correspondents* if sf_2 is the result of converting sf_1 into a skolemized signed formula. Similarly, two 0-premise single conclusion rules r_1 and r_2 are *skolemized correspondent* if their conclusions are skolemized correspondents.

The definition of definitional axioms should not be affected by the incorporation of skolemization symbols in the language. So, for a definitional theory $Th := \langle L^{skl}, Ax\rangle$, we say that the definitional axioms ax in Ax are closed formulas that do not contain skolemization symbols.

Do note that, in Fact 3.54, $\circ\varphi$ and $\circ\varphi^*$ may be satisfied in different models. The language of the model M that satisfies $\circ\varphi$ may not contain the function s, while the language of the model M^* that satisfies $\circ\varphi^*$ must contain s. In this manuscript (as it is usually done), we consider the model M^* to be an extension of M in the sense that the language of M does not contain s (and any other skolemization symbol) and the models M and M^* provide the same interpretation for the symbols of the language of M.

As well as PNF conversion, the skolemization procedure is non-deterministic, since a signed formula might have more than one positive occurrence of an existentially quantified subformula. However, we introduce the skolemization procedure specifically to deal with PNF conversions of definitional axioms. As a consequence, the occurrences of all quantifiers are nested. Then, by working from outside in, the skolemization becomes deterministic.

In fact, all the definitional axioms we work with have at most one occurrence of existentially quantified subformula. So, the determinism of this procedure would not be threatened. We make this remark for the sake of generality of the proofs we are going to present in what follows.

Given a set of definitional axioms Ax, from this point on, we refer to $R(Ax)^{PNF}$ as R_{Ax_PNF}. Moreover, a set $R^{skl}_{Ax\_PNF}$ is the result of replacing every rule $r \in R_{Ax\_PNF}$ by a skolemized correspondent.

In this subsection, we aim to prove that we can substitute the definitional axioms that were converted to PNF by skolemized correspondents without affecting the consequence relation induced by $\mathtt{CB} \cup R_{Ax\_PNF}$ when neither the formulas in the antecedent nor the formulas in the succedent contain skolemization symbols. In other words, we want to prove that, for any definitional theory $Th := \langle L^{skl}, Ax\rangle$, $R \vdash_{\mathtt{CB}\cup R_{Ax\_PNF}} S$ if and only if $R \vdash_{\mathtt{CB}\cup R^{skl}_{Ax\_PNF}} S$ for any $R, S \subseteq L$. We prove the two directions separately.

Lemma 3.59. Let r_1 and r_2 be two rules in the following format, where $n \geq 0$:

$$\frac{}{+\,(\forall x_1, \ldots, x_n)(\exists y)\varphi}\ r_1 \qquad \frac{}{+\,(\forall x_1, \ldots, x_n)\varphi[y \mapsto skl^i_n(x_1, \ldots, x_n)]}\ r_2$$

For every set of rules ER such that $\mathtt{CB} \subseteq ER$, $\vdash_{ER\cup\{r_1\}} \subseteq \vdash_{ER\cup\{r_2\}}$.

Proof. We prove that r_1 is derivable in $ER\cup\{r_2\}$. For the following tableau proof, consider that $\Phi := \varphi[x_1 \mapsto p_1, \ldots, x_n \mapsto p_n]$:

$$
\begin{array}{c}
(1)\ -\ (\forall x_1, \ldots, x_n)(\exists y)\varphi \\
(2)\ \texttt{Fresh}\ \ p_1\ \ (1) \\
(3)\ -\ (\forall x_2, \ldots, x_n)(\exists y)\varphi[x_1 \mapsto p_1]\ \ (1) \\
\vdots \\
(2n)\ \texttt{Fresh}\ \ p_n\ \ (2n-1) \\
(2n+1)\ -\ (\exists y)\Phi\ \ (2n-1) \\
(2n+2)\ +\ (\forall x_1, \ldots, x_n)\varphi[y \mapsto skl^i_n(x_1, \ldots, x_n)] \\
\vdots \\
(3n+1)\ +\ \Phi[y \mapsto skl^i_n(x_1, \ldots, x_n)]\ \ (3n) \\
(3n+2)\ -\ \Phi[y \mapsto skl^i_n(x_1, \ldots, x_n)]\ \ (2n+1) \\
\maltese \\
\text{(3n+1) (3n+2)}
\end{array}
$$

□

Lemma 3.60. Given a definitional theory $Th := \langle L^{skl}, Ax \rangle$, $R \vdash_{\texttt{CB} \cup R_{Ax\_PNF}} S$ only if $R \vdash_{\texttt{CB} \cup R^{skl}_{Ax\_PNF}} S$ for any $R, S \subseteq L$.

Proof. Let $r \in R_{Ax\_PNF}$. From the result of Lemma 3.59, we can conclude that r is derivable in $r^{skl} \cup ER$, where $\texttt{CB} \subseteq ER$ and r^{skl} is the skolemized correspondent of r. Thus, R_{Ax_PNF} is derivable in $\texttt{CB} \cup R^{skl}_{Ax\_PNF}$. □

We resort to the semantic notion of consequence presented in Subsection 3.1.3 to prove the other direction.

Lemma 3.61. Given a definitional theory $Th := \langle L^{skl}, Ax \rangle$, $R \cup Ax \models S$ if and only if $R \vdash_{\texttt{CB} \cup R_{Ax\_PNF}} S$ for any $R, S \subseteq L$.

Proof. The proof follows immediatily from Fact 3.42 and Lemmas 3.45 and 3.52. □

Lemma 3.62. Let $Th := \langle L^{skl}, Ax \rangle$ be a definitional theory, ax^{skl} be a skolemized correspondent of a PNF conversion of some $ax \in Ax$ and $S \subseteq L$ such that S does not contain any skolemization symbol of ax^{skl}. Then, $S \cup Ax$ is satisfiable only if $S \cup Ax \cup \{ax^{skl}\}$ is satisfiable.

Proof. Assume that $S \cup Ax$ is satisfiable. Then, there is a model M such that (i) $M \models S \cup Ax$. Now, let ax' be a PNF conversion of some $ax \in Ax$. By Corollary 3.49, we have that (ii) $M \models ax \leftrightarrow ax'$. From (i) and (ii), and using the Definition 3.40, we have that

$M \models ax'$, that is, ax' is satisfiable. Suppose that ax^{skl} is a skolemized correspondent of ax'. Then, the Fact 3.54 and the assumpion that neither S nor Ax contain the skolemization symbols of ax^{skl} allow us to conclude that ax^{skl} is satisfiable in a model M^{skl} that extends M. Thus, $M^{skl} \models S \cup Ax \cup \{ax^{skl}\}$, that is, $S \cup Ax \cup \{ax^{skl}\}$ is satisfiable. □

Assume that we skolemize every conclusion of R_{Ax_PNF} with different skolemization symbols. This enables us to prove the following Lemma by adopting the same reasoning usually applied to prove the soundness of a tableaux with respect to a semantic notion of consequence, as in [11] (Section 6.3) and [24] (Chapter 5).

Lemma 3.63. Given a definitional theory $Th := \langle L^{skl}, Ax\rangle$, $R \vdash_{\mathtt{CB} \cup R^{skl}_{Ax\_PNF}} S$ only if $R \cup Ax \models S$ for any $R, S \subseteq L$.

Finally, we can prove the other direction of our goal.

Theorem 3.64. Given a definitional theory $Th := \langle L^{skl}, Ax\rangle$, $R \vdash_{\mathtt{CB} \cup R_{Ax\_PNF}} S$ if $R \vdash_{\mathtt{CB} \cup R^{skl}_{Ax\_PNF}} S$ for any $R, S \subseteq L$.

Proof. The proof follows directly from Lemmas 3.61 and 3.63. □

3.2.4 Extraction of theory-specific rules

In this subsection, we present the procedure that extracts the theory-specific rules of a definitional theory. These rules are all single conclusion and we explain why in the end of this subsection. However, as it is explained in Chapter 4, we apply some mechanisms to enhance the performance of the procedure of extraction of minimal proofs that allow them to behave as multiple conclusion rules.

We start by stating Lemmas that allow us to substitute rules contaning n logical symbols in the formula of their conclusion to a rule containing $n - 1$ logical symbols.

Lemma 3.65. Let $Th := \langle L, Ax\rangle$ be a definitional theory and let r_1 and r_2 be two rules in the following format, such that t is an arbitrary term of L:

$$\frac{}{+\,(\forall x)\varphi}\ r_1 \qquad \frac{}{+\,\varphi(t)}\ r_2$$

For every set of rules ER such that $\mathtt{CB} \subseteq ER$, $\vdash_{ER \cup \{r_1\}} = \vdash_{ER \cup \{r_2\}}$.

Proof. Let ER be a set of rules such that $\texttt{CB} \subseteq ER$. We verify that r_1 and r_2 are ER-interderivable.

r_2 is derivable in $ER \cup \{r_1\}$:

$$
\begin{array}{r}
(1)\ -\varphi(t) \\
(2)\ +(\forall x)\varphi(x) \\
(3)\ +\varphi(t)\ \ (2) \\
\maltese \\
(1)\ (3)
\end{array}
$$

r_1 is derivable in $ER \cup \{r_2\}$:

$$
\begin{array}{r}
(1)\ -(\forall x)\varphi \\
(2)\ \texttt{Fresh}\ p_1\ \ (1) \\
(3)\ -\varphi(p_1)\ \ (1) \\
(4)\ +\varphi(p_1) \\
\maltese \\
(3)\ (4)
\end{array}
$$

□

Lemma 3.66. Let r_1 and r_2 be two rules in the following format:

$$
\dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \end{array}}{+\neg\varphi}\ r_1 \qquad \dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \end{array}}{-\varphi}\ r_2
$$

For every set of rules ER such that $\texttt{CB} \subseteq ER$, $\vdash_{ER \cup \{r_1\}} = \vdash_{ER \cup \{r_2\}}$.

Proof. Let ER be a set of rules such that $\texttt{CB} \subseteq ER$. We verify that r_1 and r_2 are ER-interderivable.

r_2 is derivable in $ER \cup \{r_1\}$:

$$
\begin{array}{r}
(1)\ prem_1 \\
\ldots \\
(n)\ prem_n \\
(n+1)\ +\varphi \\
(n+2)\ +\neg\varphi\ \ (1)\ldots(n) \\
(n+3)\ -\varphi\ \ (n+2) \\
\maltese \\
(\text{n+1})\ (\text{n+3})
\end{array}
$$

r_1 is derivable in $ER \cup \{r_2\}$: The proof is similar, but instead of $+\neg\mathbf{E}$, we use $-\neg\mathbf{I}$ to close the branch. □

Lemma 3.67. Let r_1 and r_2 be two rules in the following format:

$$\dfrac{\begin{array}{c}prem_1\\ \vdots \\ prem_n\end{array}}{-\neg\varphi}\ r_1 \qquad \dfrac{\begin{array}{c}prem_1\\ \vdots \\ prem_n\end{array}}{+\varphi}\ r_2$$

For every set of rules ER such that $\mathtt{CB} \subseteq ER$, $\vdash_{ER\cup\{r_1\}} = \vdash_{ER\cup\{r_2\}}$.

Proof. Similar to 3.66. □

Recall the uniform notation presented in Tables 1 to 3. It is used in the Lemmas to reduce the amount of Lemmas to be stated.

Lemma 3.68. Let r, r_1 and r_2 be three rules in the following format:

$$\dfrac{\begin{array}{c}prem_1\\ \vdots \\ prem_n\end{array}}{\alpha}\ r \qquad \dfrac{\begin{array}{c}prem_1\\ \vdots \\ prem_n\end{array}}{\alpha_1}\ r_1 \qquad \dfrac{\begin{array}{c}prem_1\\ \vdots \\ prem_n\end{array}}{\alpha_2}\ r_2$$

For every set of rules ER such that $\mathtt{CB} \subseteq ER$, $\vdash_{ER\cup\{r\}} = \vdash_{ER\cup\{r_1,r_2\}}$.

Proof. Let ER be a set of rules such that $\mathtt{CB} \subseteq ER$. We verify that r and $\{r_1, r_2\}$ are ER-interderivable. We consider the case in which $\alpha := +\varphi_1 \wedge \varphi_2$ and the other cases are similar.

r_1 is derivable in $ER \cup \{r\}$:

$$\begin{array}{c}(1)\ prem_1\\ \dots\\ (n)\ prem_n\\ (n+1)\ -\varphi_1\\ (n+2)\ +\varphi_1 \wedge \varphi_2\ \ (1)\dots(n)\\ (n+3)\ +\varphi_1\ \ (n+2)\\ \lightning\\ \text{(n+1) (n+3)}\end{array}$$

r_2 is derivable in $ER \cup \{r\}$: Similar to the derivability of r_1 in $ER \cup \{r\}$.

r_1 is derivable in $ER \cup \{r_1, r_2\}$:

$$
\begin{array}{c}
(1)\ prem_1 \\
\dots \\
(n)\ prem_n \\
(n+1)\ -\ \varphi_1 \wedge \varphi_2 \\
(n+2)\ +\ \varphi_1 \ \ (1)\dots(n) \\
(n+3)\ +\ \varphi_2 \ \ (1)\dots(n) \\
(n+4)\ +\ \varphi_1 \wedge \varphi_2 \ \ (n+2),(n+3) \\
\divideontimes \\
\text{(n+1) (n+4)}
\end{array}
$$

□

In Lemma 3.68, we could treat r_1 and r_2 as a single multiple conclusion rule r_α, whose premises would be $prem_1, \dots, prem_n$ and the conclusions would be α_1 and α_2. The rules r and r_α would be ER-interderivable and, as a consequence, we would have multiple conclusion theory-specific rules. In the end of this Subsection, we detail what would be the problem in considering an interderivability with such r_α.

Still regarding Lemma 3.68, in the remainder of the document, we say that r_2 is the *first ER-interderivable rule* to r_1 and r_3 is the *second ER-interderivable rule* to r_1. We use the same terminology to the ER-interderivabilities of Lemmas 3.69 to 3.73.

Lemma 3.69. Let r, r_1 and r_2 be three rules in the following format:

$$
\frac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \end{array}}{\beta}\ r
\qquad
\frac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \end{array}}{\begin{array}{c}\beta_1^c \\ \hline \beta_2\end{array}}\ r_1
\qquad
\frac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \end{array}}{\begin{array}{c}\beta_2^c \\ \hline \beta_1\end{array}}\ r_2
$$

For every set of rules ER such that $\mathtt{CB} \subseteq ER$, $\vdash_{ER\cup\{r\}} = \vdash_{ER\cup\{r_1,r_2\}}$.

Proof. Let ER be a set of rules such that $\mathtt{CB} \subseteq ER$. We verify that r and $\{r_1, r_2\}$ are ER-interderivable. We consider the case in which $\beta := +\,\varphi_1 \vee \varphi_2$ and the other cases are similar.

r_1 is derivable in $ER \cup \{r\}$:

$$
\begin{array}{rl}
(1) & prem_1 \\
& \dots \\
(n) & prem_n \\
(n+1) & -\varphi_1 \\
(n+2) & -\varphi_2 \\
(n+3) & +\varphi_1 \vee \varphi_2 \;\; (1)\dots(n) \\
(n+4) & -\varphi_1 \vee \varphi_2 \;\; (n+1)(n+2) \\
& \divideontimes \\
& \text{(n+3) (n+4)}
\end{array}
$$

r_2 is derivable in $ER \cup \{r\}$: Similar to the derivability of r_1 in $ER \cup \{r\}$.

r is derivable in $ER \cup \{r_1, r_2\}$:

$$
\begin{array}{rl}
(1) & prem_1 \\
& \dots \\
(n) & prem_n \\
(n+1) & -\varphi_1 \vee \varphi_2 \\
(n+2) & -\varphi_1 \;\; (1)\dots(n) \\
(n+3) & -\varphi_2 \;\; (1)\dots(n) \\
(n+4) & +\varphi_2 \;\; (1)\dots(n),(n+2) \\
& \divideontimes \\
& \text{(n+3) (n+4)}
\end{array}
$$

□

In Lemma 3.69, do note that only r_1 is necessary to prove that r is derivable in $ER \cup \{r_1, r_2\}$. A consequence of this Lemma is that we add redundant rules in the final theory-specific rules. In Subsection 4.2.1, we show how these rules are removed based on predefined criteria.

The next Lemmas, from 3.70 to 3.73, are analogous to the last four, from 3.66 to 3.69. The difference is that they show how to substitute nodes in the premises of rules. Their proofs are also very similar: we prove the derivability of rules by using their interderivable rules and one linear rule of `CB`. The only difference is the Lemma 3.73, which we work in detail.

Lemma 3.70. Let r_1 and r_2 be two rules in the following format:

$$\dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \\ +\neg\varphi \end{array}}{conc}\ r_1 \qquad \dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \\ -\varphi \end{array}}{conc}\ r_2$$

For every set of rules ER such that $\mathtt{CB} \subseteq ER$, $\vdash_{ER\cup\{r_1\}} = \vdash_{ER\cup\{r_2\}}$.

Lemma 3.71. Let r_1 and r_2 be two rules in the following format:

$$\dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \\ -\neg\varphi \end{array}}{conc}\ r_1 \qquad \dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \\ +\varphi \end{array}}{conc}\ r_2$$

For every set of rules ER such that $\mathtt{CB} \subseteq ER$, $\vdash_{ER\cup\{r_1\}} = \vdash_{ER\cup\{r_2\}}$.

Lemma 3.72. Let r_1 and r_2 be three rules in the following format:

$$\dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \\ \alpha \end{array}}{conc}\ r_1 \qquad \dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \\ \alpha_1 \\ \alpha_2 \end{array}}{conc}\ r_2$$

For every set of rules ER such that $\mathtt{CB} \subseteq ER$, $\vdash_{ER\cup\{r_1\}} = \vdash_{ER\cup\{r_2\}}$.

Lemma 3.73. Let r, r_1 and r_2 be three rules in the following format:

$$\dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \\ \beta \end{array}}{conc}\ r \qquad \dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \\ \beta_1 \end{array}}{conc}\ r_1 \qquad \dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \\ \beta_2 \end{array}}{conc}\ r_2$$

For every set of rules ER such that $\mathtt{CB} \subseteq ER$, $\vdash_{ER\cup\{r\}} = \vdash_{ER\cup\{r_1,r_2\}}$.

Proof. Let ER be a set of rules such that $\mathtt{CB} \subseteq ER$. We verify that r and $\{r_1, r_2\}$ are ER-interderivable. We consider the case in which $\beta := +\varphi_1 \vee \varphi_2$ and the other cases are similar.

The proofs the r_1 and r_2 are derivable in $ER\cup\{r\}$ are similar to the proofs presented in Lemmas from 3.66 to 3.69. So, we verify that r is derivable in $ER \cup \{r_1\}$:

$$
\begin{array}{c}
(1)\ prem_1 \\
\dots \\
(n)\ prem_n \\
(n+1)\ +\varphi_1 \vee \varphi_2 \\
(n+2)\ conc^c \\
\hline
\mathcal{B}_1 \quad \mathcal{B}_2
\end{array}
$$

$\mathcal{B}_1$:

$$
\begin{array}{l}
(n+3)\ +\varphi_1 \\
(n+4)\ conc\ (1)\dots(n),(n+3) \\
\hline
\divideontimes \\
\text{(n+2) (n+4)}
\end{array}
$$

$\mathcal{B}_2$:

$$
\begin{array}{l}
(n+5)\ -\varphi_1 \\
(n+6)\ +\varphi_2\ \ (n+1),(n+5) \\
(n+7)\ conc\ (1)\dots(n),(n+6) \\
\hline
\divideontimes \\
\text{(n+2) (n+7)}
\end{array}
$$

□

Theorem 3.74. For every definitional theory $Th := \langle L^{skl}, Ax\rangle$, there is a set of expansion rules Ts_{Th} whose rules do not contain logic symbols neither in their premises nor in their conclusions such that $R \cup Ax \vdash_{\texttt{CB}} S$ if and only if $R \vdash_{\texttt{CB}\cup Ts_{Th}} S$ for any $R, S \subseteq tsl(L)$.

Proof. Let $Th := \langle L^{skl}, Ax\rangle$ be a definitional theory. By Lemma 3.45, we know that, when the set of definitional rules $R(Ax)$ is added to CB, the conjectures that are CB-provable using definitional axioms of Ax in their antecedents remain CB-provable without the definitional axioms. Then, we convert the conclusions of the set of definitional rules to PNF and, after this, skolemize them. The result is a set of rules TS^0_{Th} such that, by Lemmas 3.52, 3.60 and 3.64, $R \cup AX \vdash_{\texttt{CB}} S$ iff $R \vdash_{\texttt{CB}\cup TS^0_{Th}} S$ for any $R, S \subseteq tsl(L)$.

We proceed by eliminating the logical symbols from the rules in TS^0_{Th}.

Step 1: Elimination of universal quantifiers

Every formula φ that was skolemized after being converted in PNF has the format $(\forall x_1, \dots, x_n)\varphi(x_1, \dots, x_n)$, where $\varphi(x_1, ..., x_n)$ is not a quantified formula. Then, from every $r \in TS^0_{Th}$, we can use the ER-interderivability presented in Lemma 3.65 n times, where n is the amount of universal quantifications in the beginning of the formula in the conclusion of r, so as to extract a rule r^{prop} whose formula in the conclusion is not a quantified formula. When substituting every rule r in TS^0_{Th} by r^{prop}, we are left with the set of rules we call Ts^1_{Th}.

At Steps 2 and 3, we present two recursive procedures, in which we use some auxiliary functions:

- *conclusion*: Returns the conclusion node of the rule given as input.
- *premises*: Returns a list of the nodes in the premises of the rule given as input.
- is_ts: Checks if the formula of a node given as input does not contain logical symbols, that is, if it is theory-specific for some language.
- are_ts: Checks if the formulas of all nodes in a list given as input do not contain logical symbols, that is, if they are theory-specific for some language.
- is_alpha: Checks if a node is an α-node.
- is_beta: Checks if a node is a β-node.
- $lemma\_x$: Returns the ER-interderivable rule presented in Lemma x to the rule given as input, $x \in \{3.66, 3.67, 3.70, 3.71, 3.72\}$.
- $lemma\_x.1$: Returns the first ER-interderivable rule presented in Lemma x to the rule given as input, where $x \in \{3.68, 3.69, 3.73\}$.
- $lemma\_x.2$: Returns the second ER-interderivable rule presented in Lemma x to the rule given as input, where $x \in \{3.68, 3.69, 3.73\}$.

Step 2: Elimination of propositional connectives in conclusion

For a rule $r \in Ts^1_{Th}$, the goal is to extract a set Ts^2_{Th} of CB-interderivable rules for which the signed formulas in their conclusions contain only theory-specific formulas of L. For this, we use the ER-interderivabilities stated in Lemmas 3.66 to 3.69 in the recursive procedure presented in Algorithm 1.

At every conditional block inside the **else** of line 3, we make a recursive function call passing as argument all rules that are CB-interderivable to the rule that is being processed in that stage of the recursion. So, in the end, it is guaranteed that we have added a set of rules that is CB-interderivable to the rule passed initially as input and for which the conclusions do not have logical symbols.

Do note also that the amount of propositional connectives is always finite in the formula in the conclusion of the rule provided as input. Since the ER-interderivabilities

Algorithm 1 elim_connec_conc($rule$)

1: **if** $is\_ts(conclusion(rule))$ **then**
2: $\quad Ts_{Th_4} \leftarrow Ts_{Th_4} \cup \{rule\}$
3: **else**
4: $\quad$ **if** $conclusion(rule) = +\neg\varphi$ **then**
5: $\quad\quad elim\_connec\_conc(lemma\_3.66(rule))$
6: $\quad$ **end if**
7: $\quad$ **if** $conclusion(rule) = -\neg\varphi$ **then**
8: $\quad\quad elim\_connec\_conc(lemma\_3.67(rule))$
9: $\quad$ **end if**
10: $\quad$ **if** $is\_alpha(conclusion(rule))$ **then**
11: $\quad\quad elim\_connec\_conc(lemma\_3.68.1(rule))$
12: $\quad\quad elim\_connec\_conc(lemma\_3.68.2(rule))$
13: $\quad$ **end if**
14: $\quad$ **if** $is\_beta(conclusion(rule))$ **then**
15: $\quad\quad elim\_connec\_conc(lemma\_3.69.1(rule))$
16: $\quad\quad elim\_connec\_conc(lemma\_3.69.2(rule))$
17: $\quad$ **end if**
18: **end if**

presented in Lemmas 3.66 to 3.69 always reduce the amount of propositional connectives in the nodes of the conclusion of the rule, this procedure always terminates.

If we apply this procedure to every rule in Ts^1_{Th}, we obtain a set of expansion rules Ts^2_{Th} that is CB-interderivable to Ts^1_{Th} and whose formulas in the nodes of the conclusion of their rules do not contain logical symbols.

Step 3: Elimination of propositional connectives in premises

Due to the CB-interderivable of Lemma 3.69, we might add rules in Ts^2_{Th} with premises whose formulas contain propositional connectives. Now, for a rule $r \in Ts^2_{Th}$, the goal is to extract a set Ts_{Th} of CB-interderivable rules for which the signed formulas in their premises contain only theory-specific formulas of L. The difference here is that we may have more than one premise with a formula containing logical symbols. So, we add a loop in the recursive procedure presented in Algorithm 2, in order to process all premises.

The verification that this procedure terminates is similar to the one presented in Step 2. For the correctness, at every conditional block inside the **for** loop of line 4, we make a recursive function call to all rules that are CB-interderivable to the rule that is being processed in that stage of the recursion. So, in the end again, it is guaranteed that we have added a set of rules that is CB-interderivable to the rule passed initially as input. Since we do this to all premises, the formulas of all premise rules in Ts_{Th} are in the theory-specific language of L.

Algorithm 2 elim_connec_prem($rule$)

if $are\_ts(premises(rule))$ **then**
  $Ts_{Th} \leftarrow Ts_{Th} \cup \{rule\}$
else
  for $prem$ **in** $premises(rule)$ **do**
    if $prem = +\neg\varphi$ **then**
      $elim\_connec\_prem(lemma\_3.70(rule))$
    end if
    if $prem = -\neg\varphi$ **then**
      $elim\_connec\_prem(lemma\_3.71(rule))$
    end if
    if $is\_alpha(prem)$ **then**
      $elim\_connec\_prem(lemma\_3.72(rule))$
    end if
    if $is\_beta(prem)$ **then**
      $elim\_connec\_prem(lemma\_3.73(rule))$
      $elim\_connec\_prem(lemma\_3.73(rule))$
    end if
  end for
end if

Finally, after these 3 steps, we have a set of rules Ts_{Th} such that $R \cup Ax \vdash_{\mathtt{CB}} S$ if and only if $R \vdash_{\mathtt{CB} \cup Ts_{Th}} S$ for any $R, S \subseteq tsl(L)$. □

Definition 3.75 (Theory-specific tableaux)**.** A theory-specific tableau is a tableau constructed only by signed formulas that do not contain logical symbols.

Now, we run the procedure of extraction of theory-specific rules for the definitional theories of sets and numbers, respectively, from Examples 3.16 and 3.17.

Example 3.76 (Extraction of theory-specific rules for the theory of sets)**.** In the remainder of this section, we refer to the result of this example as TS_{sets}. First, as the result of preprocessing the definitional axioms, we have the following rules:

Definitional theory preprocess:

$$\frac{}{+\,(\forall x)(\neg x \in \emptyset)} \qquad \frac{}{+\,(\forall x, y)(x \in \complement(y) \leftrightarrow \neg x \in y)} \qquad \frac{}{+\,(\forall x, y, z)(x \in y \cup z \leftrightarrow (x \in y \vee x \in z))}$$

$$\frac{}{+\,(\forall x, y, z)(x \in y \cap z \leftrightarrow (x \in y \wedge x \in z))} \qquad \frac{}{+\,(\forall x, y, z)(x \in y \setminus z \leftrightarrow (x \in y \wedge \neg x \in z))}$$

$$\frac{}{+\,(\forall x, y, z)(x \in y \times z \leftrightarrow (fst(x) \in y \wedge snd(x) \in z))}$$

$$\frac{}{+\,(\forall x, y, z)(x \in y \triangle z \leftrightarrow ((\neg x \in y \rightarrow x \in z) \wedge (x \in y \rightarrow \neg x \in z)))}$$

$$\frac{}{+\,(\forall x,y,z)(((skl_2^1(x,y)\in x\to skl_2^1(x,y)\in y)\to x\subseteq y)\wedge(x\subseteq y\to(z\in x\to z\in y)))}$$

$$\frac{}{+\,(\forall x,y,z)((\neg(skl_2^2(x,y)\in x\wedge skl_2^2(x,y)\in y)\to x\between y)\wedge(x\between y\to\neg(z\in x\wedge z\in y)))}$$

Then, we describe the result of each of the three steps of the extraction.

Step 1:

$$\frac{}{+\,\neg x\in\emptyset}\qquad\frac{}{+\,x\in\complement(y)\leftrightarrow\neg x\in y}\qquad\frac{}{+\,x\in y\cup z\leftrightarrow(x\in y\vee x\in z)}$$

$$\frac{}{+\,x\in y\cap z\leftrightarrow(x\in y\wedge x\in z)}\qquad\frac{}{+\,x\in y\setminus z\leftrightarrow(x\in y\wedge\neg x\in z)}$$

$$\frac{}{+\,x\in y\times z\leftrightarrow(x\in fst(y)\wedge x\in snd(z))}\qquad\frac{}{+\,x\in y\triangle z\leftrightarrow((x\in y\vee x\in z)\wedge\neg(x\in y\wedge x\in z))}$$

$$\frac{}{+\,((skl_2^1(x,y)\in x\to skl_2^1(x,y)\in y)\to x\subseteq y)\wedge(x\subseteq y\to(z\in x\to z\in y))}$$

$$\frac{}{+\,(\neg(skl_2^2(x,y)\in x\wedge skl_2^2(x,y)\in y)\to x\between y)\wedge(x\between y\to\neg(z\in x\wedge z\in y))}$$

Step 2:

$$\frac{}{-\,x\in\emptyset}$$

$$\frac{+\,x\in\complement(y)}{-\,x\in y}\qquad\frac{-\,x\in\complement(y)}{+\,x\in y}\qquad\frac{+\,\neg x\in y}{+\,x\in\complement(y)}\qquad\frac{-\,\neg x\in y}{-\,x\in\complement(y)}$$

$$\frac{-\,x\in y\quad+\,x\in y\cup z}{+\,x\in z}\qquad\frac{-\,x\in z\quad+\,x\in y\cup z}{+\,x\in y}\qquad\frac{-\,x\in y\cup z}{-\,x\in y}\qquad\frac{-\,x\in y\cup z}{-\,x\in z}$$

$$\frac{+\,x\in y\vee x\in z}{+\,x\in y\cup z}\qquad\frac{-\,x\in y\vee x\in z}{-\,x\in y\cup z}$$

$$\frac{+\,x\in y\cap z}{+\,x\in y}\qquad\frac{+\,x\in y\cap z}{+\,x\in z}\qquad\frac{+\,x\in y\quad-\,x\in y\cap z}{-\,x\in z}\qquad\frac{+\,x\in z\quad-\,x\in y\cap z}{-\,x\in y}$$

$$\frac{+\,x\in y\wedge x\in z}{+\,x\in y\cap z}\qquad\frac{-\,x\in y\wedge x\in z}{-\,x\in y\cap z}$$

$$\frac{+\,x\in y\setminus z}{+\,x\in y}\qquad\frac{+\,x\in y\setminus z}{-\,x\in z}\qquad\frac{+\,x\in y\quad-\,x\in y\setminus z}{+\,x\in z}\qquad\frac{-\,x\in z\quad-\,x\in y\setminus z}{-\,x\in y}$$

$$\frac{+\,x\in y\wedge\neg x\in z}{+\,x\in y\setminus z}\qquad\frac{-\,x\in y\wedge\neg x\in z}{-\,x\in y\setminus z}$$

$$\frac{+x \in y \times z}{+x \in fst(y)} \qquad \frac{+x \in y \times z}{+x \in snd(z)} \qquad \frac{+x \in fst(y) \quad -x \in y \times z}{-x \in snd(z)} \qquad \frac{+x \in snd(z) \quad -x \in y \times z}{-x \in fst(y)}$$

$$\frac{+x \in fst(y) \wedge x \in snd(z)}{+x \in y \times z} \qquad \frac{-x \in fst(y) \wedge x \in snd(z)}{-x \in y \times z}$$

$$\frac{-x \in y \quad +x \in y \triangle z}{+x \in z} \qquad \frac{-x \in z \quad +x \in y \triangle z}{+x \in y} \qquad \frac{+x \in y \quad +x \in y \triangle z}{-x \in z} \qquad \frac{+x \in z \quad +x \in y \triangle z}{-x \in y}$$

$$\frac{+x \in y \vee x \in z \quad -x \in y \triangle z}{+x \in y} \qquad \frac{+x \in y \vee x \in z \quad -x \in y \triangle z}{+x \in z}$$

$$\frac{+\neg(x \in y \wedge x \in z) \quad -x \in y \triangle z}{-x \in y} \qquad \frac{+\neg(x \in y \wedge x \in z) \quad -x \in y \triangle z}{-x \in z}$$

$$\frac{+(x \in y \vee x \in z) \wedge \neg(x \in y \wedge x \in z)}{+x \in y \triangle z} \qquad \frac{-(x \in y \vee x \in z) \wedge \neg(x \in y \wedge x \in z)}{-x \in y \triangle z}$$

$$\frac{+z \in x \quad +x \subseteq y}{+z \in y} \qquad \frac{-z \in y \quad +x \subseteq y}{-z \in x} \qquad \frac{-x \subseteq y}{+skl_2^1(x,y) \in x} \qquad \frac{-x \subseteq y}{-skl_2^1(x,y) \in y}$$

$$\frac{+skl_2^1(x,y) \in x \to skl_2^1(x,y) \in y}{+x \subseteq y} \qquad \frac{-z \in x \to z \in y}{-x \subseteq y}$$

$$\frac{+z \in x \quad +x \between y}{-z \in y} \qquad \frac{+z \in y \quad +x \between y}{-z \in x} \qquad \frac{-x \between y}{+skl_2^2(x,y) \in x} \qquad \frac{-x \between y}{+skl_2^2(x,y) \in y}$$

$$\frac{+\neg(skl_2^2(x,y) \in x \wedge skl_2^2(x,y) \in y)}{+x \between y} \qquad \frac{-\neg(z \in x \wedge z \in y)}{-x \between y}$$

Step 3:

$$\frac{}{-x \in \emptyset}\ -\emptyset$$

$$\frac{+x \in \complement(y)}{-x \in y}\ +\complement\mathbf{E} \qquad \frac{-x \in y}{+x \in \complement(y)}\ -\complement\mathbf{E} \qquad \frac{-x \in \complement(y)}{+x \in y}\ +\complement\mathbf{I} \qquad \frac{+x \in y}{-x \in \complement(y)}\ -\complement\mathbf{I}$$

$$\frac{-x \in y \quad +x \in y \cup z}{+x \in z}\ +\cup\mathbf{E}_1 \qquad \frac{-x \in z \quad +x \in y \cup z}{+x \in y}\ +\cup\mathbf{E}_2 \qquad \frac{+x \in y}{+x \in y \cup z}\ +\cup\mathbf{I}_1 \qquad \frac{+x \in z}{+x \in y \cup z}\ +\cup\mathbf{I}_2$$

$$\frac{-x \in y \cup z}{-x \in y}\ -\cup\mathbf{E}_1 \qquad \frac{-x \in y \cup z}{-x \in z}\ -\cup\mathbf{E}_2 \qquad \frac{-x \in y \quad -x \in z}{-x \in y \cup z}\ -\cup\mathbf{I}$$

$$\frac{+x \in y \cap z}{+x \in y}\ +\cap\mathbf{E}_1 \qquad \frac{+x \in y \cap z}{+x \in z}\ +\cap\mathbf{E}_2 \qquad \frac{+x \in y \quad +x \in z}{+x \in y \cap z}\ +\cap\mathbf{I}$$

$$\frac{+x \in y \quad -x \in y \cap z}{-x \in z}\ -\cap\mathbf{E}_1 \qquad \frac{+x \in z \quad -x \in y \cap z}{-x \in y}\ -\cap\mathbf{E}_2 \qquad \frac{-x \in y}{-x \in y \cap z}\ -\cap\mathbf{I}_1 \qquad \frac{-x \in z}{-x \in y \cap z}\ -\cap\mathbf{I}_2$$

$$\frac{+x \in y \setminus z}{+x \in y}\ +\setminus\mathbf{E}_1 \qquad \frac{+x \in y \setminus z}{-x \in z}\ +\setminus\mathbf{E}_2 \qquad \frac{+x \in y \quad -x \in z}{+x \in y \setminus z}\ +\setminus\mathbf{I}$$

$$\frac{+x \in y \quad -x \in y \setminus z}{+x \in z}\ -\setminus\mathbf{E}_1 \qquad \frac{-x \in z \quad -x \in y \setminus z}{-x \in y}\ -\setminus\mathbf{E}_2 \qquad \frac{-x \in y}{-x \in y \setminus z}\ -\setminus\mathbf{I}_1 \qquad \frac{+x \in z}{-x \in y \setminus z}\ -\setminus\mathbf{I}_2$$

$$\frac{+x \in y \times z}{+x \in fst(y)}\ +\times\mathbf{E}_1 \qquad \frac{+x \in y \times z}{+x \in snd(z)}\ +\times\mathbf{E}_2 \qquad \frac{+x \in fst(y) \quad +x \in snd(z)}{+x \in y \times z}\ +\times\mathbf{I}$$

$$\frac{+x \in fst(y) \quad -x \in y \times z}{-x \in snd(z)}\ -\times\mathbf{E}_1 \qquad \frac{+x \in snd(z) \quad -x \in y \times z}{-x \in fst(y)}\ -\times\mathbf{E}_2 \qquad \frac{-x \in fst(y)}{-x \in y \times z}\ -\times\mathbf{I}_1 \qquad \frac{-x \in snd(z)}{-x \in y \times z}\ -\times\mathbf{I}_2$$

$$\frac{-x \in y \quad +x \in y \triangle z}{+x \in z}\ +\triangle\mathbf{E}_{1a} \qquad \frac{-x \in z \quad +x \in y \triangle z}{+x \in y}\ +\triangle\mathbf{E}_{1b} \qquad \frac{+x \in y \quad +x \in y \triangle z}{-x \in z}\ +\triangle\mathbf{E}_{2a} \qquad \frac{+x \in z \quad +x \in y \triangle z}{-x \in y}\ +\triangle\mathbf{E}_{2b}$$

$$\frac{-x \in y \quad -x \in y \triangle z}{-x \in z}\ -\triangle\mathbf{E}_{1a} \qquad \frac{-x \in z \quad -x \in y \triangle z}{-x \in z}\ -\triangle\mathbf{E}_{1a'} \qquad \frac{-x \in z \quad -x \in y \triangle z}{-x \in y}\ -\triangle\mathbf{E}_{1b} \qquad \frac{-x \in y \quad -x \in y \triangle z}{-x \in y}\ -\triangle\mathbf{E}_{1b'}$$

$$\frac{+x \in y \quad -x \in y \triangle z}{+x \in z}\ -\triangle\mathbf{E}_{2a} \qquad \frac{+x \in z \quad -x \in y \triangle z}{+x \in z}\ -\triangle\mathbf{E}_{2a'} \qquad \frac{+x \in z \quad -x \in y \triangle z}{+x \in y}\ -\triangle\mathbf{E}_{2b} \qquad \frac{+x \in y \quad -x \in y \triangle z}{+x \in y}\ -\triangle\mathbf{E}_{2b'}$$

$$\frac{+x \in y \quad -x \in z}{+x \in y \triangle z}\ +\triangle\mathbf{I}_1 \qquad \frac{+x \in y \quad -x \in y}{+x \in y \triangle z}\ +\triangle\mathbf{I}_{1'} \qquad \frac{+x \in z \quad -x \in y}{+x \in y \triangle z}\ +\triangle\mathbf{I}_2 \qquad \frac{+x \in z \quad -x \in z}{+x \in y \triangle z}\ +\triangle\mathbf{I}_{2'}$$

$$\frac{-x \in y \quad -x \in z}{-x \in y \triangle z}\ -\triangle\mathbf{I}_1 \qquad \frac{+x \in y \quad +x \in z}{-x \in y \triangle z}\ -\triangle\mathbf{I}_2$$

$$\frac{+z \in x \quad +x \subseteq y}{+z \in y}\ +\subseteq\mathbf{E}_1 \qquad \frac{-z \in y \quad +x \subseteq y}{-z \in x}\ +\subseteq\mathbf{E}_2 \qquad \frac{-skl_2^1(x,y) \in x}{+x \subseteq y}\ +\subseteq\mathbf{I}_1 \qquad \frac{+skl_2^1(x,y) \in y}{+x \subseteq y}\ +\subseteq\mathbf{I}_2$$

$$\frac{-x \subseteq y}{+skl_2^1(x,y) \in x}\ -\subseteq\mathbf{E}_1 \qquad \frac{-x \subseteq y}{-skl_2^1(x,y) \in y}\ -\subseteq\mathbf{E}_2 \qquad \frac{+z \in x \quad -z \in y}{-x \subseteq y}\ -\subseteq\mathbf{I}$$

$$\frac{+z \in x \quad +x \between y}{-z \in y}\ +\between\mathbf{E}_1 \qquad \frac{+z \in y \quad +x \between y}{-z \in x}\ +\between\mathbf{E}_2 \qquad \frac{-x \between y}{+skl_2^2(x,y) \in x}\ -\between\mathbf{E}_1 \qquad \frac{-x \between y}{+skl_2^2(x,y) \in y}\ -\between\mathbf{E}_2$$

$$\frac{-\, skl_2^2(x,y) \in x}{+\, x \between y}\, +\between \mathbf{I}_1 \qquad \frac{-\, skl_2^2(x,y) \in y}{+\, x \between y}\, +\between \mathbf{I}_2 \qquad \frac{\begin{array}{c}+\, z \in x \\ +\, z \in y\end{array}}{-\, x \between y}\, -\between \mathbf{I}$$

Do note that the general redundant rules (recall Definition 3.33), as $-\triangle \mathbf{E}_{1a'}$ and $+\triangle \mathbf{I}_{1'}$, are extracted. In Section 4.2, we present a procedure to remove these and other redundant rules.

Example 3.77 (Extraction of theory-specific rules for the theory of numbers)**.** In the remainder of this section, we refer to the result of this example as $TS_{numbers}$. Again, we start by presenting the result of preprocessing the definitional axioms, then, the result of the three steps of the extraction are described:

Definitional theory preprocess:

$$\overline{+\, (\forall x, y, z)(x \approx y + z \to x \approx z + y)} \qquad \overline{+\, (\forall x, y, z)(x + y \approx x + z \to y \approx z)}$$

$$\overline{+\, (\forall x, w, y, z)((x \approx w \wedge y \approx z) \to (x + y \approx w + z))}$$

$$\overline{+\, (\forall x, y, z)(x \approx y \cdot z \to x \approx z \cdot y)} \qquad \overline{+\, (\forall x, y, z)(x \cdot y \approx x \cdot z \to y \approx z)}$$

$$\overline{+\, (\forall x, w, y, z)((x \approx w \wedge y \approx z) \to (x \cdot y \approx w \cdot z))}$$

$$\overline{+\, (\forall x, y, z)((x \leq y \to y \approx x + skl_2^1(x,y)) \wedge (y \approx x + z \to x \leq y))}$$

$$\overline{+\, (\forall x, y, z)((x \mid y \to y \approx x \cdot skl_2^2(x,y)) \wedge (y \approx x \cdot z \to x \mid y))}$$

Step 1:

$$\overline{+\, x \approx y + z \to x \approx z + y} \qquad \overline{+\, x + y \approx x + z \to y \approx z}$$

$$\overline{+\, (x \approx w \wedge y \approx z) \to (x + y \approx w + z)}$$

$$\overline{+\, x \approx y \cdot z \to x \approx z \cdot y} \qquad \overline{+\, x \cdot y \approx x \cdot z \to y \approx z}$$

$$\overline{+\, (x \approx w \wedge y \approx z) \to (x \cdot y \approx w \cdot z)}$$

$$\overline{+\, (x \leq y \to y \approx x + skl_2^1(x,y)) \wedge (y \approx x + z \to x \leq y)}$$

$$\frac{}{+\,(x \mid y \to y \approx x \cdot skl_2^2(x,y)) \wedge (y \approx x \cdot z \to x \mid y)}$$

Step 2:

$$\frac{+\,x \approx y + z}{+\,x \approx z + y} \qquad \frac{-\,x \approx z + y}{-\,x \approx y + z}$$

$$\frac{+\,x + y \approx x + z}{+\,y \approx z} \qquad \frac{-\,y \approx z}{-\,x + y \approx x + z}$$

$$\frac{+\,x \approx w \wedge y \approx z}{+\,x \cdot y \approx w \cdot z} \qquad \frac{+\,x \approx w \quad -\,x \cdot y \approx w \cdot z}{-\,y \approx z} \qquad \frac{+\,y \approx z \quad -\,x \cdot y \approx w \cdot z}{-\,x \approx w}$$

$$\frac{+\,x \approx y \cdot z}{+\,x \approx z \cdot y} \qquad \frac{-\,x \approx z \cdot y}{-\,x \approx y \cdot z}$$

$$\frac{+\,x \cdot y \approx x \cdot z}{+\,y \approx z} \qquad \frac{-\,y \approx z}{-\,x \cdot y \approx x \cdot z}$$

$$\frac{+\,x \approx w \wedge y \approx z}{+\,x \cdot y \approx w \cdot z} \qquad \frac{+\,x \approx w \quad -\,x \cdot y \approx w \cdot z}{-\,y \approx z} \qquad \frac{+\,y \approx z \quad -\,x \cdot y \approx w \cdot z}{-\,x \approx w}$$

$$\frac{+\,x \leq y}{+\,y \approx x + skl_2^1(x,y)} \qquad \frac{-\,y \approx x + skl_2^1(x,y)}{-\,x \leq y}$$

$$\frac{+\,y \approx x + z}{+\,x \leq y} \qquad \frac{-\,x \leq y}{-\,y \approx x + z}$$

$$\frac{+\,x \mid y}{+\,y \approx x \cdot skl_2^2(x,y)} \qquad \frac{-\,y \approx x \cdot skl_2^2(x,y)}{-\,x \mid y}$$

$$\frac{+\,y \approx x \cdot z}{+\,x \mid y} \qquad \frac{-\,x \mid y}{-\,y \approx x \cdot z}$$

Step 3:

$$\frac{+\,x \approx y + z}{+\,x \approx z + y}\ +_{+com} \qquad \frac{-\,x \approx z + y}{-\,x \approx y + z}\ -_{+com}$$

$$\frac{+\,x + y \approx x + z}{+\,y \approx z}\ +_{+inv} \qquad \frac{-\,y \approx z}{-\,x + y \approx x + z}\ -_{+inv}$$

$$\frac{+x \approx w \quad +y \approx z}{+x + y \approx w + z}\ +_{+cong} \qquad \frac{+x \approx w}{-x + y \approx w + z \quad -y \approx z}\ -_{+cong1} \qquad \frac{+y \approx z}{-x + y \approx w + z \quad -x \approx w}\ -_{+cong2}$$

$$\frac{+x \approx y \cdot z}{+x \approx z \cdot y}\ +_{\cdot com} \qquad \frac{-x \approx z \cdot y}{-x \approx y \cdot z}\ -_{\cdot com}$$

$$\frac{+x \cdot y \approx x \cdot z}{+y \approx z}\ +_{\cdot inv} \qquad \frac{-y \approx z}{-x \cdot y \approx x \cdot z}\ -_{\cdot inv}$$

$$\frac{+x \approx w \quad +y \approx z}{+x \cdot y \approx w \cdot z}\ +_{\cdot cong} \qquad \frac{+x \approx w}{-x \cdot y \approx w \cdot z \quad -y \approx z}\ -_{\cdot cong1} \qquad \frac{+y \approx z}{-x \cdot y \approx w \cdot z \quad -x \approx w}\ -_{\cdot cong2}$$

$$\frac{+x \leq y}{+y \approx x + skl_2^1(x, y)}\ +\leq\mathbf{E} \qquad \frac{-x \leq y}{-y \approx x + z}\ -\leq\mathbf{E}$$

$$\frac{+y \approx x + z}{+x \leq y}\ +\leq\mathbf{I} \qquad \frac{-y \approx x + skl_2^1(x, y)}{-x \leq y}\ -\leq\mathbf{I}$$

$$\frac{+x \,|\, y}{+y \approx x \cdot skl_2^2(x, y)}\ +|\,\mathbf{E} \qquad \frac{-x \,|\, y}{-y \approx x \cdot z}\ -|\,\mathbf{E}$$

$$\frac{+y \approx x \cdot z}{+x \,|\, y}\ +|\,\mathbf{I} \qquad \frac{-y \approx x \cdot skl_2^2(x, y)}{-x \,|\, y}\ -|\,\mathbf{I}$$

As well as the rules from CB, presented in Definition 3.21, theory-specific rules are also rule schemas. This is due the fact that Lemma 3.65 works for arbitrary terms of a given language. So a rule such as $-\emptyset$ represents all 0-premise rules having as conclusion $-t \in \emptyset$ in which t is a term of the language of the theory of sets, described in Definition 3.16. Variables are the chosen terms of the theory-specific rules because we can resort to the notion of substitution to talk about the application of these rules in tableaux.

To finish the presentation of extraction of theory-specific rules, we present a motivational example for restricting theory-specific rules to be single conclusion. Let $Th := \langle L, Ax \rangle$ be a definitional theory such that the only symbols of L are the unary predicate symbols P, Q, R and S and the only definitional axiom of Ax is $[ax_{Th}]$ $(\forall x)((P(x) \vee Q(x)) \wedge (R(x) \vee S(x)))$.

Among Lemmas from 3.65 to 3.73, the one that could be replaced so multiple conclusion theory-specific rules are extracted is the Lemma 3.68. We present an alternative version for it and describe what would be the problem in adopting it.

Lemma 3.78. Let r and r_α be two rules in the following format:

$$\frac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \end{array}}{\alpha}\ r \qquad \frac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \end{array}}{\begin{array}{c}\alpha_1 \\ \alpha_2\end{array}}\ r_\alpha$$

For every set of rules ER such that $\mathtt{CB} \subseteq ER$, $\vdash_{ER\cup\{r\}} = \vdash_{ER\cup\{r_\alpha\}}$.

Proof. The proof is almost similar to the proof of Lemma 3.68. So, we ommit the details. □

By using the Lemma 3.78 instead of Lemma 3.68 in Algorithm 1, the extracted rules for ax_{Th} would be:

$$\frac{\begin{array}{c} -P(x) \\ -R(x) \end{array}}{\begin{array}{c} +Q(x) \\ +S(x) \end{array}}\ r_1 \qquad \frac{\begin{array}{c} -P(x) \\ -S(x) \end{array}}{\begin{array}{c} +Q(x) \\ +R(x) \end{array}}\ r_2 \qquad \frac{\begin{array}{c} -Q(x) \\ -R(x) \end{array}}{\begin{array}{c} +P(x) \\ +S(x) \end{array}}\ r_3 \qquad \frac{\begin{array}{c} -Q(x) \\ -S(x) \end{array}}{\begin{array}{c} +P(x) \\ +R(x) \end{array}}\ r_4$$

Nevertheless, the definitional axiom rule for $[ax_{Th}]$ and the set $\{r_1, r_2, r_3, r_4\}$ are not $\mathtt{CB}$-interderivable. Before, providing a counterexample for this, recall that the introduction rules of $\mathtt{CB}$, labeled with **I**, can be removed from it and its deductive strength is yet preserved. That is, if $\mathtt{CB}_{elim}$ is the result of removing all introduction rules of $\mathtt{CB}$, then $\vdash_{\mathtt{CB}_{elim}} = \vdash_{\mathtt{CB}}$.

Now, as a counterexample, take the set $SF := \{-P(p_1), -Q(p_1)\}$. SF is provable in $\mathtt{CB}\cup\{[ax_{Th}]\}$, as it can be seen in the proof below, but not provable in $\mathtt{CB}\cup\{r_1, r_2, r_3, r_4\}$. From the initial tableau for SF, there is no rule of $\mathtt{CB}_{elim} \cup \{r_1, r_2, r_3, r_4\}$ that could be applied to expand it. So, SF is not provable in $\mathtt{CB}_{elim} \cup \{r_1, r_2, r_3, r_4\}$ and, hence, not provable in $\mathtt{CB} \cup \{r_1, r_2, r_3, r_4\}$.

(1) $-P(p_1)$

(2) $-Q(p_1)$

(3) $+(\forall x)((P(x) \vee Q(x)) \wedge (R(x) \vee S(x)))$

(4) $+(P(p_1) \vee Q(p_1)) \wedge (R(p_1) \vee S(p_1))$ (3)

(5) $+P(p_1) \vee Q(p_1)$ (4)

(6) $+Q(p_1)$ (1), (5)

※

(2) (6)

The problem in adopting the interderivability of Lemma 3.78 is that, after using it, we may end-up adding unnecessary premises. To conclude $+Q(p_1)$, for example, only $-P(p_1)$ should suffice. Such issue occurs when the α_1 and α_2 of the α-node are β-nodes.

3.2.5 Cut rule in theory-specific tableaux

When the *cut* rule of CB is applied in a tableau, the cut formula must be a subformula of a formula in a node above that application. In the case of theory-specific tableaux, such restriction does not fit well since the formulas of the nodes in the tableaux do not contain logical symbols. Then, we need to work with an alternative notion of analyticity.

Cut-based systems regulated by different notions of analyticity are not a novelty. In [20], the authors present a cut-based tableau system for a logic of formal inconsistency, the logic mCi [6], in which one of the rules has in the conclusion one formula that is not a subformula of the formula in the premises. In this subsection, we present the cut rule for theory-specific tableaux, *cut*$'$, an alternative version for the *cut* rule of CB in which the analyticity is no longer regulated by the formulas in the tableaux, but rather by the rules we have extracted from the definitional axioms.

For an arbitrary definitional theory $Th := \langle L^{skl}, Ax\rangle$, we refer to $TS_{Th} \cup cut'$ as TS^{cut}_{Th}. We aim that the equivalence between $R, Ax \vdash_{\mathtt{CB}} S$ and $R, Ax \vdash_{TS^{cut}_{Th}} S$, where $R, S \subseteq tsl(L)$, holds. We dot not provide a formal verification for this, but the intuition is that in a proof for $R, Ax \vdash_{\mathtt{CB}} S$ the set Δ of cut formulas of interest are just the subformulas of the definitional axioms of Ax. The set Δ, in turn, is represented exactly by the premises and conclusions of the rules of TS_{Th}.

Definition 3.79 (Main premise)**.** Let r be a rule containing two or more premises. A premise *prem* of r is said to be a *main premise* of r if the set of variables of all premises of r is a subset of the set of variables of *prem*.

A rule may have premises without necessarily containing a main premise, as it is the case of the rules $+\subseteq$ and $+\backslash$ in TS_{sets} of Example 3.16, and the rules $+\cdot_{cong}$ in $TS_{numbers}$ of Example 3.17. It does not happen in our case studies, but it might be the case that all premises of a rule are main premises of it. This fact is important in the proof of Lemma 3.82.

Definition 3.80 (Co-premises)**.** Let sf_1 and sf_2 be two signed formulas and ER be a set of expansion rules. We say that sf_1 and sf_2 are *co-premises* in ER if there is a rule

$r \in ER$ such that r contains two or more premises and sf_1 and sf_2 can be two different premises of r.

If sf_1 is a main premise of r, then sf_2 is said to be a *minor co-premise* of sf_1 and sf_1 a *major co-premise* of sf_2.

Definition 3.81 (Cut rule for theory-specific tableaux)**.** Let TS_{Th} be a set of theory-specific rules. The cut for a TS_{Th}-tableau is defined as follows:

$$\frac{}{+\varphi \mid -\varphi}\ cut'$$

Either $+\varphi$ or $-\varphi$ is a minor co-premise
in TS_{Th} of a node n above

In the TS^{cut}_{sets}-proof of Figure 3, we have an application of cut' to derive the nodes (5) and (8). This application is correct because $+p_1 \in p_2 \cup p_4$ is a minor co-premise of $-p_1 \in (p_2 \cup p_4) \cap (p_3 \cup p_5)$ in TS^{cut}_{sets}.

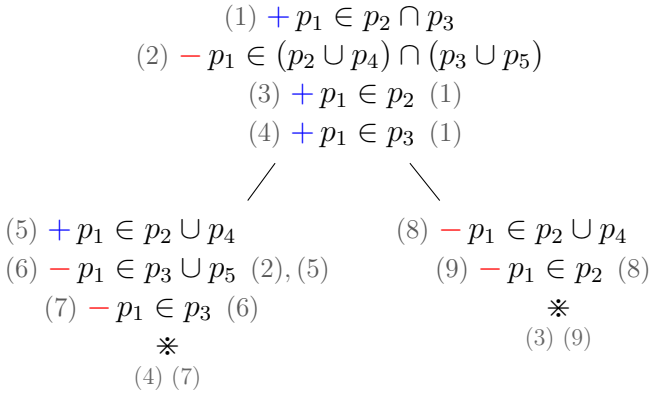


Figure 3: A TS^{cut}_{sets}-proof for $\{+p_1 \in p_2 \cap p_3, -p_1 \in (p_2 \cup p_4) \cap (p_3 \cup p_5)\}$.

Theorem 3.82. Let TS_{Th} be a set of theory specific rules and T be a TS_{Th}-tableau. The set of cut for T is finite.

Proof. Let TS_{Th} be a set of theory specific rules and assume that $\{r_1, \ldots, r_n\}$ is the set of rules of TS_{Th} that have main premises. Now, let $i \in \mathbb{N}$ such that $1 \leq i \leq n$. The amount of premises of r_i is defined as n_i. Then, in the worst case, r_i has n_i minor co-premises, that is, when all the premises of r_i are main premises of it. Thus, the size of the cut for T is $\sum_{i=1}^{n} n_i$ and, hence, finite. □

4 Automated generation of proof exercises with a comparable level of complexity

In this chapter, we present the central method of this study that automatically generates a set of proof exercises with a level of proving complexity that is identical to that of a proof exercise given as input. In Section 4.1, using the background developed in Chapter 3, we provide the definitions which allow us to state when two proof exercises have a comparable level of complexity. The method, described in Section 4.2, is divided into two search procedures: search for minimal proof and search for proof-isomorphic sets of signed formulas. Finally, in Section 4.3, we comment on how our method meets the challenges in the state-of-the-art of AQG tools described in Chapter 2.

4.1 Complexity of proof exercises

The complexity of an exercise should reflect the effort put into solving it. So, to claim that two proof exercises have a comparable level of complexity, we need to provide a way of calculating how complex it is to prove a proof exercise. In this section, we begin by explaining why the theory-specific proofs presented in Chapter 3 can be an adequate alternative to translating informal proofs into formal ones. Then, we proceed by proposing definitions regarding the proving complexity of a theory-specific proof which allows us to state when two provable sets of signed formulas have a comparable level of proving complexity.

4.1.1 Formalizing informal proofs

Novice students in higher Mathematics courses are usually educated into the practice of proving theorems via informal proofs. They learn how to manipulate the givens and

the goals of a theorem via proof strategies, sentences combining natural and mathematical languages. Such sentences are connected in order to construct a continuous text, an informal proof, with no apparent mathematically rigorous structure. So, if we want to state when two proof exercises have a comparable level of complexity, how do we provide a method that precisely calculates the proving complexity of such apparent unstructured objects? In what follows, we justify why the theory-specific proofs, developed in Section 3.2, can be an adequate alternative to translating informal proofs into formal ones. Via the tree structure that formal proofs have, we can develop a method to calculate their proving complexity, as we show in Subsection 4.1.2.

Commonly, students have their first touch with proofs without having any background in formal Logic. As a consequence, the informal proofs they learn to construct frequently keep hidden some of the logical steps behind the proof strategies employed. This is our main argument in favor of adopting theory-specific proofs as a formal counterpart for informal proofs. Take the example of the "element argument", illustrated in Figure 4, regarded as "the fundamental technique in proof technique of set theory" [10]:

Element Argument: The Basic Method for Proving That One Set Is a Subset of Another

Let sets X and Y be given. To prove that $X \subseteq Y$,

1. **suppose** that x is a particular but arbitrarily chosen element of X,
2. **show** that x is an element of Y.

Figure 4: Description of the "Element Argument" from [10].

This proof strategy is used without being necessary to make explicit what goes behind the scenes: the inclusion is defined in terms of a quantified formula in prenex form whose matrix is an implication of assertions involving the membership relation.

Another example of how the logic is put behind in informal proofs is presented in the excerpt for a proof that two consecutive integers have opposite parity in Figure 5. Formally speaking, what makes it possible to conclude that $m+1 = 2k+1$ from $m = 2k$ is the replacement axiom of equality for the sum, but this is not explicitly mentioned in the proof.

The fact that logical steps are "frequently" kept hidden in informal proofs does not mean they are always kept hidden. Now, check the example displayed in Figure 6 of a proof that $A \cap B \subseteq A$ for all sets A and B. Do note that the role of the conjunction in the definition of intersection is played by the particle "and" inside the red rectangle.

Theorem 4.4.2 The Parity Property

Any two consecutive integers have opposite parity.

Proof:

Suppose that two *[particular but arbitrarily chosen]* consecutive integers are given; call them m and $m + 1$. *[We must show that one of m and $m + 1$ is even and that the other is odd.]* By the parity property, either m is even or m is odd. *[We break the proof into two cases depending on whether m is even or odd.]*

Case 1 (m is even): In this case, $m = 2k$ for some integer k, and so $m + 1 = 2k + 1$, which is odd *[by definition of odd]*. Hence in this case, one of m and $m + 1$ is even and the other is odd.

Figure 5: A fragment of the proof that any two consecutive integers have opposite parity from [10].

This is because, in this context, the definition of intersection is the one claiming that "$x \in X \cap Y$ if and only if $x \in X$ and $x \in Y$". Consider, however, an alternative definition of intersection presented by the following list of three unidirectional statements:

1. If $x \in X \cap Y$, then $x \in X$.
2. If $x \in X \cap Y$, then $x \in Y$.
3. If $x \in X$ and $x \in Y$, then $x \in X \cap Y$.

In this example, one could also directly conclude that $x \in A$ from the fact that $x \in A \cap B$ without making explicit the "and" correspondent to the conjunction.

Example 6.2.1 Proof of a Subset Relation

Prove Theorem 6.2.1(1)(a): For all sets A and B, $A \cap B \subseteq A$.

Solution We start by giving a proof of the statement and then explain how you can obtain such a proof yourself.

Proof:

Suppose A and B are any sets and suppose x is any element of $A \cap B$.
Then $x \in A$ and $x \in B$ by definition of intersection. In particular, $x \in A$.
Thus $A \cap B \subseteq A$.

Figure 6: A proof that that $A \cap B \subseteq A$ for all sets A and B adapted from [10].

4.1.2 Comparability of complexity between proof exercises

If we adopt theory-specific proofs as a formal counterpart for informal proofs, what would be the logical description of the statement of a proof exercise? Let E be a proof

exercise. We can say that the theory-specific formulas of the givens of E form a set $\Gamma := \{\gamma_1, \ldots, \gamma_m\}$ and the theory-specific formulas of the goal of E form a set $\Delta := \{\delta_1, \ldots, \delta_n\}$. Then, the consequence statement $\Gamma \vdash \Delta$ and, hence, the set $\{+\gamma_1, \ldots, +\gamma_m, -\delta_1, \ldots, -\delta_n\}$ could be the logical description of E. In this subsection, aiming to be able to state when two proof exercises have a comparable level of complexity, we provide a notion of comparability of proving complexity between provable sets of theory-specific signed formulas.

Let ER be a set of theory-specific expansion rules for a definitional theory $Th := \langle L, Ax\rangle$ and let SF_1 and SF_2 be two sets of theory-specific signed formulas of L that are ER-provable. Intuitively, we want SF_1 and SF_2 to have a comparable level of proving complexity when the minimal effort to prove both sets produces ER-proofs with the same structure. Given an arbitrary set SF of theory-specific signed formulas of L, the strategy we employ is to provide a function *size* from the set of ER-proofs for SF to the set of natural numbers that captures the effort to construct an ER-proof for SF. With *size*, we can compare ER-proofs for SF using the less-than relation on natural numbers. Then, we are allowed to determine a set M of ER-proofs for SF that represents how complex it is to prove SF by defining M as the set of minimal ER-proofs for SF via *size*. Finally, the comparability of proving complexity between SF_1 and SF_2 is going to hold when there is a minimal ER-proof for SF_1 and a minimal ER-proof for SF that are isomorphic.

We start by defining the notion of isomorphism between proofs. To do so, we resort to the tree structure that tableau proofs have so we can rely on the concept of isomorphism between trees.

A precise definition of isomorphism between tableau proofs that expresses their deductive similarity cannot depend only on the tree structure of the tableaux involved. This should be evident by the comparison between the tableau proofs presented in Figure 7, which are constructed using the set of rules extracted in the Example 3.76. In the remainder of this Subsection, we refer to these proofs, respectively, as P_1 and P_2. Despite both tableaux P_1 and P_2 containing 6 nodes, they were constructed in slightly different ways: we use one application of a 2-premise rule in P_1, whereas we use none in P_2; also, the node at the level 3 of P_2 is dispensable and P_1 does not contain dispensable nodes. That is why the formal definition of isomorphism between proofs we provide depends on two different types of isomorphisms to be presented in what follows: between formulas and between what we call *justification trees*.

Definition 4.1 (Syntactic isomorphism)**.** Given a language L, two theory-specific formulas φ_1 and φ_2 of L are *syntactically isomorphic* if the abstract representation of φ_1 is

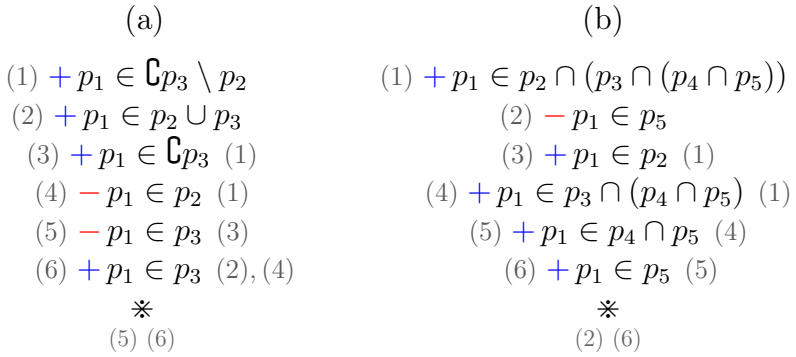


Figure 7: Tableau proofs (a) P_1 for $\{+ p_1 \in \complement p_3 \setminus p_2, + p_1 \in p_2 \cup p_3\}$ and (b) P_2 for $\{+ p_1 \in p_2 \cap (p_3 \cap (p_4 \cap p_5)), - p_1 \in p_5\}$.

isomorphic to the abstract representation of φ_2 and the set of parameters and variables of φ_1 is equal to the set of parameters and variables of φ_2. We extended this definition to signed formulas by saying that two signed formulas sf_1 and sf_2 are syntactically isomorphic if $fmla(sf_1)$ and $fmla(sf_2)$ are syntactically isomorphic.

In Figure 8, we present the abstract representation of three theory-specific formulas from the theory of sets: (a) and (b) are syntactically isomorphic, but they are not syntactically isomorphic to (c), since the parameter p_3 is present in both (a) and (b), but not in (c).

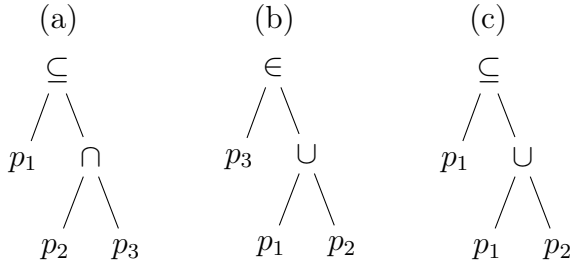


Figure 8: Abstract representation of the formulas (a) $p_1 \subseteq p_2 \cap p_3$, (b) $p_3 \in p_1 \cup p_3$ and (c) $p_1 \subseteq p_1 \cup p_2$.

If we did not impose the equality between the sets of parameters, a theory-specific formula could have an infinite amount of syntactically isomorphic formulas. Hence, this would not allow us to implement a procedure that searches for all syntactically isomorphic formulas to a given formula as we do in Section 4.2.

Definition 4.2 (Justification tree)**.** Let ER be a set of theory-specific rules for some definitional theory. For a branch θ of an ER-proof P, we define the following tree J for θ:

- The root of J is a node whose content is the symbol $\divideontimes$ and the other nodes of J are

nodes of θ.

- The children of the root node are the nodes that made the branch θ close in P. For any other node n in J, their children in J are the nodes that justify the addition of n to θ.

A tree such as J is called a *justification tree* for θ. Given an ER-proof P containing n branches $\theta_1, \ldots, \theta_n$, if $J_1, \ldots, J_n$ are, respectively, the justification trees for $\theta_1, \ldots, \theta_n$, we can also say that $J_1, \ldots, J_n$ are the justification trees for P.

When it is convenient, we represent the nodes of a justification by the indices of the nodes in the tableau tree. In Figure 9, we present justification trees for P_1 and P_2, presented in Figure 7. In the remainder of the section, we refer to the justification trees for P_1 and P_2, respectively, as J_1 and J_2. Do note that the dispensable node of P_2 is not included in J_2. We exclude dispensable nodes from justification trees, so they do not affect the definition of minimal proofs to be provided later on.

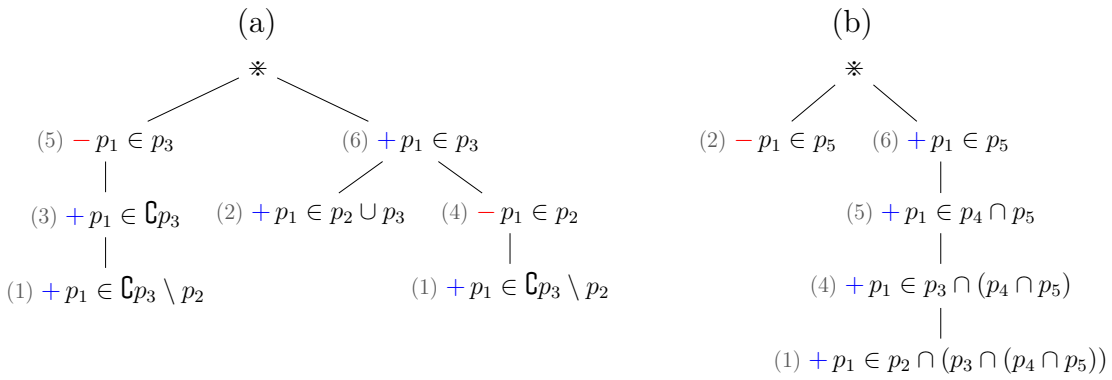


Figure 9: Justification trees (a) J_1 for P_1 and (b) J_2 for P_2.

Definition 4.3 (Deductive isomorphism between justification trees)**.** Let θ_R and θ_L be two branches of theory-specific tableau proofs. Given justification trees R for θ_R and L for θ_L, we say that R and L are *deductively isomorphic* when there is an isomorphism between R and L that preserves the syntactic isomorphism of their formulas.

In Figure 10, there is a proof for $\{+p1 \in \complement(p2 \cap p3), +p1 \in p2 \triangle p3, -p1 \in p2 \cup p3\}$ and a justification tree for it that is deductively isomorphic to J_1. In Figure 11, we illustrate a proof for $\{+p1 \in p4, -p1 \in \complement(p2 \cap (p3 \setminus p4))\}$ and its justification tree. Such justification tree is isomorphic but not deductively isomorphic to J_2, since $-p1 \in \complement(p2 \cap (p3 \setminus p4))$ and $-p1 \in p2 \cap (p2 \cap (p3 \cap p4))$ are not syntactically isomorphic signed formulas.

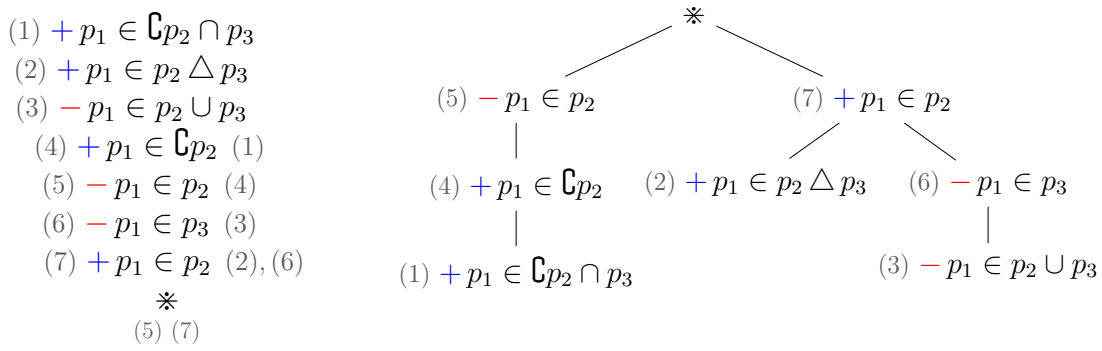


Figure 10: A proof for $\{+p_1 \in \complement p_2 \cap p_3, +p_1 \in p_2 \triangle p_3, -p1 \in p2 \cup p3\}$ and its justification tree.

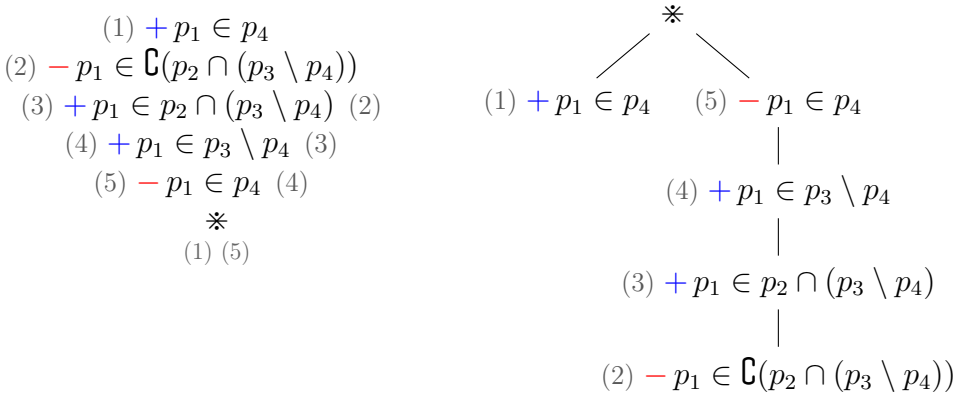


Figure 11: A proof for $\{+p_1 \in p_4, -p_1 \in \complement(p_2 \cap (p_3 \setminus p_4))\}$ and its justification tree.

Definition 4.4 (Deductive isomorphism between proofs)**.** Given a set of expansion rules theory-specific rules ER for some definitional theory and two ER-proofs S and T whose branches are respectively $\theta_{S_1}, \ldots, \theta_{S_n}$ and $\theta_{T_1}, \ldots, \theta_{T_m}$, we say that S and T are *deductively isomorphic* if there is a one-to-one correspondence between the branches of S and T mapping branches of the same size and whose justification trees are deductively isomorphic.

In Figure 12, we present a proof for $\{-p_1 \in p_2, +p_1 \in p_5, +p_1 \in (p_2 \cap p_3) \cup (p_4 \setminus p_5)\}$ that is deductively isomorphic to the proof for $\{-p_1 \in p_4, +p_1 \in p_3, +p_1 \in (p_2 \cup p_3) \setminus (p_4 \setminus p_5)\}$, presented in Figure 13. Do note that, if we replace p_3 in the tableau of Figure 13 by any other term of the language of the theory of sets, such tableau would be still a proof for $\{-p_1 \in p_2, +p_1 \in p_5, +p_1 \in (p_2 \cap p_3) \cup (p_4 \setminus p_5)\}$ having justification trees that are isomorphic to the justification trees Figure 12, but not deductively isomorphic to them. This is the reason why we introduce the notion of deductive isomorphism using syntactic

isomorphism. Otherwise, a proof could have infinite deductively isomorphic proofs and, hence, search for them would be computationally infeasible. For the same reason, we constrain the one-to-one correspondence between branches having the same size. Due to this restriction, P_1 and the proof presented in Figure 10 are not deductively isomorphic even if their justification trees are deductively isomorphic, for instance.

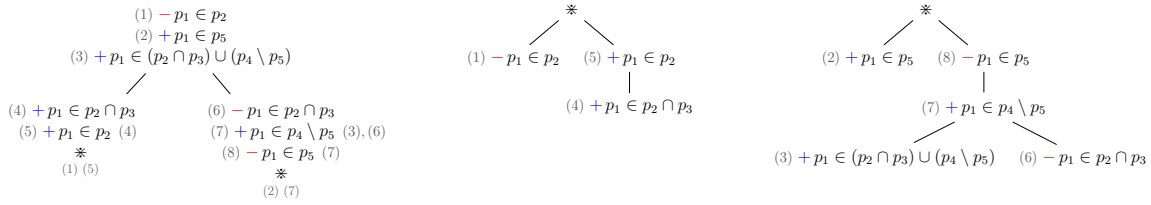


Figure 12: A proof for $\{-p_1 \in p_2, +p_1 \in p_5, +p_1 \in (p_2 \cap p_3) \cup (p_4 \setminus p_5)\}$ and its two justification trees.

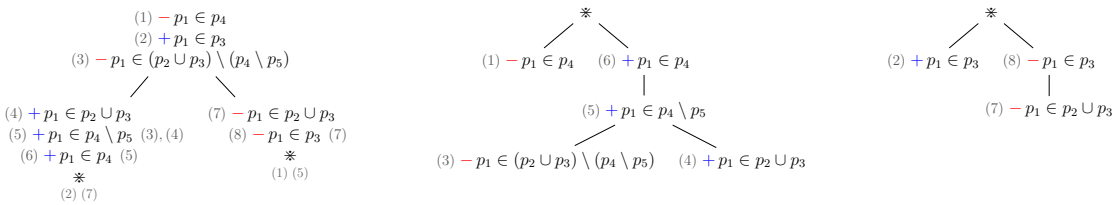


Figure 13: A proof for $\{-p_1 \in p_4, +p_1 \in p_3, +p_1 \in (p_2 \cup p_3) \setminus (p_4 \setminus p_5)\}$ and its two justification trees.

Using the rules of Example 3.77, in Figures 14 and 15, we illustrate two proofs from the theory of numbers that are deductively isomorphic. With the proof in Figure 14, we can answer one of the questions we raise in the end of Subsection 3.1.1 about the way the definitional axiom for the commutativity of + was formulated. If we define it as the formula $(\forall x, y)(x + y \approx y + x)$, it would not be possible to construct such proof for $SF_1 := \{+p_1 \leq p_2, +p_2 \leq p_3, -p_1 \leq p_3\}$. To prove SF_1, it would be necessary to add a constant symbol 0 and add more definitional axioms stating its behaviour as the identity of +.

Definition 4.5 (Size of proofs)**.** Let ER be a set of theory-specific expansion rules for some definitional theory. The *size* of a branch θ of an ER-proof is given by the amount of nodes of its justification tree and the *size* of an ER-proof is given by the sum of the sizes of its branches.

This definition of size of proof is the one we use to capture the effort to construct a proof. The sizes, for instance, of the proofs P_1 and P_2 of Figure 7 and the one illustrated in Figure 13 are, respectively, eight, six and ten.

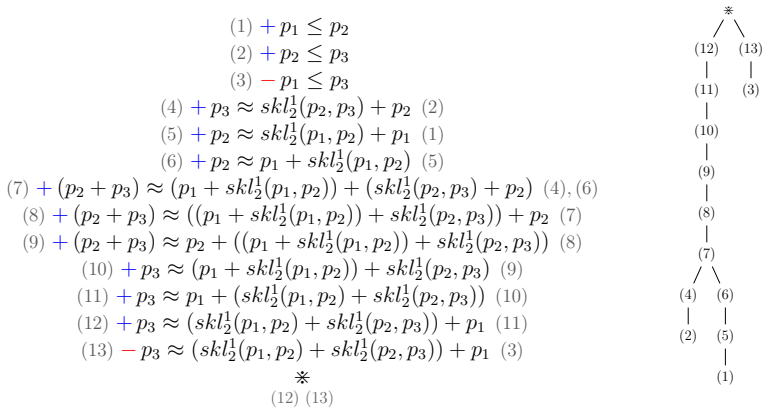


Figure 14: A proof for $\{+\,p_1 \leq p_2, +\,p_2 \leq p_3, -\,p_1 \leq p_3\}$ and its justification tree.

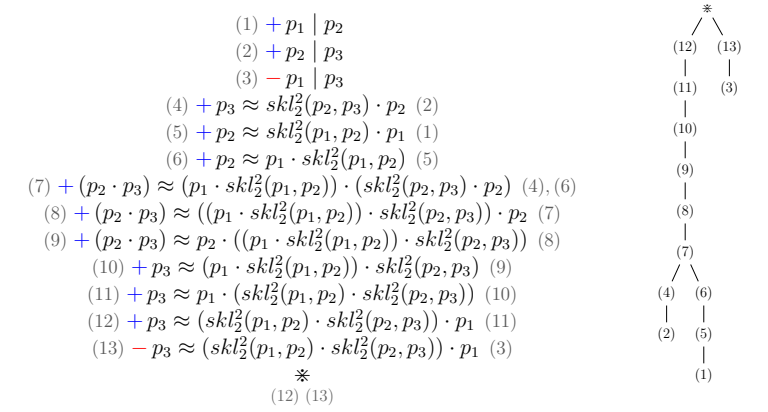


Figure 15: A proof for $\{+\,p_1 \mid p_2, +\,p_2 \mid p_3, -\,p_1 \mid p_3\}$ and its justification tree.

Definition 4.6 (Minimal proofs)**.** Let ER be a set of theory-specific expansion rules for a definitional theory $Th := \langle L, Ax \rangle$. Given a set of theory-specific signed formulas SF of L, an ER-proof P for SF is *minimal* if there is no other ER-proof for SF whose size is smaller than the size of P. The set of minimal proofs for a set of signed formulas is what represents its *proving complexity*.

Now, we possess tools that allow us to formalize the notion of comparability of proving complexity between sets of signed formulas.

Definition 4.7 (Comparable level of proving complexity)**.** Let ER be a set of theory-specific expansion rules for a definitional theory $Th := \langle L, Ax \rangle$. For any sets SF_1 and SF_2 of theory-specific signed formulas of L, we say that SF_1 and SF_2 have a *comparable level of proving complexity* when there are:

(i) An ER-proof P_1 for SF_1 that is minimal and there is an ER-proof for SF_2 that is

also minimal such that P_1 and P_2 are deductively isomorphic and

(ii) A bijective correspondence between SF_1 and SF_2 mapping syntactically isomorphic signed formulas with equal signs.

In this case, SF_1 and SF_2 are also said to be *proof-isomorphic*.

Recall that we use sets of signed formulas to represent a consequence relation that, in turn, formally describes the statement of a proof exercise. The clause (ii) in Definition 4.7 is specified to guarantee that the structure of givens and goals of proof exercises with a comparable level of complexity is preserved. In Figure 16, we present a proof for $\{+\,p_1 \in p_2 \cap p_3, -\,p_1 \in p_2\}$ and a proof for $\{+\,p_1 \in p_2 \setminus p_3, -\,p_1 \in p_2\}$ and their justification trees. These proofs are minimal, since they are constructed with only one application of expansion rule and there are no proofs with zero applications of expansion rules for these sets of signed formulas. Moreover, the justification trees are deductively isomorphic. Thus, $\{+\,p_1 \in p_2 \cap p_3, -\,p_1 \in p_2\}$ and $\{+\,p_1 \in p_2 \setminus p_3, -\,p_1 \in p_2\}$ have a comparable level of proving complexity.

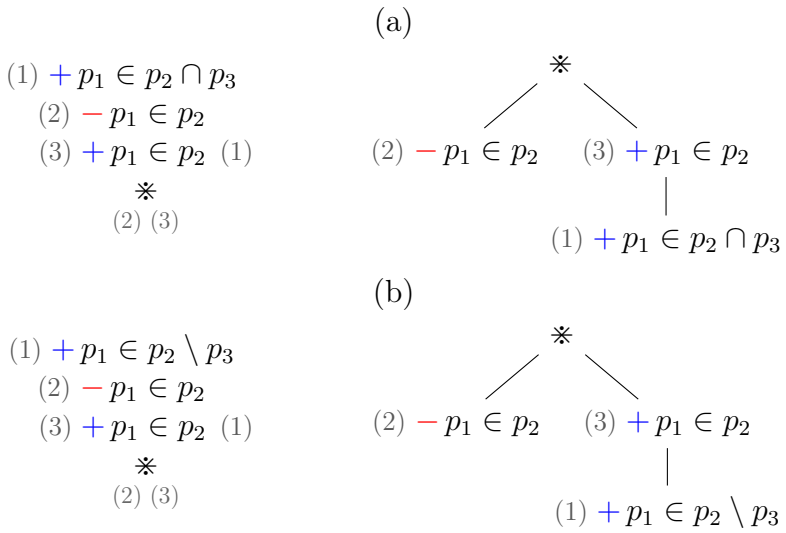


Figure 16: (a) A proof for $\{+\,p_1 \in p_2 \cap p_3, -\,p_1 \in p_2\}$ and its justification tree and (b) a proof for $\{+\,p_1 \in p_2 \setminus p_3, -\,p_1 \in p_2\}$ and its justification tree.

4.2 Description of the procedure

Recall that we formally describe a proof exercise E by a set of signed formulas in which the formulas with positive sign are the given of E and the formulas with negative sign are the alternative goals of E. Given a definitional theory $Th := \langle L, Ax \rangle$, the procedure

we provide for the automated generation of proof exercises with a comparable level of complexity has as inputs a set of signed formulas (the logical description of the exercise) of L and a set of theory-specific rules for Th. We, then, divide the procedure into two main steps:

1. Search for minimal proofs, that represents the proving complexity of the exercise given as input.

2. Search for proof-isomorphic sets of signed formulas. Each of these sets of signed formulas represents the exercises with a comparable level of proving complexity to that of the proof exercise whose logical description was given as input.

In the Figure 17, a diagram of the big picture of this procedure is illustrated.

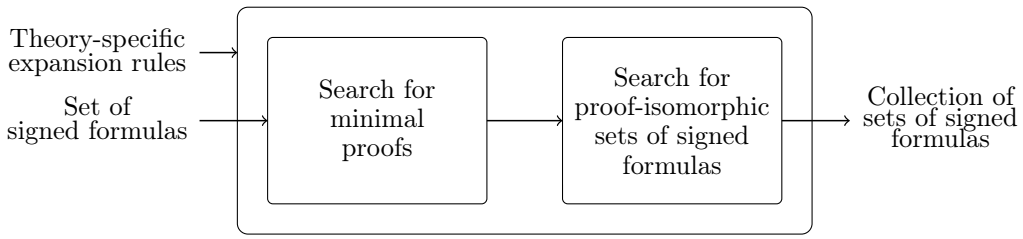


Figure 17: A diagram of the big picture of the generation of signed formulas with a comparable level of complexity.

We start this Section in Subsection 4.2.1 by presenting a preprocessing procedure of the theory-specific rules given as input to the main procedure. In Subsections 4.2.2 and 4.2.3, we present pseudocodes for the implementation of search procedures for minimal proofs and for proof-isomorphic sets of signed formulas. Both search procedures were designed using a brute force-like approach. Then, we also describe some strategies to reduce their search spaces. A prototype implementation of this method was developed and it is described in Appendix A.

4.2.1 Preprocessing of theory-specific rules

As it could be seen through Examples 3.76 and 3.77, the procedure that extracts theory-specific rules, presented in Theorem 3.74, does not protect us from the extraction of redundant rules. Remaining with these redundant rules would slow the performance and add some obstacles to the procedure of extraction of minimal proofs we present in the Subsection 4.2.2. Therefore, we describe a preprocessing procedure of the theory-specific rules provided as input to the generation of proof exercises with a comparable level of

complexity. We start with the description of which rules are to be removed and argue why their removal does not affect the deductive strength of the generation of exercises.

Definition 4.8 (Definitional predicate symbol)**.** Let $Th := \langle L, Ax\rangle$ be a definitional theory and TS_{Th} be a set of theory-specific rules for Th. We say that a predicate symbol p of L is *definitional* in TS_{Th} if p appears in the same premise or conclusion that a skolemization symbol appears in a rule of TS_{Th}.

In the set extracted in the Example 3.76, the predicate symbol $\in$ is definitional since the conclusion of the rule $-\subseteq\mathbf{E}$ is the signed formula $+\, skl^1_2(x, y) \in x$. In the set extracted in the Example 3.77, there is also one predicate definitional symbol, which is $\approx$. This is justified by the rule $-|\,\mathbf{I}$, whose premise is $-\, y \approx x \cdot skl^2_2(x, y)$.

Definition 4.9 (Symbolic introduction and elimination rules)**.** Let $Th := \langle L, Ax\rangle$ be a definitional theory, TS_{Th} be a set of theory-specific rules for Th and s be a predicate or function symbol of L. We say that a rule r in TS_{Th} is a *symbolic introduction rule* for s if s appears in the conclusion of r but not in its premises. If s appears in a premise of r, but not in its conclusion, then r is a *symbolic elimination rule*. The sets of, respectively, all symbolic introduction and elimination rules of a symbol s in TS_{Th} are denoted as $TS_{int}(s, TS_{Th})$ and $TS_{elim}(s, TS_{Th})$.

A set of rules that contains symbolic introduction and elimination rules may fall into cycles. Check the example of the tableau for the set $\{+\, p_1 \mid p_2, +\, p_2 \mid p_3, -\, p_1 \mid p_3\}$ in Figure 18 constructed with the set of rules of Example 3.77. As it can be seen in Figure 15, the conjecture is provable. But, by choosing the wrong rules to prove it, we may fall into cycles in which we eliminate the symbol $\mid$ in a node $+\, x \mid y$ and introduce $\mid$ again in a node $+\, skl^2_2(x, y) \mid y$ with a new term that has not appeared before in the proof.

$$
\begin{array}{c}
(1)\ +\, p_1 \mid p_2 \\
(2)\ +\, p_2 \mid p_3 \\
(3)\ -\, p_1 \mid p_3 \\
(4)\ +\, p_2 \approx p_1 \cdot skl^2_2(p_1, p_2)\ \ (1) \\
(5)\ +\, p_2 \approx skl^2_2(p_1, p_2) \cdot p_1\ \ (4) \\
(6)\ +\, skl^2_2(p_1, p_2) \mid p_2\ \ (5)
\end{array}
$$

Figure 18: A tableau for $\{+\, p_1 \mid p_2, +\, p_2 \mid p_3, -\, p_1 \mid p_3\}$.

If we are able to remove the symbolic introduction rules without losing the deductive strength of the system, then we can reduce our search space for proofs and, hence, the

performance of such search may be improved. Next, we define the symbols for which we remove introduction rules.

Definition 4.10 (Intelim symbols)**.** Let $Th := \langle L, Ax \rangle$ be a definitional theory and TS_{Th} be a set of theory-specific rules for Th. We say that a function or predicate symbol s of L is *intelim* in TS_{Th} if:

(i) s is not a definitional symbol in TS_{Th},

(ii) s is not a skolemization symbol,

(iii) all rules containing s in a premise or conclusion are either introduction or elimination rules for s, and

(iv) if ER is the set of rules such that $ER \cup TS_{int}(s, TS_{Th}) \cup TS_{elim}(s, TS_{Th}) = TS_{Th}$, then $TS_{int}(Th, s)$ and $TS_{elim}(Th, s)$ are interderivable in ER.

The set of intelim symbols of Example 3.76 is $\{\complement, \cup, \cap, \setminus, \times, fst, snd, \triangle, \subseteq, \rangle\!\langle\}$ and the set of intelim symbols of Example 3.77 is $\{\leq, |\}$. The symbolic introduction/elimination rules for function symbols $\times$, fst and snd are not disjoint. For example, the rule $-\times\mathbf{E}_1$ is an elimination rules for $\times$ and fst. In this case, we pick only the symbol of greater arity, which is $\times$.

Some rules generated by the extraction of theory-specific rules allow for the introduction of new terms in a proof, for instance, the 0-premise rules $-\emptyset$ and $-|\mathbf{E}$ of, respectively, Examples 3.76 and 3.77. The challenge with these rules is that we would need to implement a mechanism to guess what are good new terms to be added in a branch of a tableau in order to help it finding a closure. In what follows, we describe this type of rules and how we handle this.

Definition 4.11 (Term introduction rules)**.** Let r be a linear rule. We say that r is a *term introduction rule* if there are variable in the conclusion of r that are not present in any of its premises.

Definition 4.12 (Closure version)**.** Let r and r^* be the following rules:

$$\dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \end{array}}{conc}\ r \qquad \dfrac{\begin{array}{c} prem_1 \\ \vdots \\ prem_n \\ conc^c \end{array}}{\divideontimes}\ r^*$$

In other words, r^* is the result of moving the conjugate of the conclusion of r to the premises. We refer to r^* as the *closure version for* r.

In the preprocessing procedure that we describe in the end of this subsection, the term introduction rules are replaced by their closure versions. The correctness of this replacement is justified by the next lemma.

Lemma 4.13. For every set of rules ER, we have that r and r^* are interderivable in ER.

Proof. The proof follows immediately from the Definition 3.31 of interderivability. □

One last refinement we can do to reduce the amount of theory-specific rules is to join equal premise rules. In the end of Subsection 3.2.4, we explain why theory-specific rules are single conclusion. Basically, the root of the problem was dealing with the extraction of rules for a conclusion α-node when their α_1 and α_2 are β-nodes. However, when the rules are already extracted, there is no reason to keep as single conclusion those rules that have the same premises.

Definition 4.14 (Merging of rules)**.** Let r_1 and r_2 be theory-specific rules with the same premises. The *merging* of r_1 and r_2 is a rule r whose premises are the same premises of r_1 and the conclusions are the conclusions of r_1 and r_2. Such notion is generalized for every amount of theory-specific rules with the same premises.

The rules $-\cup\mathbf{E}_1$ and $-\cup\mathbf{E}_2$ from Example 3.76 can be merged resulting in the following rule:

$$\frac{- \; x \in y \cup z}{\begin{array}{c} - \; x \in z \\ - \; x \in y \end{array}}$$

Lemma 4.15. Let r_1 and r_2 be two theory-specific rules and r be the merging of r_1 and r_2. For every set of rules ER, $\vdash_{ER\cup\{r_1,r_2\}} = \vdash_{ER\cup\{r\}}$.

Proof. Again, the proof follows directly from the Definition 3.31 about derivability of rules. □

We list below auxiliary functions used in the preprocessing procedure:

- *is_general_redundant*: Checks if a rule is general redundant.

- *symbols*: Returns a list of the symbols sorted by arity of a language given as input.
- *is_intelim*: Checks if a symbol is intelim in a set of expansion rules.
- *symb_int_rules*: Given a set ER of theory-specific expansion rules and a function symbol s as input, it returns the symbolic introduction rules of s in ER.
- *symb_elim_rules*: Given a set ER of theory-specific expansion rules and a function symbol s as input, it returns the symbolic elimination rules of s in ER.
- *is_term_intro*: Checks if a rule is a term introduction rule.
- *closure_version*: Returns the closure version of a rule given as input.
- *premises*: Returns the premises of a rule given as input.
- *find_rules_by_premises*: Given a set of premises *prem* and a set of rules *er*, it returns the subset of rules in *er* that have *prem*, as premises.
- *merge_rules*: Returns the merging of a list of rules with the same premises.

The preprocess procedure is described in Algorithm 3. Such procedure is divided into four parts: (i) removal of general redundant rules (lines 2 to 6), (ii) removal of symbolic introduction rules for the intelim symbols (lines 8 to 14), (iii) substitution of term introduction rules by their closure versions (lines 16 to 22) and (iv) merging of rules (lines 24 to 32). By definition, after parts (i) and (ii), the consequence relation induced by *result* is identical to the one induced by the set of rules TS_{Th} given as input. Such identity still holds after parts (iii) and (iv), respectively, due to Lemmas 4.13 and 4.15. Moreover, the four loops are finite, so the procedure terminates.

Now, we apply the preprocessing procedure to the set of rules of Examples 3.76 and 3.77.

Example 4.16. We run the preprocessing procedure for the extracted rules of Example 3.76. The only general redundant rules of it are $-\triangle\mathbf{E}_{1a'}, -\triangle\mathbf{E}_{1b'}, -\triangle\mathbf{E}_{2a'}, -\triangle\mathbf{E}_{2b'}$, $+\triangle\mathbf{I}_{1'}$ and $+\triangle\mathbf{I}_{2'}$. After removing these rules and the symbolic introduction rules of the intelim symbols, the remaining rules are $-\emptyset$ and those labeled with $\mathbf{E}$. Among them, there is only one term introduction rule, which is $-\emptyset\mathbf{E}$. After replacing $-\emptyset\mathbf{E}$ by its closure version and doing all the possible mergings, the preprocessed set of rules is:

$$\frac{+x \in \emptyset}{\divideontimes}\ -\emptyset \qquad \frac{+x \in \mathsf{C}(y)}{-x \in y}\ +\mathsf{CE} \qquad \frac{-x \in \mathsf{C}(y)}{+x \in y}\ -\mathsf{CE}$$

Algorithm 3 preprocess_ts_rules(TS_{Th}, L)

```
result ← TS_Th
for rule in TS_Th do
    if is_general_redundant(rule) then
        result ← result \ {rule}
    end if
end for
aux_result ← result
for symb in symbols(L) do
    if is_int_elim(symb, aux_result) then
        e_rules ← symb_elim_rules(aux_result, symb)
        i_rules ← symb_int_rules(aux_result, symb)
        result ← result \ i_rules
    end if
end for
aux_result ← result
for rule in aux_result do
    if is_term_intro(rule) then
        cl_version ← closure_version(rule)
        result ← result \ {rule}
        result ← result ∪ {cl_version}
    end if
end for
aux_result ← result
for rule in aux_result do
    premises ← get_premises(rule)
    same_premises_rules ← find_rules_by_premises(premises, result)
    merged_rule ← merge_rules(same_premises_rules)
    if not merged_rule in result then
        result ← result \ same_premises_rules
        result ← result ∪ {merged_rule}
    end if
end for
return result
```

$$\frac{-x \in y \quad +x \in y \cup z}{+x \in z}\ +\cup\mathbf{E}_1 \qquad \frac{-x \in z \quad +x \in y \cup z}{+x \in y}\ +\cup\mathbf{E}_2 \qquad \frac{-x \in y \cup z}{-x \in y \mid -x \in z}\ -\cup\mathbf{E}$$

$$\frac{+x \in y \cap z}{+x \in y \quad +x \in z}\ +\cap\mathbf{E} \qquad \frac{+x \in y \quad -x \in y \cap z}{-x \in z}\ -\cap\mathbf{E}_1 \qquad \frac{+x \in z \quad -x \in y \cap z}{-x \in y}\ -\cap\mathbf{E}_2$$

$$\frac{+x \in y \setminus z}{+x \in y \quad -x \in z}\ +\setminus\mathbf{E} \qquad \frac{+x \in y \quad -x \in y \setminus z}{+x \in z}\ -\setminus\mathbf{E}_1 \qquad \frac{-x \in z \quad -x \in y \setminus z}{-x \in y}\ -\setminus\mathbf{E}_2$$

$$\frac{+x \in y \times z}{+x \in fst(y) \mid +x \in snd(z)}\ +\times\mathbf{E} \qquad \frac{+x \in fst(y) \quad -x \in y \times z}{-x \in snd(z)}\ -\times\mathbf{E}_1 \qquad \frac{+x \in snd(z) \quad -x \in y \times z}{-x \in fst(y)}\ -\times\mathbf{E}_2$$

$$\frac{-x \in y \quad +x \in y \triangle z}{+x \in z}\ +\triangle\mathbf{E}_{1a} \qquad \frac{-x \in z \quad +x \in y \triangle z}{+x \in y}\ +\triangle\mathbf{E}_{1b} \qquad \frac{+x \in y \quad +x \in y \triangle z}{-x \in z}\ +\triangle\mathbf{E}_{2a} \qquad \frac{+x \in z \quad +x \in y \triangle z}{-x \in y}\ +\triangle\mathbf{E}_{2b}$$

$$\frac{-x \in y \quad -x \in y \triangle z}{-x \in z}\ -\triangle\mathbf{E}_{1a} \qquad \frac{-x \in z \quad -x \in y \triangle z}{-x \in y}\ -\triangle\mathbf{E}_{1b} \qquad \frac{+x \in y \quad -x \in y \triangle z}{+x \in z}\ -\triangle\mathbf{E}_{2a} \qquad \frac{+x \in z \quad -x \in y \triangle z}{+x \in y}\ -\triangle\mathbf{E}_{2b}$$

$$\frac{+z \in x \quad +x \subseteq y}{+z \in y}\ +\subseteq\mathbf{E}_1 \qquad \frac{-z \in y \quad +x \subseteq y}{-z \in x}\ +\subseteq\mathbf{E}_2 \qquad \frac{-x \subseteq y}{+skl_2^1(x,y) \in x \mid -skl_2^1(x,y) \in y}\ -\subseteq\mathbf{E}$$

$$\frac{+z \in x \quad +x \between y}{-z \in y}\ +\between\mathbf{E}_1 \qquad \frac{+z \in y \quad +x \between y}{-z \in x}\ +\between\mathbf{E}_2 \qquad \frac{-x \between y}{+skl_2^2(x,y) \in x \mid +skl_2^2(x,y) \in y}\ -\between\mathbf{E}$$

Example 4.17. Now, we run the preprocessing procedure for the extracted rules in Example 3.77, in which there are no general redundant rules. The symbolic introduction rules we remove are $+\leq\mathbf{I}$, $-\leq\mathbf{I}$, $+|\mathbf{I}$ and $-|\mathbf{I}$. After this, two term introduction rules remain: $-|\mathbf{E}$ and $-\leq\mathbf{E}$, that are replaced by their respective closure versions. The result of the preprocessing is:

$$\frac{+x \approx y + z}{+x \approx z + y}\ +_{+com} \qquad \frac{-x \approx z + y}{-x \approx y + z}\ -_{+com}$$

$$\frac{+x + y \approx x + z}{+y \approx z}\ +_{+inv} \qquad \frac{-y \approx z}{-x + y \approx x + z}\ -_{+inv}$$

$$\frac{+x \approx w \quad +y \approx z}{+x + y \approx w + z}\ +_{+cong} \qquad \frac{+x \approx w}{-x + y \approx w + z \mid -y \approx z}\ -_{+cong1} \qquad \frac{+y \approx z}{-x + y \approx w + z \mid -x \approx w}\ -_{+cong2}$$

$$\frac{+x \approx y \cdot z}{+x \approx z \cdot y}\ +_{\cdot com} \qquad \frac{-x \approx z \cdot y}{-x \approx y \cdot z}\ -_{\cdot com}$$

$$\frac{+\, x \cdot y \approx x \cdot z}{+\, y \approx z}\ +\cdot_{inv} \qquad \frac{-\, y \approx z}{-\, x \cdot y \approx x \cdot z}\ -\cdot_{inv}$$

$$\frac{\begin{array}{c}+\, x \approx w \\ +\, y \approx z\end{array}}{+\, x \cdot y \approx w \cdot z}\ +\cdot_{cong} \qquad \frac{\begin{array}{c}+\, x \approx w \\ -\, x \cdot y \approx w \cdot z\end{array}}{-\, y \approx z}\ -\cdot_{cong1} \qquad \frac{\begin{array}{c}+\, y \approx z \\ -\, x \cdot y \approx w \cdot z\end{array}}{-\, x \approx w}\ -\cdot_{cong2}$$

$$\frac{+\, x \leq y}{+\, y \approx x + skl_2^1(x,y)}\ +\leq\mathbf{E} \qquad \frac{\begin{array}{c}-\, x \leq y \\ +\, y \approx x + z\end{array}}{\divideontimes}\ -\leq\mathbf{E}$$

$$\frac{+\, x \,|\, y}{+\, y \approx x \cdot skl_2^2(x,y)}\ +|\,\mathbf{E} \qquad \frac{\begin{array}{c}-\, x \,|\, y \\ +\, y \approx x \cdot z\end{array}}{\divideontimes}\ -|\,\mathbf{E}$$

4.2.2 Search for minimal proofs

In this subsection, our goal is to provide a procedure that extracts a set of all minimal proofs for a set of signed formulas. Given a definitional theory $Th := \langle L, Ax \rangle$, a set ER of theory-specific first-order cut-based rules for Th and an ER-provable set of signed formulas SF of L, the informal description of the strategy to produce the set M of minimal ER-proofs for SF consists of the following four steps:

1. From the initial ER-tableau for SF, we construct all possible tableaux with one application of expansion rule of ER. From them, we construct all possible tableaux with two applications of expansion rules of ER and so on.

2. When an ER-proof P for SF is found, we set an upper bound for the size of minimal ER-proofs for SF and add P to M.

3. We continue with the construction described in step 1, searching for other ER-proofs for SF:

   (a) If the size of an ER-proof P is equal to the upper bound set initially, we add P to M.

   (b) If the size m of an ER-proof P is lower than the upper bound set initially, we set a new upper bound m for the size of minimal ER-proofs for SF and empty M adding P to it.

4. At some point, by the amount of applications of expansion rules, it is possible to know that, from that point, the sizes of the proofs exceed the current upper bound. Then, we can terminate the procedure.

Now, we proceed with the formalization of this informal specification. We start by defining a structure that captures the construction described in step 1 and, then, we prove the fact stated in step 4.

Definition 4.18 (Successor/predecessor tableau)**.** Given a set of expansion rules ER and two ER-tableau T and T^*, we say that T^* is an *ER-successor* of T (and T is an *ER-predecessor* of T^*) if T^* was obtained by the application of one expansion rule of ER in T.

Definition 4.19 (Tree of successor tableaux)**.** Given a set of signed formulas SF and a set of expansion rules ER, we construct a tree F whose nodes have as content ER-tableaux such that:

(i) The root of F is the initial ER-tableau for SF.

(ii) For a node n_1 of F, if the ER-tableau T of n_1 is not closed, then the children of n_1 are all the ER-successors of T. Otherwise, n_1 has no children.

A tree such as F is called *tree of successor tableaux* for SF via ER. When there is no risk of ambiguity, we omit the set of expansion rules and simply refer to the tree of successor tableaux for a set of signed formulas. Do note that the tableaux that are obtained by n successive application of expansion rules in the initial ER-tableau for SF are at the level $n+1$ of F.

In Figure 19, we present a fragment of a tree of successor tableaux for $\{+\, p_1 \in p_2, -\, p_1 \in (p_3 \setminus p_4) \cup (p_2 \cup p_5)\}$. As one of the non-displayed nodes, for example, the node 4 has as one of its children a proof in which the rule $-\cup\mathbf{E}$ was applied having $-\, p_1 \in p_2 \cup p_5$ as premise.

One concern that may be raised in the construction of the trees of successor tableaux is the one regarding their finiteness. The theory-specific tableaux we are working with are the result of joining rules extracted from the computational procedure described in Theorem 3.74 and the cut for theory-specific tableaux (Definition 3.81). As a consequence, these sets of expansion rules are always finite. Therefore, the width of a tree of successor tableaux is also always finite.

Regarding the height of a tree of successor tableaux, we cannot claim it is finite. However, for a provable set of signed formulas SF, at least one of the branches of a tree of successor tableaux is finite, which is the one containing a proof for SF. So, we avoid

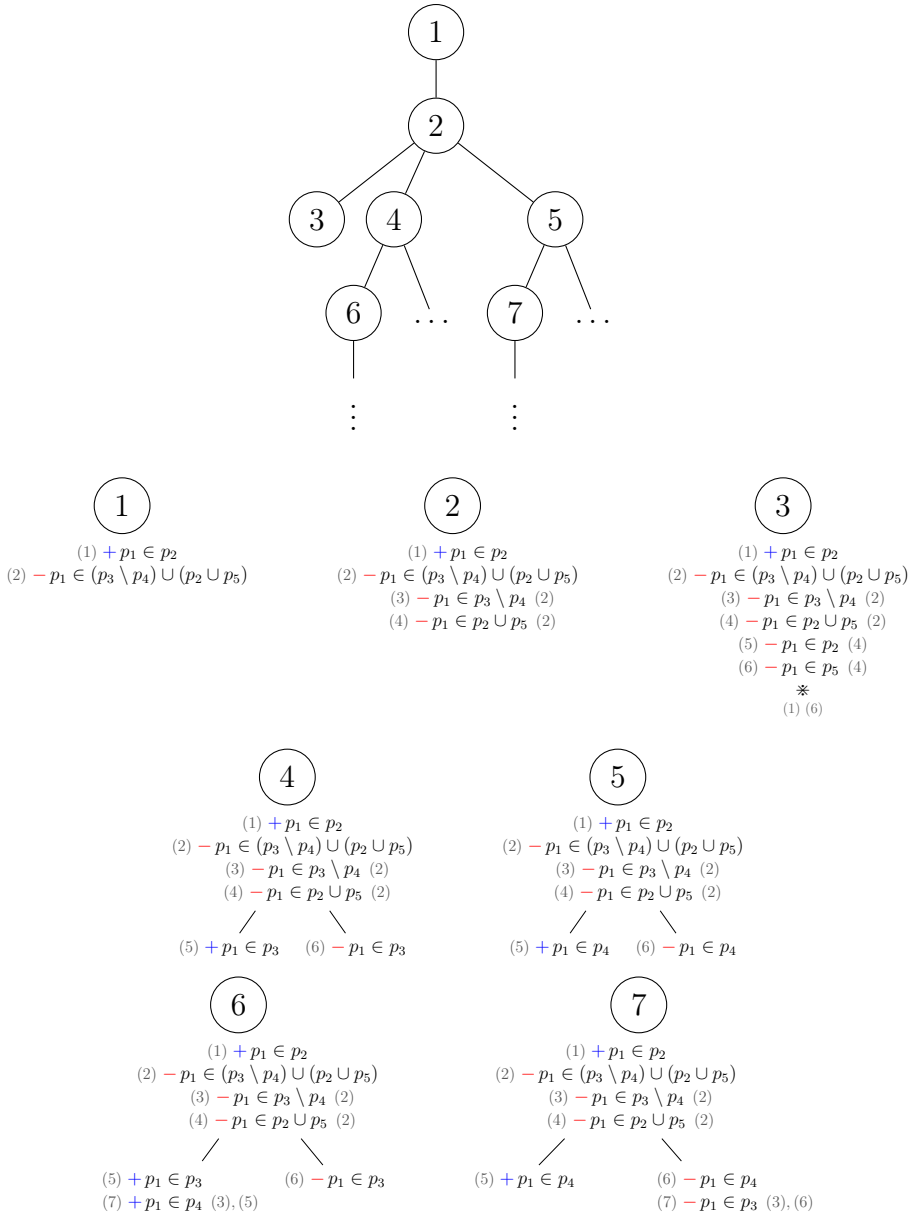


Figure 19: A fragment of a tree of successor tableaux for $\{+p_1 \in p_2, -p_1 \in (p_3 \setminus p_4) \cup (p_2 \cup p_5)\}$.

this termination issue by producing the tree of successor tableaux under demand in a Breadth-First Search (BFS) traversal as our initial informal specification has suggested. Before proceeding with the description of the procedure, we prove the fact of step 4 and define what "point" is this in which the sizes of the proofs exceed the upper bound previously defined.

Theorem 4.20. Given a definitional theory Th, a set ER of first-order theory specific rules for Th and an ER-provable set of signed formulas SF, if P is an ER-proof for SF and it is at the level l of the tree of successor tableaux, then the size of P is greater than or equal to $l + 1$.

Proof. This follows from the fact that every non-eliminable rule application is responsible for increasing in at least one the size of the proof it is part of the construction of, since at least one of the conclusions of this application is in the justification tree. So, a proof P at the level l must have at least l nodes in the justification tree plus the closure node. Thus, the size of P is greater than or equal to $l + 1$.

In the case of non-eliminable rules, as we try to construct all possible tableaux during the extraction of minimal proofs, a clean proof is guaranteed to be found in a level above in the tree of successor tableaux. □

After having laid the groundwork, we are able to provide the desired procedure. Given a definitional theory $Th := \langle L, Ax \rangle$, a set ER of theory-specific rules for Th and an ER-provable set of signed formulas SF of L, the algorithm that extracts the set of minimal ER-proofs for a set of signed formulas SF of L is described in the Algorithm 4. The inputs are a set of signed formulas, sf, representing the set SF and a set of expansion rules, er, representing ER. Recall that we construct the tree of successor tableaux F for SF via ER under demand in a BFS traversal. As usual in a BFS, we use an auxiliary queue, aux_queue, to store the nodes we are going to visit. In this algorithm, ax_queue stores 3-tuples, whose first element is the content of a node n of F, the second element is the level of n in F and the third element is the size of n if n is an ER-proof and -1 otherwise. The role of the upper bound for the size of minimal ER-proofs is played by $minimal\_proof\_size$, which is initialized with -1.

Moreover, in Algorithm 4, there are four auxiliary functions:

- is_closed: Checks if a tableau is closed.
- $get\_successors\_tableaux$: Given a tableau T and a set of rules ER, it produces all ER-successors of T. Recall that, as the amount of rules of ER is finite, there are finite ER-successors for T.
- get_size: Returns the size of a tableau if it is a proof and -1 otherwise.
- $clean$: Returns the clean proof of a proof given as input.

Regarding the termination of the Algorithm 4, we must verify that the **for** loop starting in line 29 and the **while** loop starting in line 5 both finish. The loop starting in line 29 cannot be infinite as the size of $tab\_children$ is finite, so we only add a finite amount of 3-tuples to aux_queue. For the **while** loop starting in line 5, we only push

Algorithm 4 search_min_proofs(sf, er)

```
1: aux_queue ← [ ]
2: aux_queue.push((sf, 1, 0))
3: minimal_proofs ← [ ]
4: minimal_proof_size ← −1
5: while aux_queue.empty() == False do
6:     current := aux_queue.front()
7:     tab := current.first()
8:     level := current.second()
9:     size := current.third()
10:    aux_queue.pop()
11:    if is_closed(tab) == True then
12:        clean_tab ← clean(tab)
13:        if len(minimal_proofs) == 0 then
14:            minimal_proofs ← size
15:            minimal_proofs.add(clean_tab)
16:        else
17:            if size < minimal_proof_size then
18:                minimal_proofs ← size
19:                minimal_proofs.empty()
20:                minimal_proofs.add(clean_tab)
21:            end if
22:            if size == minimal_proof_size then
23:                minimal_proofs.add(clean_tab)
24:            end if
25:        end if
26:    else
27:        if len(minimal_proofs) == 0 or level ≤ minimal_proofs_size + 1 then
28:            tab_children ← get_successors_tableaux(tab, er)
29:            for tab_child in tab_children do
30:                size_child ← get_size(tab_child)
31:                aux_queue.push(tab_child, level + 1, size_child)
32:            end for
33:        end if
34:    end if
35: end while
36: return minimal_proofs
```

3-tuples into aux_queue if either no ER-proof for SF has been found yet or the current level of F is less than or equal to the upper bound for minimal ER-proofs plus one. Since SF is ER-provable, an ER-proof for SF is going to be found in some iteration of such **while** loop and an upper bound for minimal proofs is set. The upper bound might only be updated to lower upper bounds. Moreover, the width of F is finite. So, in some iteration, the level we are traversing in F will be greater than the current upper bound plus one. That is when we stop pushing 3-tuples to aux_queue and the **while** loop finishes after we pop all remaining 3-tuples from aux_queue.

We proceed by proving that the Algorithm 4 outputs the set of all minimal ER-proofs for SF. The condition in line 11, guarantees that all ER-tableaux in the set of minimal proofs are closed ER-tableaux. By contradiction, assume that m is the size of the minimal ER-proofs for SF and that there is an ER-proof P for SF having size n such that $n < m$ and P is not in the set of minimal proofs. We show that P is visited and, from this, a contradiction is immediately derived due to the execution of the commands of lines 18, 19 and 20. Since the size of the minimal proofs is m, the BFS goes until the level $m+1$ of the tree of successor tableaux F for SF. If P is a clean proof, in the worst case (one in which every rule application is responsible for adding only one non-dispensable node in P), P is in the level $n+1$. We know that $n+1 < m+1$, thus P is visited. If P is not a clean proof, by construction, there is a clean proof for P that is, at most, in the level $n+1$ of F.

4.2.3 Search for proof-isomorphic sets of signed formulas

Given a minimal ER-proof M extracted for a set of signed formulas SF of a language L, we provide a procedure that search for all sets of signed formulas of L that have a minimal proof that is deductively isomorphic to M. To exemplify a simple, but not efficient, approach to this, consider the conjecture $\{+\, p_1 \in p_2 \cap (p_3 \cup p_4), -\, p_1 \in (p_2 \cap p_3) \cup p_4\}$ and the minimal proof M for it is described in Figure 20. For every set $\{+\, p_1 \,\square_1\, p_2 \circ_1 (p_3 \circ_2 p_4), -\, p_1 \,\square_2\, (p_2 \circ_1 p_3) \circ_2 p_4\}$ such that $\{\square_1, \square_2\} \subseteq \{\in, \subseteq, \not\subseteq\}$ and $\{\circ_1, \circ_2, \circ_3, \circ_4\} \subseteq \{\cup, \cap, \setminus, \triangle, \times\}$, we could try to construct a proof that is deductively isomorphic to M. However, it would be 5625 cases to test. In this subsection, we show how we can restrict the search space of proof-isomorphic candidate signed formulas.

Consider the case of the conjecture $\{+\, p_1 \in p_2 \cap (p_3 \cup p_4), -\, p_1 \in (p_2 \cap p_3) \cup p_4\}$ again. It would not make sense to check if there is a minimal proof for $\{+\, p_1 \in p_2 \triangle (p_3 \cup p_4), -\, p_1 \subseteq (p_2 \cap p_3) \cup p_4\}$ that is deductive isomorphic to M. First, after an application of the

rule $-\subseteq\mathbf{E}$ having as premise $-\, p_1 \subseteq (p_2 \cap p_3) \cup p_4$, the skolemization symbol skl_2^1 should appear in the proof and there is no skolemization symbol in M. Second, all the rules involving $\triangle$ are 2-premise rules.

$$
\begin{array}{c}
(1)\ +\, p_1 \in p_2 \cap (p_3 \cup p_4) \\
(2)\ -\, p_1 \in (p_2 \cap p_3) \cup p_4 \\
(3)\ +\, p_1 \in p_2 \ \ (1) \\
(4)\ +\, p_1 \in p_3 \cup p_4 \ \ (1) \\
(5)\ -\, p_1 \in p_2 \cap p_3 \ \ (2) \\
(6)\ -\, p_1 \in p_4 \ \ (2) \\
(7)\ -\, p_1 \in p_3 \ \ (3), (5) \\
(8)\ +\, p_1 \in p_3 \ \ (4), (6) \\
\maltese \\
(7)\ (8)
\end{array}
$$

Figure 20: A minimal proof M for $\{+\, p_1 \in p_2 \cap (p_3 \cup p_4), -\, p_1 \in (p_2 \cap p_3) \cup p_4\}$.

Informally, given a minimal ER-proof M extracted for a set of signed formulas SF of a language L, what we do is to search for all the symbols that "make sense" to replace a symbol of a formula in SF by a symbol of L considering the structure of M and the rules of ER. We formalize such notion in what follows.

Definition 4.21 (Occurrence of a symbol in a formula)**.** The *occurrence n of a symbol s on a formula φ* is the n-th occurrence of s in φ considering its concrete (that is, sequential) representation.

In a theory-specific formula φ, a predicate symbol p has at most one occurrence in φ. However, the same cannot be claimed about function symbols. This is why we introduce the following definition.

Definition 4.22 (Syntactic matching symbol)**.** Let L be a language and $\varphi_1, \varphi_2 \in L$ such that φ_1 and φ_2 are syntactically isomorphic. Now, let p_1 and p_2 be two symbols of L that appear, respectively, in φ_1 and φ_2. We say that p_2 is a *syntactic matching symbol occurrence* of the occurrence o_1 of p_1 with respect to φ_1 and φ_2 if p_2 appears in the same place of the abstract representation of φ_2 that the occurrence o_1 of p_1 appear in the abstract representation of φ_1. When there is only one occurrence of s_1 in φ_1, we simply say that s_2 is a syntactic matching symbol of s_1 with respect to φ_1 and φ_2.

For example, for the formulas $\varphi_1 := p_1 \in (p_2 \setminus p_3) \cap p_4$ and $\varphi_2 := p_1 \in (p_2 \triangle p_3) \cup p_4$, the symbol $\setminus$ is a syntactic matching symbol of $\triangle$ with respect to φ_1 and φ_2. By definition, a predicate symbol p appears in a abstract representation R of a theory-specific formula

if p is in the root of R. Hence, predicate symbols can only match predicate symbols and function symbols can only match function symbols.

Definition 4.23 (Deductive matching symbol)**.** Let s_1 and s_2 be two symbols of a language L. A symbol s_2 is a *deductive matching symbol* of the occurrence o_1 of s_1 in a premise $\circ\varphi_1$ of a rule $r_1 \in ER$ when there is a rule r_2 in ER such that:

(i) The premises of r_1 and the premises of r_2 are syntactically isomorphic,

(ii) The conclusion of r_1 is isomorphic to the conclusion of r_2 and

(iii) The symbol s_2 appears in a premise $\circ\varphi_2$ of r_2 such that φ_1 and φ_2 are isomorphic and s_2 is a syntactic matching symbol of the occurrence o_1 of s_1 with respect to φ_1 and φ_2.

Again, when there is only one occurrence of s_1 in the premises of r_1, we simply say that s_2 is a deductive matching symbol of s_1 with respect to the rule r_1.

In the set of rules of Example 4.16, the set of deductive matching symbols of $\cap$ with respect to the rule $+\cap\mathbf{E}$ is $\{\cap, \cup, \setminus\}$. But, in the same set of rules, the set of deductive matching symbols of $\cap$ with respect to the rule $-\cap\mathbf{E}_1$ is $\{\cap, \cup, \setminus, \triangle\}$. In the set of rules of Example 4.17, the only deductive matching symbols of $|$ with respect to the rule $-|\mathbf{E}$ is $\leq$.

Definition 4.24 (Justification nodes)**.** Let r be an n-premise theory-specific rule, whose premises are $prem_1, \ldots, prem_n$. The rule r is applied on a tableau T if there are nodes $m_1, \ldots, m_n$ in T and a substitution σ such that $sign(prem_1) = sign(m_1), \ldots, sign(prem_n) = sign(m_n)$ and σ unifies the formulas $fmla(prem_1), \ldots, fmla(prem_n)$ with the formulas $fmla(m_1), \ldots, fmla(m_n)$ respectively. In this case, we say that the nodes $m_1, \ldots, m_n$ justify the application of r in T and that m_i is the *justification node* of the premise $prem_i$ for this application of r in T via σ, where $1 \leq i \leq n$.

In the proof M, the nodes (2) and (4) justify the application of $-\cap\mathbf{E}_1$, and the node (1) justifies the application of $+\cap\mathbf{E}$ in such proof. Besides, the node (1) is the justification of the premise $+\, x \in y \cap z$ of $+\cap\mathbf{E}$ via a substitution σ defined as $\sigma(x) = p_1$, $\sigma(y) = p_2$ and $\sigma(z) = p_3 \cup p_4$.

Definition 4.25 (Justification matching symbol occurrence)**.** Let ER be a set of theory-specific rules and T be an ER-tableau. Now, let r be a rule of ER having one of its

premises the signed formula $prem$, n be a node of T such that n is the justification of $prem$ for an application of r in T via a substitution σ and s be a symbol that occurs in $prem$ and in n. We say that the occurrence o_2 of s in $prem$ is the *justification matching symbol occurrence* of the occurrence o_1 of s in n if the occurrence o_2 of s appears in the same place of the abstract representation of $\sigma(fmla(prem), \emptyset)$ that the occurrence o_1 of p_1 appear in the abstract representation of $fmla(n)$.

For example, in the proof M, the justification matching symbol of the only occurrence of $\cap$ in the node (1) is its only occurrence in the premise $+\, x \in y \cap z$. We could, then, argue that it would make sense to try to replace, in the procedure we aim to present, $\cap$ in (1) by any of the symbols of the set $\{\cap, \cup, \setminus\}$, since this set represents the deductive matching symbols of the occurrence of $\cap$ in the premise of $+\cap\mathbf{E}$. However, the occurrence of $\cup$ in the node (1) has no matching justification symbols, since $\cup$ does not even appear in the premise of $+\cap\mathbf{E}$. For the case of such occurrences, we describe the next definitions.

Definition 4.26 (Direct descendant occurrences of a function symbol)**.** Let m be a direct descendant node of n in a proof P. Then, there is a substitution σ and a rule r having as premises $prem_1, \ldots, prem_n$ and conclusion $conc$ such that $\sigma(prem_i, \emptyset) = n$ for some $1 \leq i \leq n$ and $\sigma(conc, \emptyset) = m$. For every occurrence o of a function symbol f in n, if $x \in Var(prem_i)$ and the occurrence o of f is the result of the substitution of x in $prem_i$ via σ, then the occurrences of f in m that resulted from the substitution of x in $conc$ via σ are called *direct descendant occurrences* of f in P.

In Figure 21, the occurrence 2 of $\cap$ in (3), highlighted in green, is a direct descendant occurrence 3 of $\cap$ in (1), highlighted in blue. The occurrence 1 of $\cap$ in (1) has no direct descendant occurrences and the same holds for the occurrence of $\cap$ in (4), highlighted in red.

$$
\begin{array}{c}
(1)\ +\, p_1 \in p_2 \cap (p_3 \cap (p_4 \cap p_5)) \\
(2)\ -\, p_1 \in p_5 \\
(3)\ +\, p_1 \in p_3 \cap (p_4 \cap p_5)\ \ (1) \\
(4)\ +\, p_1 \in p_4 \cap p_5\ \ (3) \\
(5)\ +\, p_1 \in p_5\ \ (4) \\
\maltese \\
(2)\ (5)
\end{array}
$$

Figure 21: A minimal proof for $\{+\, p_1 \in p_2 \cap (p_3 \cap (p_4 \cap p_5)), -\, p_1 \in p_5\}$.

Definition 4.27 (Descendant occurrences of a function symbol)**.** Let P be a proof and n be a node of P. The set of *descendant occurrences* of the occurrence o of a function

symbol f in n is the reflexive-transitive closure of the direct descendant occurrences of the occurrence o of f in P.

Again in Figure 21, the descendant occurrences of the occurrences of $\cap$ highlighted in blue, are the occurrences of it highlighted in blue, green and red. Now, we can define the set of candidate signed formulas we restrict our search in this step of the generation.

Definition 4.28 (Proof-isomorphic predicate symbols)**.** Let SF be a set of signed formulas of L, ER be a set of expansion rules, T be an ER-proof for SF and $sf \in SF$ be such that the node n that has sf as signed formula is a justification for an application of the rule r of ER. If p is a predicate symbol that occurs in sf, then the set of *proof-isomorphic predicate symbols* of the occurrence o of p in sf is either:

(i) The set of deductive matching symbols of p' in ER with respect to r if p' is the matching justification of the occurrence o of p in T, or

(ii) The set of predicate symbols of L if p does not have a matching justification symbol in T.

Definition 4.29 (Proof-isomorphic function symbols)**.** Let SF be a set of signed formulas of L, ER be a set of expansion rules, T be an ER-proof for SF and $sf \in SF$ be such that the node n that has sf as signed formula is a justification for an application of the rule r of ER. For the occurrence o of a symbol f is a function symbol in sf, the set of *proof-isomorphic function symbols* of the occurrence o of f in sf is:

(i) The set of deductive matching symbols of all matching justification symbols of descendant occurrences of f in T.

(ii) The set of function symbols of L if none of the descendant occurrences of f in T have matching justification symbols in T.

Definition 4.30 (Proof-isomorphic candidate set of signed formulas)**.** Let SF be a set of signed formulas, ER be a set of expansion rules and P be an ER-proof for SF. A set of signed formulas SF' is a *proof-isomorphic candidate set of signed formulas* for SF with respect to P in ER if $SF' = SF$ or SF' is the result of replacing at least one occurrence of predicate (or function) symbol s of a signed formula $sf \in SF$ by a proof-isomorphic predicate (or function) symbol of such s in sf.

Consider set $\{+\,p_1 \in p_2 \cap (p_3 \cup p_4), -\,p_1 \in (p_2 \cap p_3) \cup p_4\}$ in the proof of Figure 20. We calculate the size of the set of proof-isomorphic candidate signed formulas for them. For this, we refer to $+\,p_1 \in p_2 \cap (p_3 \cup p_4)$ as sf_1, and to $-\,p_1 \in (p_2 \cap p_3) \cup p_4$ as sf_2. The occurrence of $\cap$ in sf_1 and the occurrence of $\cup$ in sf_2 may be replaced by either $\cap$, $\cup$ or $\setminus$. The occurrence of $\cap$ in sf_1 and the occurrence of $\cup$ in sf_2 may be replaced by either $\cap$, $\cup$, $\setminus$ or $\triangle$. Then, the size of the set is $3 \times 3 \times 4 \times 4 = 144$ and this reduces almost, for this example, 40 times the amount of cases to be tested.

After all these definitions, we can describe our procedure for searching for proof-isomorphic sets of signed formulas. The auxiliary functions for the search for proof-isomorphic sets of signed formulas are listed below:

- $get\_sf\_candidates$: Given a set of signed formulas sf, a set of expansion rules er and a minimal proof m, it returns the set of proof-isomorphic candidate set of signed formulas for sf with respect to m in er.
- $is\_proof\_isomorphic\_sf\_set$: Given a set of expansion rules er, a set of signed formulas sf and a proof p, it checks if it is possible to construct an er-proof for sf that is deductively isomorphic to p.

In the Algorithm 5, we present the procedure for searching for proof-isomorphic sets of signed formulas. Consider a definitional theory $Th := \langle L, Ax \rangle$, a set ER of theory-specific rules for Th and an ER-provable set of signed formulas SF of L, and a set M of minimal ER-proofs for SF. The inputs are a set of signed formulas sf representing SF, a set of expansion rules er representing ER, and a set of ER-proofs min_proofs representing M. The procedure consists of two finite nested loops in which we check if it is possible to construct an isomorphic proof to m for a set of signed formulas $sf_{candidate}$ such that $m \in min\_proofs$ and $sf_{candidate}$ is a candidate proof-isomorphic set of signed formulas for sf with respect to m. When that is the case, $sf_{candidate}$ is added to the output set $result$.

The sets of signed formulas that result from running the procedure of Algorithm 5 giving as input the set of signed formulas $SF := \{+\,p_1 \in p_2 \cap (p_3 \cup p_4), -\,p_1 \in (p_2 \cap p_3) \cup p_4\}$, the set of expansion rules ER of Example 4.16 and the minimal ER-proofs for SF are listed below:

- $\{+\,p_1 \in p_2 \cap (p_3 \cup p_4), -\,p_1 \in (p_2 \cap p_3) \cup p_4\}$
- $\{+\,p_1 \in p_2 \cap (p_3 \triangle p_4), -\,p_1 \in (p_2 \cap p_3) \cup p_4\}$

Algorithm 5 search_proof_isomorphic_sf(sf, er, min_proofs)

```
result ← { }
for m in min_proofs do
    sf_candidates ← get_sf_candidates(sf, er, m)
    for sf_candidate in sf_candidates do
        if is_proof_isomorphic_sf_set(m, er, sf_candidate) == True then
            result ← result ∪ {sf_candidate}
        end if
    end for
end for
return result
```

- $\{+\, p_1 \in p_2 \setminus (p_3 \setminus p_4), -\, p_1 \in (p_2 \setminus p_3) \cup p_4\}$
- $\{+\, p_1 \in p_2 \setminus (p_3 \triangle p_4), -\, p_1 \in (p_2 \setminus p_3) \cup p_4\}$
- $\{+\, p_1 \in p_2 \setminus (p_3 \setminus p_4), -\, p_1 \in (p_2 \triangle p_3) \cup p_4\}$
- $\{+\, p_1 \in p_2 \setminus (p_3 \triangle p_4), -\, p_1 \in (p_2 \triangle p_3) \cup p_4\}$

A minimal proof for the fourth set of this list is illustrated in Figure 22. Do note that this proof is deductively isomorphic to the proof M of Figure 20. In the end, what we have in practical terms is that two proofs exercises such as "Prove that $a \in b \cap (c \cup d)$ implies $a \in (b \cap c) \cup d$" and "Prove that $a \in b \setminus (c \triangle d)$ implies $a \in (b \setminus c) \cup d$" have a comparable level of complexity.

$$
\begin{array}{c}
(1)\ +\, p_1 \in p_2 \setminus (p_3 \triangle p_4) \\
(2)\ -\, p_1 \in (p_2 \setminus p_3) \cup p_4 \\
(3)\ +\, p_1 \in p_2 \ \ (1) \\
(4)\ -\, p_1 \in p_3 \triangle p_4 \ \ (1) \\
(5)\ -\, p_1 \in p_2 \setminus p_3 \ \ (1) \\
(6)\ -\, p_1 \in p_4 \ \ (1) \\
(7)\ +\, p_1 \in p_3 \ \ (3), (5) \\
(8)\ -\, p_1 \in p_3 \ \ (4), (6) \\
\lightning \\
(7)\ (8)
\end{array}
$$

Figure 22: A minimal proof for $\{+\, p_1 \in p_2 \setminus (p_3 \triangle p_4), -\, p_1 \in (p_2 \setminus p_3) \cup p_4\}$.

4.3 Tackling the challenges

We finished Chapter 2 summarizing three challenges currently faced in the development of AQG tools: *a priori metrics*, *subjectivity* and *lack of adaptability*. In this Section,

we comment on how a tool implemented using our method tackles these challenges. Starting by the metric of complexity, the proving complexity of a proof exercise is calculated in terms of a minimal solution for it. Then, our measurement of complexity is not a priori, since the effort to solve the exercise provided as input is taken into account.

Considering subjectivity, it is evident that the classification of two proof exercises as having a comparable level of complexity by our method is not affected by different human judgements. As we remark on Section 5.2 about future works, this classification may be empirically refined based on pedagogical experiments. It might be, for example, that the fewer one-premise rules are present in a proof the more difficulty students have to construct it. In this case, we could attribute a weight in terms of the breadth of the justification tree when calculating its size.

Lastly, there is the problem of adaptability. The input exercise of our method is assumed to be hand-curated by the tutor who is going to use it. So, if the user provides an exercise they consider easy, the method generates also easy exercises according to our classification. Again, this can be revealed different in practice and students may have more difficulty in solving an exercise E_1 in comparison to another exercise E_2 even if E_1 and E_2 have a comparable proving complexity. But we are only capable of asserting this after the method is tested with tutors and students.

5 Conclusion

This work aimed at proposing a method for the generation of proof exercises with comparable level of complexity. The effectiveness of our method was demonstrated through two case studies: the definitional theories of sets and numbers, that are, respectively, fragments of Set Theory and Number Theory. We started Chapter 2 by presenting an overview of the current development of AQG tools. Chapter 2 finished with the summary of three challenges observed in the literature that we intend the proposed method to tackle.

In Chapter 3, we defined a notion of first-order cut-based tableau proofs whose formulas do not contain logical symbols, the so-called theory-specific proofs, to be employed in our method. The rules used in theory-specific proofs are extracted from a definitional theory through a mechanical procedure, described in Theorem 3.74. Considering consequence relations whose formulas in both the antecedent and succedent are atomic, the set of rules this procedure extracts was proved to have the same deductive strength than the definitional theory from which the rules are extracted.

The method for the generation of proof exercises of comparable complexity is presented in Chapter 4. The first section of this chapter is dedicated to formalizing the notion of comparable complexity between proof exercises. In the second Section, the method itself is described. In the final third Section, we comment on how our proposed method tackles the challenges posed in Chapter 2. A prototype implementation of the method is presented in Appendix A.

In the remainder of this Conclusion, we describe, in Section 5.1, limitations that we have identified specially regarding the definitional theory of numbers and, in Section 5.2, we outline perspectives for future works.

5.1 Limitations

Almost all examples we present from our case studies throughout this work are from the definitional theory of sets. This is not by chance. For this theory, the definitional axioms (recall Example 3.16), except for $[ax\emptyset]$, have the format $(\forall x_1, \ldots, x_n)(\varphi_1 \leftrightarrow \varphi_2)$ such that one of the symbols that appears in φ_1 does not appear in φ_2. For instance, in the definitional axiom $(\forall x, y, z)(x \in y \cup z \leftrightarrow (x \in y \vee x \in z))$, the symbol $\cup$ appears in $x \in y \cup z$, but not in $x \in y \vee x \in z$. Therefore, it was easier to impose restrictions in such a way that, in the end, we have a set of elimination rules with which it is possible to construct tableau proofs with some sort of analyticity.

By the way the definitional axioms from the definitional theory of sets were designed, we can search for proofs in such theory directed towards finding as the justifications for a closure, signed formulas having one of the following four formats: $\circ\, x \in p_i$, $\circ\, x \in fst(p_i)$, $\circ\, x \in snd(p_i)$ or $+\, x \in \emptyset$, where x is a term, p_i is a parameter and $\circ \in \{+, -\}$. What is common between them is that they cannot be justification for a one-premise rule which has at least one conclusion.

The limitation with the definitional theory of numbers is not to present theory-specific proofs for sets of signed formulas that could be applied in practical educational experiments (see the examples in Figures 14 and 15), but rather executing the search for minimal proofs. Some rules from Example 4.17, such as $+{+}_{com}$ and $+{\cdot}_{com}$, have conclusions that are isomorphic to their premises. Thus, it becomes harder to find a format for the signed formulas that justify a closure as it is possible in the definitional theory of sets.

5.2 Future works

Our perspectives for future works consist of: (i) improvements in the method and in its implementation and (ii) pedagogical applications for the method.

5.2.1 Improvements in the method and the prototype implementation

There are two immediate refinements in the prototype implementation we visualize:

1. Implementation of a parser that allows formulas in the input files to be written in infix notation.

2. Error handling for input not written according the desired format.

The other refinement is regarding the performance of the prototype. As we describe in Algorithm 5, the search for proof-isomorphic sets of signed formulas is implemented in a nested loop. The steps of each iteration are independent from one another, which means that they may be executed in parallel. As a future work, we aim to implement a parallelism mechanism for this search procedure.

After implementing these improvements, we aim to make the implementation available in a web platform to run experiments in real pedagogical environments. In order to validate our metric of proving complexity, one of the aspects that we aim to focus is in the analysis of possible proof characteristics that might reveal to be more difficult to students. For example, let E_1 and E_2 be two exercises with a comparable level of complexity. For E_1, there is a minimal proof with only one-premise rule and all minimal proofs for E_2 contains at least one rule that is not one-premise. It might be the case that students have more difficult to solve exercise E_2 than the solve exercise E_1.

Initially, our method was designed to receive as input hand-curated exercises and expansion rules. During the research, we have realized the necessity of restricting the rules so as to avoid the introduction of new terms in a proof. With the aim of making the method flexible in this respect, we plan to work on the adjustment of our method so that this restriction is no longer necessary and, hence, the expansion rules can be hand-curated.

Considering the pedagogical applications of the method, the proving complexity of an exercise should not be affected if we permute its parameters or swap terms that are the input of some specific symbols. For example, if φ^* is the result of replacing $x \cap y$ by $y \cap x$ in φ and $\{- \varphi\}$ is provable, then it is possible to prove that $\{- \varphi^*\}$ remains provable and that a minimal proof for $\{- \varphi\}$ is deductively isomorphic to a minimal proof for $\{- \varphi^*\}$. Then, the exercises $\{- p_1 \cap p_2 \subseteq p_1 \setminus p_2\}$, $\{- p_2 \cap p_1 \subseteq p_1 \setminus p_2\}$ and $\{- p_2 \cap p_1 \subseteq p_2 \setminus p_1\}$ should all have a comparable level of proving complexity. In the future, we intend to explore these features and incorporate them in our method, thus enhancing its generative potential.

5.2.2 Integration with a teaching logic suite

The method we present in this work was intended to be integrated into a teaching logic suite. In one side of this suite, tutors could assess their students with prove/refute exercises controlling their level of complexity. On the other side, students could solve such

exercises through a proof assistant and would be automatically graded by the system. We intend also to work on the implementation of the modules of this suite.

The interaction with the proof assistant is intended to be mediated through a custom language whose commands are similar to proof strategies employed in pen-and-paper informal proofs, as in [19] and [27]. These commands should make explicit the formal rules they are referring to in order to avoid that the same code is used to solve different proof exercises with a comparable level of complexity. From a pedagogical perspective, the adoption of proof assistants may benefit the learning process since they can instantaneously give feedback to the students about the correctness of their proof steps.

The generation of proof exercises with controlled level of complexity still lacks refinements to be tested in a real educational environment, as discussed in Section 5.2.1. Regarding the development of a similar method for refutable exercises, the matter is a bit more complicated. While it is only necessary to provide a proof to solve a proof exercise, to solve a refutable exercise, it is necessary to construct a model and verify that this model is a countermodel for the exercise statement. Then, a fundamental question is posed: How to define the complexity of a refutable exercise? Using the theory-specific tableaux as a recipe to the construction of the countermodels, we aim to find an answer to this question by working with examples from the two case studies of the theory of sets and theory of numbers as a starting point.

Bibliography

[1] Said Al Faraby, Adiwijaya Adiwijaya, and Ade Romadhony. "Review on Neural Question Generation for Education Purposes". In: *International Journal of Artificial Intelligence in Education* 34.3 (Sept. 2024), pp. 1008–1045. ISSN: 1560-4306. DOI: 10.1007/s40593-023-00374-x. URL: https://doi.org/10.1007/s40593-023-00374-x (cit. on pp. 13, 16).

[2] Samah AlKhuzaey et al. "Text-based Question Difficulty Prediction: A Systematic Review of Automatic Approaches". In: *International Journal of Artificial Intelligence in Education* 34.3 (Sept. 2024), pp. 862–914. ISSN: 1560-4306. DOI: 10.1007/s40593-023-00362-1. URL: https://doi.org/10.1007/s40593-023-00362-1 (cit. on p. 17).

[3] Tahani Alsubait. "ONTOLOGY-BASED MULTIPLE-CHOICE QUESTION GENERATION". PhD thesis. University of Manchester, 2015. URL: https://research.manchester.ac.uk/en/studentTheses/ontology-based-multiple-choice-question-generation (cit. on pp. 13, 15, 16).

[4] Franz Baader et al. *An Introduction to Description Logic*. 1st. Cambridge: Cambridge University Press, 2017. DOI: https://doi.org/10.1017/9781139025355 (cit. on p. 17).

[5] Luca Benedetto et al. "A Survey on Recent Approaches to Question Difficulty Estimation from Text". In: *ACM Comput. Surv.* 55.9 (Jan. 2023). ISSN: 0360-0300. DOI: 10.1145/3556538. URL: https://doi.org/10.1145/3556538 (cit. on p. 17).

[6] Walter Carnielli, Marcelo E. Coniglio, and João Marcos. "Logics of Formal Inconsistency". In: *Handbook of Philosophical Logic*. Ed. by D.M. Gabbay and F. Guenthner. Dordrecht: Springer Netherlands, 2007, pp. 1–93. ISBN: 978-1-4020-6324-4. DOI: 10.1007/978-1-4020-6324-4_1. URL: https://doi.org/10.1007/978-1-4020-6324-4_1 (cit. on p. 65).

[7] Marcello D'Agostino. *INVESTIGATIONS INTO THE COMPLEXITY OF SOME PROPOSITIONAL CALCULI*. Tech. rep. PRG88. OUCL, Nov. 1990, p. 135. URL:

https://www.cs.ox.ac.uk/publications/publication3869-abstract.html (cit. on pp. 13, 28, 29, 39).

[8] Marcello D'Agostino and Marco Mondadori. "The Taming of the Cut. Classical Refutations with Analytic Cut". In: *Journal of Logic and Computation* 4 (June 1994), pp. 285–319. DOI: 10.1093/logcom/4.3.285 (cit. on pp. 13, 28, 29).

[9] Bidyut Das et al. "Automatic question generation and answer assessment: a survey". In: *Research and Practice in Technology Enhanced Learning* 16.1 (Mar. 2021), p. 5. ISSN: 1793-7078. DOI: 10.1186/s41039-021-00151-1. URL: https://doi.org/10.1186/s41039-021-00151-1 (cit. on p. 15).

[10] Susanna S. Epp. *Discrete Mathematics with Applications*. 4th. USA: Brooks/Cole Publishing Co., 2010. ISBN: 0495391328 (cit. on pp. 68, 69).

[11] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. 2nd ed. Berlin, Heidelberg: Springer-Verlag, 1996. ISBN: 0387945938. DOI: https://doi.org/10.1007/978-1-4612-2360-3 (cit. on pp. 39, 42, 44, 48).

[12] Foteini Grivokostopoulou, Isidoros Perikos, and Ioannis Hatzilygeroudis. "Difficulty Estimation of Exercises on Tree-Based Search Algorithms Using Neuro-Fuzzy and Neuro-Symbolic Approaches". In: *Advances in Combining Intelligent Methods: Post-proceedings of the 5th International Workshop CIMA-2015, Vietri sul Mare, Italy, November 2015 (at ICTAI 2015)*. Ed. by Ioannis Hatzilygeroudis, Vasile Palade, and Jim Prentzas. Cham: Springer International Publishing, 2017, pp. 75–91. ISBN: 978-3-319-46200-4. DOI: 10.1007/978-3-319-46200-4_4. URL: https://doi.org/10.1007/978-3-319-46200-4%5C_4 (cit. on p. 18).

[13] Dorit Hutzler et al. "Learning Methods for Rating the Difficulty of Reading Comprehension Questions". In: *2014 IEEE International Conference on Software Science, Technology and Engineering*. 2014, pp. 54–62. DOI: 10.1109/SWSTE.2014.16. URL: https://doi.org/10.1109/SWSTE.2014.16 (cit. on p. 18).

[14] Ghader Kurdi et al. "A Systematic Review of Automatic Question Generation for Educational Purposes". In: *International Journal of Artificial Intelligence in Education* 30.1 (Mar. 2020), pp. 121–204. ISSN: 1560-4306. DOI: 10.1007/s40593-019-00186-y. URL: https://doi.org/10.1007/s40593-019-00186-y (cit. on pp. 13, 15, 16).

[15] Ghader Kurdi et al. “A comparative study of methods for a priori prediction of MCQ difficulty”. In: *Semantic Web* 12.3 (2021), pp. 449–465. DOI: 10.3233/SW-200390. eprint: https://journals.sagepub.com/doi/pdf/10.3233/SW-200390. URL: https://journals.sagepub.com/doi/abs/10.3233/SW-200390 (cit. on p. 17).

[16] John A. N. Lee. “History of Computing in Education”. In: *History of Computing in Education*. Ed. by John Impagliazzo and John A. N. Lee. New York, NY: Springer US, 2004, pp. 1–16. ISBN: 978-1-4020-8136-1. DOI: 10.1007/1-4020-8136-7_1. URL: https://doi.org/10.1007/1-4020-8136-7%5C_1 (cit. on p. 15).

[17] Sathiamoorthy Manoharan. “Personalized Assessment as a Means to Mitigate Plagiarism”. In: *IEEE Transactions on Education* 60.2 (2017), pp. 112–119. DOI: 10.1109/TE.2016.2604210. URL: https://doi.org/10.1109/TE.2016.2604210 (cit. on p. 17).

[18] Sonia Marin et al. “From axioms to synthetic inference rules via focusing”. In: *Annals of Pure and Applied Logic* 173.5 (2022), p. 103091. ISSN: 0168-0072. DOI: https://doi.org/10.1016/j.apal.2022.103091. URL: https://www.sciencedirect.com/science/article/pii/S0168007222000057 (cit. on p. 39).

[19] Patrick Massot. “Teaching Mathematics Using Lean and Controlled Natural Language”. In: *15th International Conference on Interactive Theorem Proving (ITP 2024)*. Ed. by Yves Bertot, Temur Kutsia, and Michael Norrish. Vol. 309. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 27:1–27:19. ISBN: 978-3-95977-337-9. DOI: 10.4230/LIPIcs.ITP.2024.27. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2024.27 (cit. on p. 100).

[20] Adolfo Neto and Marcelo Finger. “A KE tableau for a logic of formal inconsistency”. In: *Proceedings of TABLEAUX’07 position papers and Workshop on Agents, Logic and Theorem Proving* (Jan. 2007), p. 15 (cit. on p. 65).

[21] Seymour Papert and Cynthia Solomon. “Twenty Things to Do with a Computer”. In: *Educational Technology* 12.4 (1972), pp. 9–18. ISSN: 00131962. URL: http://www.jstor.org/stable/44417821 (visited on 03/13/2025) (cit. on p. 15).

[22] Ariel J. Roffe and Joaquin S. Toranzo Calderon. *Random Formula Generators*. 2021. arXiv: 2110.09228 [cs.LO]. URL: https://arxiv.org/abs/2110.09228 (cit. on p. 18).

[23] Rohit Singh, Sumit Gulwani, and Sriram Rajamani. "Automatically Generating Algebra Problems". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 26.1 (Sept. 2021), pp. 1620–1628. DOI: 10.1609/aaai.v26i1.8341. URL: https://ojs.aaai.org/index.php/AAAI/article/view/8341 (cit. on p. 18).

[24] Raymond Merrill Smullyan. *First-Order Logic*. New York [etc.]: Springer Verlag, 1968. DOI: https://doi.org/10.1007/978-3-642-86718-7 (cit. on pp. 28, 31, 48).

[25] Patrick Terrematte. "A integração do tutorial interativo TryLogic via IMS Learning Tools Interoperability: construindo uma infraestrutura para o ensino de Lógica através de estratégias de demonstração e refutação". MA thesis. Natal, RN, Brasil: UFRN, June 2013. URL: https://repositorio.ufrn.br/handle/123456789/18685 (cit. on p. 18).

[26] Ke Wang and Zhendong Su. "Dimensionally guided synthesis of mathematical word problems". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. IJCAI'16. New York, New York, USA: AAAI Press, 2016, pp. 2661–2668. ISBN: 9781577357704. URL: https://dl.acm.org/doi/abs/10.5555/3060832.3060993 (cit. on p. 19).

[27] Jelle Wemmenhove et al. "Waterproof: Educational Software for Learning How to Write Mathematical Proofs". In: *Electronic Proceedings in Theoretical Computer Science* 400 (Apr. 2024), pp. 96–119. ISSN: 2075-2180. DOI: 10.4204/eptcs.400.7. URL: http://dx.doi.org/10.4204/EPTCS.400.7 (cit. on p. 100).

[28] Dylan Wiliam and Marnie Thompson. "The Future of Assessment: Shaping Teaching and Learning". In: ed. by Carol Anne Dwyer. 1st. Routledge, 2008. Chap. Integrating Assessment with Learning: What Will It Take to Make It Work?, pp. 53–82. DOI: 10.4324/9781315086545-3. URL: https://doi.org/10.4324/9781315086545-3 (cit. on p. 16).

[29] Yicheng Yang et al. *Automatic question generation for propositional logical equivalences*. 2024. arXiv: 2405.05513 [cs.CL]. URL: https://arxiv.org/abs/2405.05513 (cit. on p. 18).

APPENDIX A – Implementation of the generation of proof exercises with a comparable level of complexity

In this Appendix, we present a prototype implementation for the generation of proof exercises with a comparable level of complexity described in Section 4.2, whose big picture is illustrated in Figure 17. The rules given as input are assumed to be preprocessed by the procedure described in the Subsection 4.2.1. As we comment in Section A.2, besides the two inputs described in Figure 17, it is required that a description of the symbols of the language is also provided as input.

The project was implemented using `C++` and the code is available at `https://github.com/joaomendesln/GePECC`. The inputs are provided as text files and the output is displayed in the terminal in which the program is executed. Moreover, the program also outputs the result of the intermediate step of searching for minimal proofs.

In this Appendix, we briefly present this implementation. In Section A.1, we give the instructions on how to build and run the program. In Section A.2, we describe the format of the input files. Some constraints were necessary to be employed in the implementation of the search for minimal proofs. They are presented in Section A.3. Finally, in Section A.4, we present examples from the case studies being executed by our implementation.

A.1 Building the project and running the program

This implementation was thought of to be executed in a UNIX-based operational system. It was tested in a machine running a Ubuntu 24.04.

To build the project, it is required to have installed `g++` with support to `C++17`, `make` and `git`. The commands to run the program are listed below:

1. Clone the `git` repository:

```
git clone https://github.com/joaomendesln/GePECC.git
```

2. Access the source folder:

```
cd GePECC/src
```

3. Build the project:

```
make
```

4. Copy the content of an example to the language symbols file:

```
cp path_to_symbols_file inputs/symbols
```

5. Run the program:

```
./main path_to_expansion_rules path_to_signed_fmlas
```

A.2 Input files format

Three input files are required for the procedure to work:

1. Language symbols (`symbols`): Function and predicate symbols of the signature utilized in the working language and their respective arities.
2. Expansion rules (`expansion_rules`): Set of theory-specific rules,
3. Signed formulas (`signed_fmlas`): Set of signed formulas describing the input exercise.

The language symbols file, placed at `src/inputs/symbols`, is not provided as an argument of the program so it must be manually filled. One can find examples of these three inputs files at examples folders placed at `src/inputs/`. The examples in the folders `inputs/sets` and `inputs/sets_ascii` are the same, but, in the former, they are written with unicode characters and, in the latter, with ascii characters. The same holds for the folders `inputs/numbers` and `inputs/numbers_ascii`.

The format of the input files are described in the next subsections.

A.2.1 Language symbols

The language symbols are specified in the file via lines with the format:

$$\langle arity \rangle \text{`:'} \langle list\_of\_symbols \rangle$$

where $\langle arity \rangle$ is a numeral representing an arity and $\langle list\_of\_symbols \rangle$ is a list of symbols of that arity splitted by commas. Backslashes ("\") are not accepted as symbols.

The file, then, is divided into three sections: function symbols, predicate symbols and skolemization symbols. Before each section, there is a line indicating its starting point.

Despite not being described as an input of the procedure presented in Subsection 4.2, the language symbols file is important in the parsing of the other two input files. Moreover, binary skolemization function symbols are displayed in prefix notation in the output, different from the other binary function symbols, that are displayed in infix notation. That is why the user needs to make explicit what are the skolemization function symbols.

A.2.2 Expansion rules

The rules are specified in the file via lines with the format:

$$\langle premises \rangle \text{`;'} \langle conclusions \rangle$$

where both $\langle premises \rangle$ and $\langle conclusions \rangle$ are lists of signed formulas splitted by commas. Between these lines, we can add comment lines by starting them with an opening square bracket, '[]'. Moreover $\langle conclusions \rangle$ is an empty list when the line is representing a closure rule.

A signed formula is specified as:

$$\text{`('} \langle \circ \rangle \text{`,'} \langle formula \rangle \text{`)'}$$

where $\langle \circ \rangle$ is either '+' or '-', and $\langle formula \rangle$ is an atomic formula written in prefix notation.

A.2.3 Signed formulas

The signed formulas file is constituted by lines containing exactly one signed formula with the format described in A.2.2.

A.3 Constraints on the implementation of the search for minimal proofs

After implementing of the search for minimal proofs following the method described in Subsection 4.2.2, we have realized the necessity of adopting some constraints to improve the performance of the prototype. In this section, we describe such constraints.

A.3.1 Empirical constraints

The restrictions we describe in the sequel were imposed after testing the implementation with examples from our case studies, the theory of sets and theory of numbers. The first restriction is more related to theory of sets, while the other two to the theory of numbers.

1. Cut applications: In the theory of sets, the cut rule poses a risk to increase the width of a successor tableaux tree. Consider the rule $+\cup\mathbf{E}_1$ of the Example 4.16. To apply it in a tableau, it is necessary to have two signed formulas in the format $-\, x \in y$ and $+\, x \in y \cup z$. However, to apply the cut, it is only necessary to have one signed formula following the format $+\, x \in y \cup z$. Because of this, a tableau can only have one cut application in our implementation. One of the examples we provide, the set of signed formulas $\{+\, p_1 \in p_2 \cup (p_2 \cap (p_3 \cap p_4)), -\, p_1 \in ((p_2 \cup p_3) \cap (p_2 \cup p_4)) \cap (p_2 \cup p_5)\}$, described in the file `src/inputs/sets/signed_fmlas/example12`, is provable using two application of cuts, but our implementation does not find a proof for it.

2. Tableaux size: The bigger the tableau, the more verifications must be done to apply a rule on it. Because of this, the tableaux in the tree of successor tableaux can have at most 120 nodes.

3. Height of the formulas: By the height of a formula φ, we refer to the height of the tree that abstractly represents φ. Let n be the maximum height of a formula in the initial tableau for a set of signed formulas. During the search for minimal proofs, the upper bound of the height of the formulas in the tableaux is $n+5$. Such restriction is due to the rules $+{+_{cong}}$ and $+{\cdot_{cong}}$ in Example 4.17, whose conclusions have greater heights than their premises.

A.3.2 Extraction of successor tableaux

In our implementation, the application of rules in the extraction of successor tableaux follows a predefined order. We start by postponing the application of the cut rule and the rules with more than one premise. After applying all one-premise rules, we extract the successor tableaux obtained by the application of rules with more than one premise. If no rules with premises can be applied in a tableau, then we resort to the cut rule. The reason for postponing rules with more than one premise is reducing the amount of deductively isomorphic solutions obtained in the search for minimal proofs. The postponement of the application of the cut rule has the same motivation as the empirical restriction of two cut applications we comment in Subsection A.3.1.

In Algorithm 6, we present the pseudocode of the implementation for the extraction of successor tableaux given a tableau T and set of theory-specific rules TS_Th as inputs. The auxiliary functions we employ are described in what follows:

- $is\_one\_premise$: Checks if an expansion rule is a one-premise rule.
- try_apply_rule: Given a tableau T and an expansion rule r, it checks if it is possible to apply r in T.
- $apply\_rule$: Given a tableau T and an expansion rule r, it returns the result of applying r in T.
- $apply\_cut$: Given a tableau T and a set of expansion rules, it returns a list of the resulting tableaux of all possible cut rule applications in T.

A.4 Examples from case studies

In this subsection, we present some examples of the execution of our implementation with the two case studies we approach in this work. In Figure 23, we illustrate the output displayed in a terminal when the input set of signed formulas is $\{+\, p_1 \in (p_2 \cap p_3), -\, p_1 \in p_2\}$ and the input set of expansion rules is the one presented in Example 4.16. To run this example, execute the following commands in the `src` folder:

- `cp inputs/sets/symbols inputs/symbols`
- `./main inputs/sets/expansion_rules inputs/sets/signed_fmlas/example1`

Algorithm 6 get_successor_tableaux(T, TS_{Th})

```
successors_amt ← 0
one_premise_rules ← one_premise(TS_Th)
n_premise_rules ← n_premise(TS_Th)
for rule in one_premise_rules do
    if is_one_premise(rule) ==True then
        if try_apply_rule(rule, T) == True then
            successors_amt ← successors_amt + 1
            tbl ← apply_rule(rule, T)
        end if
    end if
end for
if successors_amt > 0 then
    return [tbl]
end if
successor_tbl ← [ ]
if successors_amt == 0 then
    for rule in n_premise_rules do
        if is_one_premise(rule) ==False then
            if try_apply_rule(rule, T) == True then
                successors_amt ← successors_amt + 1
                successor_tbl.add(apply_rule(rule, T))
            end if
        end if
    end for
end if
if successors_amt == 0 then
    cut_tbl ← apply_cut(TS_Th, T)
    for tbl in cut_tbl do
        successor_tbl.add(tbl)
    end for
end if
return successor_tbl
```

For the sake of simplicity, we employ the symbols f and g as the skolemization symbols, instead of skl_2^1 and skl_2^2. The program finds one minimal proof and, from it, two proof-isomorphic sets of signed formulas are found, $\{+\, p_1 \in (p_2 \setminus p_3), -\, p_1 \in p_2\}$ and $\{+\, p_1 \in (p_2 \cap p_3), -\, p_1 \in p_2\}$.

```
===== Pre-processing symbols of the language
>> Function symbols
Constants: ∅
Arity 1: fst, snd
Arity 2: −, f, g, ×, ∩, ∪, △
Skolemization: f, g

>> Predicate symbols
Arity 2: ∈, ⊆, ≍

===== Pre-processing expansion rules file
===== Pre-processing signed formulas file
+ p1 ∈ (p2 ∩ p3)
- p1 ∈ p2

===== Searching for minimal proofs
Amount of proofs: 1
Size of proofs: 4

>> Proof 1
= 0: + p1 ∈ (p2 ∩ p3)
== 1: - p1 ∈ p2
=== 2: + p1 ∈ p2, [0]
==== *: [2, 1]

===== Searching for proof-isomorphic sets of signed formulas
>> Set 1
+ p1 ∈ (p2 - p3)
- p1 ∈ p2

>> Set 2
+ p1 ∈ (p2 ∩ p3)
- p1 ∈ p2
```

Figure 23: Output of the prototype implementation for an example from the definitional theory of sets.

Regarding the other examples from the `sets` folder, we mention two examples that our implementation does not find proofs. The first is the set of signed formulas $\{+\, p_1 \in p_2 \cup p_3, -\, p_1 \in p_2\}$, described in the file `sets/signed_fmlas/example3`, and the second is $\{+\, p_1 \in p_2 \cup (p_2 \cap (p_3 \cap p_4)), -\, p_1 \in ((p_2 \cup p_3) \cap (p_2 \cup p_4)) \cap (p_2 \cup p_5)\}$, described in the file `sets/signed_fmlas/example12`. The first is, indeed, not provable. The second is provable, but it demands two applications of the cut rule, which our implementation is constrained not to do.

To the theory of numbers, we have found several limitations, as detailed in Section 5.1. The only interesting examples of provable sets of signed formulas with the rules from Example 4.17 we have manage to run were $SF_1 := \{+\, p_1 \leq p_2, +\, p_2 \leq p_3, -\, p_1 \leq p_3\}$ and $SF_2 := \{+\, p_1 \mid p_2, +\, p_2 \mid p_3, -\, p_1 \mid p_3\}$. It is worth mentioning that, in this case, the execution of the search for minimal proofs is much slower than in the examples from the theory of sets. To run this example, execute the following commands in the `src` folder:

- `cp inputs/numbers/symbols inputs/symbols`
- `./main inputs/numbers/expansion_rules inputs/numbers/signed_fmlas/example1`

```
===== Pre-processing symbols of the language
>> Function symbols
Arity 2: +, f, g, ·
Skolemization: f, g

>> Predicate symbols
Arity 2: |, ≈, ≤

===== Pre-processing expansion rules file
===== Pre-processing signed formulas file
+ p1 ≤ p2
+ p2 ≤ p3
- p1 ≤ p3

===== Searching for minimal proofs
Amount of proofs: 1
Size of proofs: 13

>> Proof 1
= 0: + p1 ≤ p2
== 1: + p2 ≤ p3
=== 2: - p1 ≤ p3
==== 3: + p3 ≈ (f(p2, p3) + p2), [1]
===== 4: + p2 ≈ (f(p1, p2) + p1), [0]
====== 5: + p2 ≈ (p1 + f(p1, p2)), [4]
======= 6: + (p2 + p3) ≈ ((p1 + f(p1, p2)) + (f(p2, p3) + p2)), [5, 3]
======== 7: + (p2 + p3) ≈ (((p1 + f(p1, p2)) + f(p2, p3)) + p2), [6]
========= 8: + (p2 + p3) ≈ (p2 + ((p1 + f(p1, p2)) + f(p2, p3))), [7]
========== 9: + p3 ≈ ((p1 + f(p1, p2)) + f(p2, p3)), [8]
=========== 10: + p3 ≈ (p1 + (f(p1, p2) + f(p2, p3))), [9]
============ 11: + p3 ≈ ((f(p1, p2) + f(p2, p3)) + p1), [10]
============= *: [11, 2]

===== Searching for proof-isomorphic sets of signed formulas
>> Set 1
+ p1 | p2
+ p2 | p3
- p1 | p3

>> Set 2
+ p1 ≤ p2
+ p2 ≤ p3
- p1 ≤ p3
```

Figure 24: Output of the prototype implementation for an example from the definitional theory of numbers.