Hierarchical Contextual Reasoning

Serge Autexier

Dissertation zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Saarbrücken, 2003

Dekan	Prof. Dr. Philipp Slusallek
Vorsitzender	Prof. Dr. Reinhard Wilhelm
Gutachter	Prof. Dr. (PhD) Jörg Siekmann, Universität des Saarlandes
	Prof. Dr. (PhD) Frank Pfenning, Carnegie Mellon University, Pittsburgh, USA
	Prof. Dr. Gert Smolka, Universität des Saarlandes
Kolloquium	19. Dezember 2003

Contents

Kurzzusammenfassung

Das computergestützte Beweisen von Theoremen erfordert den Eingriff des menschlichen Benutzers selbst für nach menschlichen Maßstäben einfache Theoreme. Diese Arbeit definiert eine Kommunikationsplattform, die eine synergetische Kooperation des Benutzers mit den Beweisverfahren ermöglicht. Auf der Grundlage einer neuen Beweistheorie für kontextbasiertes Beweisen werden fast alle Aspekte der Kommunikation abgedeckt, angefangen bei der Präsentation des Beweiszustandes über die Bereitstellung kontextabhängiger Informationen zur Fortführung des Beweises, bis hin zur Unterstützung einer hierarchischen Beweisentwicklung. Die für eine ganze Klasse von Logiken einheitliche Beweistheorie beruht auf beweistheoretischen Annotationen in Formeln. Sie unterstützt eine kontextabhängige Beweisführung, die möglichst intuitiv für den Benutzer und gleichzeitig noch adäquat für automatische Beweisverfahren ist. Darüberhinaus werden Konzepte zur Unterstützung hierarchischer Beweismethodiken entwickelt.

Abstract

Computer supported development of proofs requires user interaction even for theorems that are simple by human standards. In this thesis we define a communication infrastructure as a mediator between the user and the automatic reasoning procedures. It is based on a new uniform meta proof theory for contextual reasoning and encompasses most aspects of communication from the presentation of the proof state, via the supply of relevant contextual information about possible proof continuations, to the support for a hierarchical proof development. The proof theory is uniform for a variety of logics. It exploits proof theoretic annotations in formulas for a contextual reasoning style that is as far as possible intuitive for the user while at the same time still adequate for automatic reasoning procedures. Furthermore, concepts are defined to accomodate both the use and the explicit representation of hierarchies that are inherent in problem solving in general.

Zusammenfassung

Das computergestützte Beweisen von Theoremen erfordert den gezielten Eingriff des menschlichen Benutzers selbst für nach menschlichen Maßstäben einfache Theoreme. Diese Arbeit definiert eine Kommunikationsplattform, die eine synergetische Kooperation des Benutzers mit den Beweisverfahren ermöglicht. Die anhand eines idealen Beweissystems erstellte Anforderungsspezifikation ordnet allgemeine Beweisschritte in zwei informelle Kategorien ein. Die erste Kategorie beinhaltet die Beweischritte, die einen Beweis innerhalb einer Abstraktionsebene fortführen und unterscheidet weiterhin zwischen nachweisbar korrekten Beweisschritten und spekulativen Beweisschritten. Die zweite Kategorie umfaßt dahingegen hierarchische Beweisschritte. Hierarchische Strukturen sind allgegenwärtig in jeder Art von Beweismethodik und resultieren aus der Struktur der Formeln, der Verwendung von Abstraktionen sowie der Strukturierung der Beweisverfahren untereinander. Die Anforderungsspezifikation und die Kategorien motivieren die Entwicklung der Kommunikationsplattform CoRE auf der Grundlage eines neuen Beweiskalküls für kontextbasiertes Beweisen. Die Kommunikationsplattform umfaßt dabei fast alle Aspekte der Kommunikation, angefangen mit der Präsentation des Beweiszustandes über die Bereitstellung relevanter und kontextabhängiger Informationen für mögliche Fortführungen des Beweises, bis hin zur Unterstützung einer hierarchischen Beweisentwicklung.

Im ersten Teil der Arbeit wird die Beweistheorie definiert. Die Hauptanforderung dabei war, dass der Zustand eines Beweises in einer Form repräsentiert wird, die sowohl adäquat für automatische Beweisverfahren ist als auch intuitiv für einen menschlichen Benutzer. Dies erfordert eine direkte Unterstützung für kontextbasiertes Beweisen auf beliebigen Teilformeln sowie zum Beispiel die unmittelbare Entwicklung eines Beweises auf der Fakten-Ebene, welche als adäquate Beweisrepräsentation zur Präsentation von Beweisen in natürlicher Sprache vorgeschlagen wurde. Die Beweistheorie beruht auf zwei Säulen: Im ersten Teil wird ein korrekter und vollständiger Matrix-Beweiskalkül für indexierte Formelbäume einheitlich für eine Klasse von Logiken definiert. Indexierte Formelbäume sowie die verwandten Expansionsbäume für Logik höherer Stufe repräsentieren Quantoren-Abhängigkeiten auf einheitliche und effiziente Art und Weise. Die indexierten Formelbäume werden durch Zerlegung der zu beweisende Formeln entlang ihrer Struktur gewonnen, wobei die einzelnen Teile der Formel mit Polaritäten und einheitlichen Typen annotiert werden. Jede frei instantiierbare quantifizierte Teilformel wird dabei in eine beliebige aber feste Anzahl von Instanzen zerlegt, welche als Multiplizitäten der Teilformeln bezeichnet werden. Zur Vermeidung dieser statischen Festlegung der Multiplizitäten wird der Beweiskalkül um eine effiziente Beweisregel erweitert, die es erlaubt zu jedem Zeitpunkt im Beweis die Multiplizitäten dynamisch anzupassen. Der resultierende Beweiskalkül ist korrekt und vollständig und bildet das Rückgrad der Beweistheorie für die Überprüfung der Zulässigkeit von Substitutionen. Der zweite Teil der Beweistheorie erweitert den Matrix-Beweiskalkül um intuitive und kontextbasierte Beweisregeln. Dabei wird der Matrix-Kalkül ergänzt um eine Formel mit freien Variablen, die aus dem indexierten Formelbaum resultiert. Die Formel wird vollständig mit Polaritäten und einheitlichen Typen annotiert und diese beweistheoretischen Annotationen bilden die Grundlage für die Definition des logischen Kontextes von Teilformeln sowie der darin enthaltenen Ersetzungsregeln. Die definierten Ersetzungsregeln können sowohl als Realisierung von Beweisschritten auf der Fakten-Ebene angesehen werden, als auch als verallgemeinerte Resolutions- und Paramodulations-

Regeln. Die janusköpfigen Ersetzungsregeln unterstützen somit zum einen die Beweisführung auf der Fakten-Ebene, die ideale Grundlage für Benutzerinteraktion und abstraktes Beweisplanen, und zum anderen die Implementierung automatischer Beweisverfahren, wie zum Beispiel Taktiken und ordnungsbasierte Beweisverfahren. Im entwickelten Beweiskalkül besteht ein Beweiszustand aus einem indexierten Formelbaum und einer Formel mit freien Variablen. Ein Beweis ist abgeschlossen, wenn diese Formel zu der trivialerweise gütligen Formel *True* reduziert wurde. Der Kalkül besteht aus zwölf Beweisregeln, die jede einen Beweiszustand zu genau einem neuen Beweiszustand reduziert. Somit ist gewährleistet, dass der Zustand eines Beweises zu jedem Zeitpunkt in einer einzigen Formel repräsentiert werden kann, eine für den Benutzer intuitive Art der Repräsentation.

Der zweite Teil der Arbeit beschäftigt sich mit den hierarchischen Aspekten einer Beweisführung. Hierarchien in Beweisen entstehen auf drei Arten: Durch die Struktur der Formel eines Beweiszustandes, die Verwendung von Abstraktionen, und die Struktur der Beweisverfahren untereinander, wie zum Beispiel in der Beweisplanung oder rekursiv definierten Taktiken. Die Ausnutzung der Formelstruktur resultiert in einer hierarchischen Beweisführung, die unter dem Begriff "*window inference*" formalisiert wurde. Dabei wird das Fokussieren auf und die logische Transformation von Teilformeln ermöglicht. Aufbauend auf dem Beweiskalkül für kontextbasiertes Beweisen erweitert der CoRe *window* Beweiskalkül das bekannte *window inference* um eine einheitliche Bestimmung von Ersetzungsregeln aus einem logischen Kontext sowie eine einheitliche Vererbung der Fokusstrukturen während der Anwendung einer Beweisregel. Um die Verwendung von Abstraktionen zu unterstützen, werden notwendige Konzepte zur Definition und Verwendung von Abstraktionen definiert. Der Anwendbarkeitstest einer Abstraktion wird mittels *Beweisdomänen* explizit repräsentiert, die die Ausgangs- und Zielbeweisdomäne einer Abstraktion approximieren. Zur Repräsentation von Beweisen auf unterschiedlichen Hierarchie-Ebenen wird eine hierarchische Beweisrepräsentation für CoRe definiert, die CoRe *window* Beweisregeln, die Verwendung von Abstraktionen und Hierarchien in den Beweisverfahren einheitlich repräsentiert.

Der dritte und letzte Teil der Arbeit beschäftigt sich mit der Anwendung der Kommunikationsplattform. Die Implementierung von automatischen Beweisverfahren für den CoRe Kalkül wird erschwert durch die Vielzahl möglicher Ersetzungsregeln für die Teilformeln – ein ansonsten klarer Hinweis für das durch den Kalkül bereitgestellte hohe Maß an Flexibilität in der interaktiven Beweisführung. Dieser Problematik wird mit dem Konzept von Filtern begegnet, die eine zielgerichtete Auswahl geeigneter Ersetzungsregeln ermöglichen. Vor der Präsentation von Beispielbeweisen in CoRe, wird vor allem die Implementierung eines Kalküls ähnlich dem Sequenzenkalkül auf der Grundlage des CoRe window Kalküls dargestellt. Neben den CoRe window Kalkülregeln wird dabei eine Regel zur internen Zerlegung von Formeln verwendet, die eine Verallgemeinerung einer entsprechenden Regel aus Schütte's Beweistheorie ist. Die definierten Kalküle zeichnen sich dadurch aus, dass sie besonders für die interaktive und automatische Beweissuche geeignet sind. Neben anderen Vorteilen sind vor allem effiziente Beweistransformationen zu nennen, die aus der dynamischen Anpassung von Multiplizitäten resultieren, sowie die Technik des *Theorembeweisen Modulo*, die durch die kontextabhängige Auswahl und Anwendung von Ersetzungsregeln subsumiert wird.

Extended Abstract

Computer supported development of proofs requires user interaction even for theorems that are simple by human standards. In order to enable a synergetic cooperation of the user with the automatic reasoning procedures this thesis defines a communication infrastructure to mediate between the user and the reasoning procedures. A requirement analysis conducted along the sketch of an ideal proof development environment results in a classification of proof construction steps into two informal categories. The first category consists of the continuation of proofs within one level of abstraction and is further refined into verifiably sound proof continuations and speculative proof continuations. The second category of proof steps are those that introduce vertical hierarchies in proofs. The use of hierarchies is inherent in problem solving in general and due to the structure of the formula, the use of representational abstractions, as well as the structuring of reasoning procedures. The requirement analysis and the resulting classification motivate the definition of the communication infrastructure CORE, which is based on a new uniform meta proof theory for contextual reasoning and encompasses most aspects of the communication that range from the presentation of the proof state, via the supply of relevant contextual information about possible proof continuations, to the support for a hierarchical proof development.

The first major part of the thesis is concerned with the definition of the proof theory. Its development is conducted under the requirement that it shall allow for a representation of the proof state that is both intuitive for the user and adequate for automatic reasoning procedures. Furthermore it shall support reasoning inside formulas by exploiting the logical context of subformulas. Finally, it shall equally support the integration of automatic reasoning procedures as well as allow for an intuitive reasoning style for the user, such as directly reasoning on the assertion level, which has been proposed as an adequate representation to support natural language presentations of proofs. The proof theory rests on two pillars: in a first part a sound and complete matrix calculus on indexed formula trees is defined uniformly for a variety of logics. Indexed formula trees as well as their higher-order logic pendant expansion tree proofs provide a uniform and concise representation of the dependencies between variable and modal quantifiers. The formula is decomposed along its tree structure and annotated by polarities and uniform types. Thereby the instantiable variable and modal quantifiers are assigned an arbitrary but fixed number of instances, so-called multiplicities. In order to overcome the limitation to assign multiplicities beforehand, the calculus is extended by a rule to dynamically and efficiently increase the multiplicities of quantifiers at any stage of the proof development. The resulting sound and complete calculus serves as the soundness backbone of the proof theory with respect to checking the admissibility of substitutions.

The second part of the proof theory adds intuitive and contextual reasoning capabilities to the matrix calculus. Intuitively, it extends it by a free variable representation of the complete formula contained in the indexed formula tree. The entire free variable formula is annotated with polarities and uniform types and this proof theoretic information is the basis for a uniform notion of logical context for subformulas as well as replacement rules contained in logical contexts. The replacement rules are Janus-faced by nature: on the one hand they can be viewed as the operationalisation of assertion level proof steps. They therefore enable the development of proofs directly at the assertion level and are the ideal basis for user interaction and high-level proof planning. On the other hand

from a logical point of view they are generalised resolution and paramodulation rules, which is a suitable representation for automatic reasoning procedures such as tactics or ordering based reasoning procedures. A proof state in the developed calculus consists of an indexed formula tree to represent quantifier dependencies and a free variable formula that represents the status of the proof. The proof is completed when that formula has been reduced to the trivially valid formula *True*. The complete calculus consists of twelve calculus rules, each reducing a proof state to exactly one new proof state. This ensures that the whole status of the proof is always contained inside a single formula, which is an intuitive representation for the user as it overcomes the use of normalforms and skolemization.

The second major part of the thesis is concerned with the hierarchical aspects of problem solving. We identify three sources that lead to hierarchies in proofs: the structure of the formula representing the proof state, the use of representational abstractions during proof development, and the hierarchical structure induced by the involved reasoning procedures, such as proof planning or recursive definitions of tactics. Exploiting the structure of formulas gives rise to a hierarchical reasoning style that has been formalised by window inference. Window inference supports the focusing on arbitrary parts of the formula and supports their manipulation. Due to the contextual reasoning capabilities of the underlying calculus, the CoRe window calculus extends standard window inference by a uniform determination of replacement rules from a logical context and a uniform principle to inherit window structures during rule application. The use of representational abstractions is addressed by the formalisation of the basic concepts that support their definition as well as their use at any stage of a proof development. The applicability conditions of abstraction functions are captured by the notion of reasoning domains, which approximate the source and target domains of representational abstractions. Finally, we define a hierarchical proof datastructure that accomodates the application of CoRe window calculus rules, the use of representational abstractions, and the representation of derivational abstractions and refinements.

The third and final major part of the thesis presents applications of the developed communication infrastructure. The definition of automated reasoning procedures on top of CoRe is hampered by the exponential amount of possible replacement rules for each subformula – a clear indication of the high flexibility provided by CoRe during proof development. To remedy this problem, we introduce filters to be used for a goal directed selection of appropriate replacement rules. Before the presentation of some example proofs in the CoRe window calculus, we present an implementation of a calculus that resembles a sequent calculus using CoRe. It is based on the CoRe window calculus rules and an admissible subformula decomposition rule which generalises a respective rule from Schütte's sentential calculus. Due to the capabilities of the underlying CoRe calculus, the adequacy of the resulting sequent style calculus to support interactive and automated proof search surpasses standard implementations of this calculus. Beyond other benefits, the most striking advantages are complex proof transformations that result from the dynamic and efficient increase of multiplicities and the subsumption of theorem proving modulo by the support of contextual reasoning inside subformulas.

Acknowledgements

First of all I would like to thank my supervisor Jörg Siekmann for all his support during these years. His enthusiasm paired with critical advise were very valuable for my research and for writing this thesis. Moreover, I would like to thank him for letting me work in this huge group of high quality researchers in Saarbrücken which was so beneficial to me and my work.

Second, I would like to thank Frank Pfenning and Gert Smolka for serving on my thesis committee. In particular, I would like to thank Frank Pfenning for many interesting discussions and his enlightening comments on earlier versions of this work.

Third, my special thanks goes to Dieter Hutter for his critical advise and discreet guidance over many years, his openness to discuss new ideas and for always encouraging me. Moreover, I would like to thank Christoph Benzmüller and Chad Brown for stimulating discussions about the higher-order logic aspects of the work presented in this thesis. Thanks also to Claus-Peter Wirth for stimulating discussions, whose enthusiasm also helped me throughout the difficult stages of this thesis. For proof reading (parts of) this thesis I am indebted to Christoph Benzmüller, Alexander Caviedes, Dieter Hutter, and Andreas Nonnengart.

Aside from my thesis a lot of effort during the last years went into research only sometimes related to this thesis and I would like to thank my co-authors for these productive cooperations: Christoph Benzmüller, Armin Fiedler, Helmut Horacek, Malte Hübner, Dieter Hutter, Bruno Langenstein, Heiko Mantel, Andreas Meier, Till Mossakowski, Georg Rock, Axel Schairer, Carsten Schürmann, Werner Stephan, Quoc Bao Vo, Roland Vogt, and Andreas Wolpers.

For many discussions, valuable feedback and pleasant coffee breaks I would like to thank all of my colleagues at the formal methods and Activemath groups at DFKI and the ΩMEGA group at Saarland University. In particular, I thank Armin Fiedler, Heiko Mantel, Andreas Meier, Martin Pollet, Axel Schairer, and Werner Stephan for stimulating discussions. Thanks to Chris for having been such a pleasant office mate. My work also profited a lot from research visits at the University of Edinburgh, Carnegie Mellon University, and Yale University. Thanks to Alan Bundy, Frank Pfenning, and Carsten Schürmann for providing me with these opportunities.

Last but not least, I would like to thank all of my friends and especially my family for their unconditional loyality and moral support. Above all, I thank Sandra Veeck for her moral and emotional support during these years and for being a partner in the rough and in the good times.

Il nous faut peu de mots pour exprimer l'essentiel;
il nous faut tous les mots pour le rendre réel.
Eluard Eugène Grindel, dit Paul

Introduction

–

Part I

Chapter 1

Introduction

Ce qui est le meilleur dans le nouveau est ce qui répond à un désir ancien.
Paul Valéry

Inspite of almost four decades of research on automated theorem proving, mainly theorems considered easy by human standards can be proved fully automatically without human assistance. For instance humans must provide guidance information about how to explore the search space or specify intermediate lemmata. The computer-supported development of proofs for more difficult theorems requires user interaction and will require it for the foreseeable future. The main application domains of computer-based theorem proving systems are mathematical assistants, mathematical teaching assistants, and hardware as well as software verification. For these domains, a tight intertwining of user interaction with the theorem proving system is necessary, to obtain proofs in formal logic – even for theorems which are relatively simple from a human point of view. In this thesis we propose a communication infrastructure as mediator between the user and the theorem proving system that provides the basis for a tight integration of the user and the theorem proving system.

1.1 Motivation

1.1.1 Communication of Proof Knowledge

In an *ideal proof development environment* the user and the theorem proving system can cooperate in a synergetic manner. Such a theorem proving system consists of one or more reasoning engines, where "reasoning engine" is a general term that encompasses any kind of automated proof search procedure by abstracting from the implementation paradigm and denotes specialised automatic theorem provers, tactics, proof-planners, or computer algebra systems and constraint solving systems. The objective is to develop formal proofs for theorems where the partners are the user and the theorem proving system with its reasoning engines. We envision the development of proofs as a synergetic cooperation of equal partners, in contrast to the current situation of a master/slave relationship between the user and the theorem proving system. Each partner contributes (implicit) knowledge on how to approach and solve proof obligations. It is the partner's decision *when* to bring in its specific knowledge about proofs and *how* to convey the knowledge. This requires that the status of the proof and the means of knowledge communication are understandable to all partners, even if the user is a mathematician, scholar or software engineer with no expertise in formal logic.

In order to efficiently support the user, communication between the user and the theorem proving system is crucial. In principle each partner can communicate the proof knowledge he incorporates. However, the specific proof knowledge must be expressed in a manner that is intelligible to the other partners. This is the bottleneck for the communication of information. A user like a mathematician or software engineer usually has a semantic representation of the problem domain and exploits it to approach and solve proof obligations. They usually have little or no knowledge about formal logic. State of the art automated theorem provers, however, only incorporate deep knowledge about the search space structure based on the syntax and calculus rules. In interactive theorem provers tactics are used to incorporate more high-level proof procedures, but these still stick to the syntax and the basic calculus rules. Proof planning has been designed to overcome these limitations. However, in practice it also requires an understanding of the underlying calculus from the user and does not completely overcome the limitations imposed by the lack of abstraction imposed by the underlying calculus.

Coming back to the ideal proof development environment, the proof knowledge of the partners includes both the definition of new proof techniques and how and when to use the available proof techniques within an actual proof. Examples of basic proof techniques are calculus rules, computations to simplify or adapt sentences, small mathematical computations of values or symbolic evaluation of some formula. Examples for higher proof techniques are abstractions, such as diagrams for conjectures about natural numbers, the diagonalisation method, proofs by induction, a pigeonhole argument and similar mathematical techniques. Thus, the state of an ideal proof development environment consists of both the available proof techniques and an actual (partial) proof.

To communicate this knowledge, we propose to use a communication infrastructure that mediates between the user and the theorem proving system. Its role is on the one hand to represent all the information about the proof and especially to *intelligibly* communicate this information to both the user and the reasoning engines. On the other hand it must support the continuation of the proof by the user and the reasoning engines, which requires an interaction interface that is, again, *intelligible* for both the user and the reasoning engines. This tension between intelligible information for the user on the one hand *and* the reasoning engines on the other strongly influences the design of the communication infrastructure. Resolving this inside the communication infrastructure that is used for proof construction by both the user and the reasoning engines could lead to a synergetic cooperation of the partners involved.

The communication infrastructure must provide information about the *history of the proof*, i.e. what has been done so far, about the *status of the proof*, and about the *possible next steps* to continue the proof. The manner in which the proof history is represented depends on the kinds of possible proof steps, which are discussed in the following Section 1.1.2. In Section 1.1.3 we discuss what kind of information about the status of the proof is required for the ideal proof development environment.

1.1.2 Proof Construction Steps and Proof History

The ideal proof development environment requires a clarification of what a proof is in this context. In formal logic a proof is a sequence or a proof tree built from the calculus rules of the underlying logic. In proof planning it is a proof plan, i.e. a sequence of methods, that can be refined to a calculus proof. In mathematical textbooks a proof can be any natural language justification indicating how a sentence is inferred from some other sentences. In order to conduct a requirement analysis we need a notion of proof that encompasses all these different types of proofs. In the following we introduce different informal categories of proof construction steps to refine the notion of a proof for the ideal proof development environment.

The proof construction steps are the major means to communicate proof knowledge between the partners. Any kind of communication of proof knowledge adds information to the proof. We view a proof as a three dimensional object that consists of *horizontal* and *vertical* structures. A horizontal structure is a (partial) proof on a specific *level* of abstraction, while a vertical structure links proofs at different levels of abstraction. Each proof within some level consists of a sequence of justifications relating a goal sentence to subgoals. Proof steps that extend a proof within a level of abstraction are denoted by *intra-level proof construction steps*. These proof steps are *horizontal* and each proof step relates sentences and is annotated by some justification, like the name of the calculus rule used, the name of a tactic, a lemma being applied, or simply the description of the proof technique used. Among those steps, we need to differentiate between those steps that *soundly* extend a proof, like some verifiably valid derivation, and those that are *speculative*, like the formulation of proof intentions as for example in proof planning. We denote the former by *(local) lemma application* steps and the latter by *(local) lemma speculation* steps.

Each subproof within some level of abstraction can be related to a subproof at a different level of abstraction. An example for this is to abbreviate a sequence of proof-steps by a single proof-step describing that sequence of proof steps. Another example is the application of a lemma of the form $H_1 \ldots H_n \Rightarrow H$, where this "macro-step" can be expanded by its proof. Similarly, the validation of a speculative proof step by a sequence of concrete proof steps can be see as a refinement, as it also introduces a lower level of abstraction into the proof. We denote those *inter-level proof construction steps* by *vertical* proof steps. The activity of relating a subproof at some level of abstraction to some subproof at a higher level of abstraction is called *vertical abstraction*, and the dual activity is *vertical refinement*.

1.1.2.1 Intra-Level Proof Steps

Local Lemma Application. The local lemma application proof steps are those that reduce an open goal to a possible empty list of new subgoals. The communication infrastructure has to provide the user and the reasoning engines with information about possible continuations and to subsequently ease their application. The information about the possible continuations must be intelligible to both the user and the reasoning engines. The information must be provided for the user in a way that allows for a semantical interpretation of the information. For example, the user may be told that the group-property of the set G with operation $\odot$ occurring in the actual goal can be expanded. This information must be provided to the reasoning engine in a way that allows for an operational reading, for example as an inference rule that reduces the conclusion to certain premises. More specifically, it should tell the reasoning engines the rule or the name of the rule that reduces the actual goal that contains "G *with* $\odot$ *is a group*" into a subgoal that contains the expanded group-definition.

Furthermore, this information should be derived from the context of the actual goal, such that if the logical context is changed then this is immediately reflected in the set of choices presented to the user and reasoning engines.

This requires from the communication infrastructure that it incorporates a mechanism for *contextual reasoning*, which supports a goal-dependent presentation and application of the information contained in the logical context. Ideally the context should be represented in a uniform manner and the way this information is provided to the user and the reasoning engines are only different views on the *same* information.

Local Lemma Speculation. The local lemma speculation proof steps must allow for speculative or unverified proof steps. For example, the step may be that the user or the reasoning engine can

reduce a goal to some subgoals of their choice, without having to prove that step immediately. This kind of steps offers a means to express intentions on how the proof could or should proceed and to communicate these intentions to the other partners. Furthermore, local lemma speculation proof steps serve to integrate mathematical computations into the proof tree, without having to provide an actual proof that establishes the soundness of this computation. The communication infrastructure must support these unverified proof steps and it must keep track of them until later when they are actually verified.

1.1.2.2 Inter-Level Proof Steps

For an ideal proof development environment we identify two kinds of vertical abstraction and vertical refinement: those that operate on the proof structure and those that operate on the representation of objects. The former are denoted by *proof abstractions* respectively *proof refinements*, and the latter as *representational abstractions* respectively *representational refinements*.

Vertical Abstractions. *Proof abstractions* abbreviate sequences of proof steps into a single proof step. The new proof step is a local lemma speculation proof step at some higher level of abstraction. Having these steps explicitly represented eases the communication as it shortens the proof to its essential steps. Take as an example a proof procedure performing an induction proof. Abbreviating the induction proof into a single proof step with an explicit description of the abbreviated proof sequence captures the implicit knowledge contained in the proof procedure. Furthermore, the abstract and explicit description is certainly more comprehensible than the whole proof sequence. For instance, the *assertion level* introduced by Huang [Huang, 1996] is an abstract proof representation which presents a proof as a sequence of axiom, lemma and hypothesis applications and is the basis for the natural language presentation of formal proofs [Huang, 1996, Fiedler, 2001].

Representational abstraction is an important feature not only in mathematical problem solving, but in problem solving in general. Examples are the use of diagrams to reason about sums and products of natural numbers (cf. [Jamnik *et al*, 1997]), or the labelled fragment abstraction [Hutter, 1994] used in inductive theorem proving which abstracts from the specific differences between the induction conclusion and the induction hypothesis. Those changes of the representation into more adequate representations are often crucial proof steps and their explicit representation provides important information about the proof. They must be annotated with specific information about the type of the representational change. A necessary prerequisite for representational abstractions is that the ideal proof development environment supports the definition of various representation languages, and especially, that it supports multiple proof states with respect to different representation languages and has the means to switch between the representation at any stage of the proof search process.

Vertical Refinements. *Proof refinements* validate speculative proofs. These can be local lemma speculation proof steps or proofs performed on a different representation, initiated by a representational abstraction proof step. They explicitly represent how a formulated proof intention has been actually realised by relating the proof on some higher level of abstraction to its actual proof. Take as an example a diagonalisation argument used to abstractly finish a proof. The diagonalisation argument is represented by a local lemma speculation proof step. However, the proof is not formally closed by this argument, and only the actual execution of the intended diagonalisation argument completes the proof. The resulting proof sequence is a validation of the diagonalisation argument, of which it is a vertical refinement.

Representational refinement is the inverse of representational abstraction. It is required in order to map back from an abstract representation to the initial source representation of the problem. However, although representational abstraction is usually computable, representational refinement usually is not. The reason is that representational abstraction abstracts details in order to obtain a simpler representation of the problem, which is more concise with respect to the problem domain. Thus, representational abstraction is usually a "many-to-one" relationship, which hampers its inversion, i.e. representational refinement. Representational refinement is usually achieved in combination with a vertical refinement of the proof at the abstract representation. Indicating which abstract proof state corresponds to which proof state in the source representation provides important information about the proof, and representational refinement relations serve to represent this information explicitly.

1.1.3 Status of Proofs

The status of the proof is an important information to be provided to the partners. Traditionally in interactive theorem proving it is a proof tree where the leaves are either goals closed by an axiom rule or open goals that must still be solved. The open goals are conjunctively related by the proof tree structure and may share common variables whose instantiation affects several open goals, which usually causes problems for the design of automated proof procedures, since the instantiation of a global variable in one conjunct affects the proof of the other conjunct. Although this representation of the proof status is concise, we argue that it lacks important information: first, alternatives are not explicitly represented in the proof structure, except for alternatives inside the open goals themselves. A mathematician has the alternatives to the actual proof goal structure in mind and an ideal proof development environment should support this more adequate, though more redundant, representation of the proof status. Secondly, the list of open goals can grow rapidly even for problems that are simple from a human point of view. A mathematician usually takes a "birds eye view" on large proofs, which gives a better assessment of the overall proof status. This global view of the proof contains both all open goals and its alternatives. We argue that a good approximation of this birds eye view is to represent the proof status as a single formula at any stage of the proof search process. The compact representation of the proof status as a single formula is a useful information complementary to the usual lists of open goals. Standard formal calculi do not support this compact representation of a proof status, since they usually rely on normal forms or formula decomposition, which both hamper the reconstruction of the proof status as a single formula which resembles the original conjecture. An ideal proof development environment should support the view of any proof status as a list of open goals and also provide information about discarded or ignored alternatives, and finally allow the partners to have a global view of the proof state.

1.2 The CoRe System

The main contribution of this thesis is the engineering of the CoRe system and the development of its logical foundations. CoRe tries to bridge the gap between the intuitive development of proofs and supports a synergetic cooperation between reasoning engines and the user. The key idea of the CoRe-system is that *the whole proof state is always a formula*. Proof construction proceeds by using information contained in the formula to successively transform (parts of) the formula until the proof state is a trivially valid formula. Possible case splits are also represented in the formula by means of logical connectives. Thus, the proof structure is represented in the formula which allows us to view the proof state on the one hand as structured proof representation with open goals and on the other

hand as a single formula which is the birds eye view. Take as an example the following formula about sums of natural numbers:

$$(\forall n \,.\, n = 0 \Rightarrow \sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2) \wedge (\forall n, m \,.\, (n > m \wedge \sum_{i=1}^{m} i^3 = (\sum_{i=1}^{m} i)^2) \Rightarrow \sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2)$$

The formula as whole represents the whole proof state. However, by fixing the parts we want to consider as open goals, for instance the occurrences of $\sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2$, the structure of the formula above these occurrences allows to present the proof state as the following structured proof:

Case 1: *Assume* $n = 0$, *prove* $\sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2$.
Case 2: *Assume* $n > m$ *and* $\sum_{i=1}^{m} i^3 = (\sum_{i=1}^{m} i)^2$, *prove* $\sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2$.

Logical contexts are treated as first-class citizens and can be statically determined for any part of the formula. They are provided as *replacement rules*, which are a formalisation of the notion of assertion level rules. The assertion level has been introduced Xiarong Huang in [Huang, 1996] as an abstraction from the pure natural deduction calculus and it is the basis for the generation of the proof presentation in natural language [Huang, 1996, Horacek, 1999, Fiedler, 2001]. The idea is to subsume axioms, definitions, lemmas, and theorems as *assertions*, and the use of a single assertion in the proof search corresponds to a whole proof segment in the underlying calculus. Consider the example assertion taken from [Huang, 1996]:

$$\forall S_1, S_2 : \mathit{Set} \,.\, S_1 \subseteq S_2 \Leftrightarrow \forall x : \mathit{Element} \,.\, x \in S_1 \Rightarrow x \in S_2$$

This assertion allows us to derive

- $a \in S_2'$ from $a \in S_1'$ and $S_1' \subseteq S_2'$;
- $S_1' \not\subseteq S_2'$ from $a \in S_1'$ and $a \notin S_2'$;
- $\forall x : \mathit{Element} \,.\, x \in S_1' \Rightarrow x \in S_2'$ from $S_1' \subseteq S_2'$.

Xiarong Huang states in [Huang, 1996] "although introspection seems impossible to reveal the internal structure of the interpreter applying assertions, every application of an assertion can be associated with a proof segment [...]", and a procedure to generate the associated proof segments is given in Chapter 6 of [Huang, 1996].

Replacement rules are a generalisation of assertion level rules that capture concisely the internal structure of the application of an assertion. They provide the "necessary introspection" and overcome the need to verify assertion applications by constructing an associated proof segment. Thus, replacement rules allow for the direct and verified use of assertions already in the proof search. Furthermore, we define a uniform characterisation of replacement rules that are in the logical context of some subformula.

During the proof search, replacement rules are used to manipulate parts of the formula provided they are in the logical context of this part. They transform the proof-state formula into a new formula which represents the new proof state. In this thesis we develop the theoretical foundations for such a calculus, which are based on the notion of an *indexed formula tree* and exploit the tree structure of formulas to annotate each node of the tree with some logical information. The logical information is an encoding of the proof-theoretic semantics of a formula at some node, which can be statically determined.

The logical annotations are the basis for the formal definition of the logical context of a subformula and the formal definition of replacement rules. The overall reasoning style supported by the final calculus is based on this information and it enables us to prove uniformly the soundness and completeness of the calculus for a variety of logics.

The kernel proof system supports the intra-level proof steps of Section 1.1.2.1. The main local lemma application proof steps are instantiation, contraction, and replacement rule application. Since the replacement rules are designed to support an intuitive interpretation by the user as well as an operational interpretation by the reasoning engines, they exactly match the requirements for local lemma application. Furthermore, the proof system supports local lemma speculation proof steps by encoding the speculated intermediate goals as logical cut inside the proof state, i.e. the logical cut is used as a means to keep track of pending open goals.

As a further feature to support intuitive reasoning we add a mechanism inspired by the window inference technique [Robinson & Staples, 1993, Staples, 1995] which focuses the reasoning process onto specific subformulas. The subformulas within the focus are then open to manipulation by replacement rules from the logical context of the focus.

To support the inter-level proof steps and especially representational abstraction, CoRE supports the explicit definition of domain specific representation languages, called *reasoning domains*. It supports also the abstraction functions that formalize switching between different reasoning domains. Intuitively a reasoning domain consists of a logic and a set of predefined symbols. The kernel proof system is parametric in the logic, where a logic can be selected from a given set of logics. These logics are built into the system and can be used for the definition of reasoning domains. Additionally, predefined symbols can be defined for a reasoning domain, which give rise to the respective object-level type and constant declarations, i.e. the signature.

A change in the representation of a given conjecture is a vertical abstraction that maps a formula with respect to a source reasoning domain to a formula with respect to a target reasoning domain. They are similar in nature to the local lemma speculation proof steps. The user can select a target reasoning domain and provide a formula with respect to the target reasoning domain which is considered an abstraction of the source formula. Typically the user exploits its semantic understanding of the problem in order to come up with the right abstraction, i.e. the specified target reasoning domain and formula. In order to mimic this problem solving behaviour and make it available to the reasoning engines, abstraction functions that map a source formula from some reasoning domain to a formula in the target domain can be defined in CoRE. They can be used to encode a specific abstraction methodology, which operationalises the behaviour of the user and at the same time are available to the reasoning engines.

The remaining inter-level proof steps require an explicit proof object that represents all the more abstract as well as the more refined proof steps. To this end we introduce a hierarchical proof representation which represents the inter-level proof steps that have been introduced so far. Additionally, we introduce proof steps to represent vertical abstraction and vertical refinement.

1.3 Overview of the Thesis

The thesis is organized into five parts: the first and introductory part ends with Chapter 2 by recapitulating the state of the art. The main contributions of the thesis are presented in Part II and Part III.

Part II presents the proof-theory for contextual reasoning with some preliminary notions and the definition of the proof theory underlying the CoRE reasoning system. Following [Wallen, 1990, Miller, 1983, Pfenning, 1987] we introduce indexed formula trees based on uniform notation and po-

larities for a variety of classical and modal logics, with an emphasis on first- and higher-order logics (Chapter 4). These can be extended to dynamically increase the multiplicities of formulas and provide a sound and complete but unintuitive calculus for the whole class of the logics considered. Intuitiveness is added in Chapter 5 which defines the actual CoRE proof theory. The indexed formula trees of the preceding chapter are extended by working copies, which are denoted by *free variable indexed formula trees*. The basic calculus rules provided by the framework support an intuitive reasoning style that tries to overcome the need for the user to reason in the specific calculus of some special logic in order to prove a theorem (Section 5.3). The soundness of the rules is proved along with the description of the rules, while the completeness with respect to the class of considered logics is proved in Section 5.4, which completes the definition of the kernel reasoning systems.

Part III is devoted to hierarchical reasoning and addresses the requirements sketched out in Section 1.1.2. In Chapter 6 window inference reasoning is added onto the contextual reasoning system in order to support a hierarchical reasoning style. The kernel reasoning system already provides all necessary features to support window inference. The window inference reasoning rules rely entirely on the reasoning rules of the kernel system. In Chapter 7 we present the notions underlying the support for changing representation languages by abstraction during proof search. Finally, the CoRE proof datastructure is defined in Chapter 8

In Part IV we present different applications of CoRE. Chapter 9 defines the interface provided by CoRE for the development of automatic reasoning procedures on top of CoRE. In order to ease the comparison between a standard sequent calculus and the CoRE calculus we define in Chapter 10 how a similar calculus is implemented in the CoRE calculus and examples are presented in Chapter 11.

In Part V, Chapter 12 quotes related work and, finally, in Chapter 13 we summarise the contributions of this work and present an outlook for future research based on the foundations laid in this thesis.

Chapter 2

Historical Overview and State of the Art

The area of research on automated and interactive theorem proving may have been inspired by Leibniz' dream of developing a "*lingua characteristica universalis*" in which every problem should be expressible together with a "*calculus ratiocinator*" to mechanise subsequent logical reasoning. The dream was to provide a basis to solve every logical dispute by encoding it in the "*lingua characteristica universalis*" and then resolving the dispute by calculation: "CALCULEMUS!" – Let us calculate the outcome of the dispute.

Risking oversimplification, the research devoted to the realisation of Leibniz' dream can be divided into three parts: first, the foundational research on mechanised reasoning was and is concerned with the development of *formal* logic, i.e. the achievement to separate syntax and semantics in the definition of logics [Frege, 1879, Tarski, 1936], the definition of various general but also domain specific logics along with investigations of their consistency [Zermelo, 1908, Whitehead & Russell, 1910, Fraenkel, 1922, Von Neumann, 1928, Bernays, 1937, Gödel, 1940, Bernays, 1941], (un-)decidability results [Church, 1936, Turing, 1937], the (non-)existence of complete calculi for these logics [Gödel, 1930, Gödel, 1931] and the development of such calculi [Church, 1940, Gentzen, 1969, Robinson, 1965, Andrews, 1989, Miller, 1983, Bachmair *et al*, 1992]. Despite many throwbacks, Herbrand's [Herbrand, 1930] and Gentzen's work [Gentzen, 1969] and the development of computers smoothed the way for the second part of research concerned with the actual development of mechanised reasoning systems. The first such system was the **logical theorist** [Newell *et al*, 1957]. These first attempts at fully automatic "logic calculators" made it quickly apparent that the inference rules defining a calculus were far from sufficient to build an efficient automatic theorem prover. The central theme of this second part was – and still is – the development of techniques to guide the proof search. The research in that area led to the identification of different paradigms of the representation of guidance information. These proof search paradigms are reflected by the terminology given to the different theorem proving styles, namely: *automatic theorem proving*, *tactical theorem proving*, and *proof planning*.

Finally, the third and most recent part is concerned with the actual application of the developed techniques and systems to mathematics. This emphasis on mathematics is not the least due to Hilbert's idea of formalising mathematics as articulated in Hilbert's program [Hilbert, 1930]. The main objectives with respect to mathematics are its formal representation, the discovery of new theorems, and to formally prove mathematical properties. However, recent activities are also devoted to the teaching of mathematics to scholars. Finally, the triumphal procession of the computer and its universal presence in all domains of our real life has created further application domains for mechanised reasoning, an important one being *formal software development*.

In order to support complex tasks in all these application domains the interaction of the user with the reasoning engine became more and more important. For teaching purposes user interaction is at the heart of the application scenario and for the other application scenarios it became apparent that both the complexity of the problems and the size of the proofs still require and will require user interaction for the foreseeable future. On the other hand, there is a need to develop interfaces for the reasoning systems that allow for the use of these systems by users that are not familiar with the foundations of formal logic. Although there has been some research in that direction, this whole area of research is still in its infancy.

2.1 Foundations of Mechanised Reasoning

Important milestones were the calculus for propositional logic of Boole [Boole, 1847] and the calculus for first-order logic of Frege in his famous "Begriffsschrift" [Frege, 1879]. Frege was the first to clearly separate syntax and semantics of formulas, and is today one of the precursors to the development of mechanised reasoning systems on a computer.

In the middle of the 19^{th} century, research was also begun to use set theory as a foundation for mathematics. Cantor defined naive set theory, which turned out to allow for antinomies that were found by Russell at the beginning of the 20^{th} century. In order to remedy these antinomies Whitehead and Russell developed the ramified theory of types and used it in their foundation of mathematics in the "Principia Mathematica" [Whitehead & Russell, 1910]. Other approaches to remedy Russell's antinomies were the axiomatic foundations of set theory by Zermelo [Zermelo, 1908] and Fraenkel [Fraenkel, 1922], Von Neumann [Von Neumann, 1928], Gödel [Gödel, 1940], and Bernays [Bernays, 1937, Bernays, 1941]. An alternative to the axiomatic approach to mathematics is constructive mathematics propagated by Brouwer [Brouwer, 1914, Brouwer, 1925] and Heyting [Heyting, 1956]. The constructive or intuitionistic position strictly rejects the axiomatic approach and the main difference is the rejection of the law of the excluded middle.

The first formulation of higher-order logic by Church [Church, 1940] was based on his theory of types and the λ-calculus. Henkin introduced the concept of general models [Henkin, 1950] as a semantic notion for higher-order logic. This was an extension of Tarski's semantic notion by which Henkin could define a complete calculus for higher-order logic.

The definition of these logics and the logical formalisation of mathematics was accompanied by research on their decidability and, if undecidability was established, on their semi-decidability and the existence of complete calculi. Hilbert was confident that his program [Hilbert, 1930] was achievable and Gödel's completeness theorem for first-order logic [Gödel, 1930] seemed to confirm this expectation. However, Hilbert's program seemed jeopardised by Gödel's, Church's and Turing's incompleteness results. Gödel showed that every system that supports the formalisation of arithmetic [Gödel, 1931] is incomplete, while Church [Church, 1936] and Turing [Turing, 1937] proved the undecidability of first-order logic. Herbrand's work proving the semi-decidability of first-order logic relativised these negative results [Herbrand, 1930]. Furthermore, Herbrand integrated parts of the syntax into the semantics of quantified formulas. This so-called "Herbrand universe" together with its properties with respect to any other semantic interpretation of these formulas smoothed the way for the implementation of computer programs with the capacity for mechanised reasoning.

A similar result to Herbrand was obtained by Gentzen [Gentzen, 1969] in the sharpened version of his Hauptsatz. Gentzen overcomes the rather unintuitive formulations of logic given by Frege, Russell, and Hilbert by defining natural deduction and sequent calculi for predicate logic. The natural

deduction calculus was reformulated by Beth [Beth, 1965] to obtain the semantic tableau calculus, a suitable basis for automated reasoning systems. At about the same time Robinson introduced the resolution calculus [Robinson, 1965], which relied on a normalisation process, the elimination of quantifiers by "Skolemisation", and a generalisation of modus ponens. The resolution calculus also proved to be very suitable for automation and has been extended to treat primitive equality by adding the paramodulation rule.

Gentzen's natural deduction and sequent calculi proceed by decomposition of the conjecture taking the structure of the formula into account and instantiating the quantified variables. The resolution calculus requires the transformation of the conjecture into a normal form. Beth's tableau calculus resides somewhere in-between these two approaches. A slightly different approach is taken by Schütte [Schütte, 1977] by defining a calculus that allows for the decomposition of formulas that are wrapped inside the whole formula, without actually requiring the decomposition of the whole formula, which is an approach very much at the heart of what is being proposed in this thesis as well.

2.2 Development of Programs for Mechanised Reasoning

The development of the first computers together with, for instance, the work of Gentzen and Herbrand made it conceivable that it should be possible to build computer programs that have non-trivial reasoning capabilities. The *logical theorist* [Newell *et al*, 1957] was the first implementation of this idea and is one of the ancestors of mechanised reasoning programs. In a first phase research was thus focused on the development of programs that were able to prove theorems automatically. These implementations were essentially based on the resolution calculus and unification [Siekmann, 1987]. In the early seventies a different approach was taken in the AUTOMATH project [De Bruijn, 1973b, De Bruijn, 1973a, De Bruijn, 1980]: the objective here was to build a system where humans can develop proofs interactively while the system guarantees soundness. This founded the tradition of interactive theorem proving. However, even in interactive theorem proving there is a clear need to support the automation of parts of proofs, for instance in order to tackle specific and simple subproblems automatically.

Both in automated and interactive theorem proving it became apparent that the implementation of the calculus rules alone is not sufficient to build automatic proof procedures. Subsequently, research concentrated on strategic and heuristic organisation of the proof search, the fine graining and specialisation of calculi, and the analysis of the properties of these systems. Researchers in automated theorem proving were interested in the design of general problem solvers, and thus the proof procedures were required to be complete. Examples for this kind of systems are OTTER [McCune, 1990], SPASS [Weidenbach, 1999], SETHEO [Letz & Stenz, 1999], VAMPIRE [Riazanov & Voronkov, 2001], MKRP [Eisinger & Ohlbach, 1986], TPS [Andrews *et al*, 1990, Andrews *et al*, 2000], or Twelf [Pfenning & Schürmann, 1999]. Proof procedures in interactive theorem proving systems were expected to be efficient for specific problems, but not necessarily complete. These systems are typically based on a natural deduction or sequent calculus such as Isabelle [Paulson, 1989], NuPrl [Constable *et al*, 1986], Oyster [Bundy *et al*, 1990a], or KIV [Heisel *et al*, 1991].

Different paradigms of how to integrate guidance information into the theorem proving systems emerged: on the one hand, guidance information was integrated into the search procedures. Prominent representatives of such procedures are the set-of-support or unit-preference strategies in automated theorem proving, as well as specific programming languages such as in the NQTHM-system [Boyer & Moore, 1979] or the "Logic for Computable Functions (LCF)" tactic language used in the meta-component of NuPrl [Constable *et al*, 1986]. On the other hand guidance information is integrated into the calculus rules themselves. This was mainly motivated by the work of Knuth and Bendix for

pure equational theorem proving, and resulted in the superposition calculus [Bachmair *et al*, 1992].

Research was also devoted to integrate domain specific knowledge into the theorem proving systems, which led to the use of sorts [Walther, 1987], the design of decision procedures à la Nelson-Oppen [Nelson & Oppen, 1977], Rippling [Bundy *et al*, 1990b, Hutter, 1990] for proofs by induction, superposition calculi for groups [Stuber, 1996] and monoids [Ganzinger & Waldmann, 1996], and many more.

All proof procedures mentioned so far, no matter which paradigm dominated their design, stick to the basic rules of the calculus. In contrast to this, Bundy introduced the notion of proof planning [Bundy *et al*, 1990b] as a paradigm advocating top-down proof construction. Proof planning consists of first finding a proof sketch by using AI planning techniques. Proof plans (or proof sketches) are built from methods, which are planning operators wrapped around a tactic [Gordon *et al*, 1979]. The pre- and postconditions of a method are declarative descriptions of the proof situation in which the tactic contained in the method should be applied together with an estimation of the new proof situation without actually having to execute the tactic. A completed proof plan is then refined to a calculus proof by executing the tactics contained in the methods in the order indicated by the proof plan. The proof planning technique is the most recent new paradigm to design and manage proof search, a line of research extensively explored in the context of the ΩMEGA proof planning system [Siekmann *et al*, 2002a]. In proof planning high-level domain specific knowledge is encoded as methods, such as knowledge about how to guide inductive proofs [Bundy, 1988], knowledge about how to do ε-δ-proofs, or proving the irrationality of square roots [Siekmann *et al*, 2002b]. Furthermore, the proof planning technique proved to be very suitable for the integration of external reasoning systems such as other automated theorem provers or computer algebra systems. Thereby these "external" proofs and computations are integrated by encoding these proofs as proof planning steps, possibly after translation [Meier, 2000, Kerber *et al*, 1998].

2.3 Application of Programs for Mechanised Reasoning

The main motivation for the development of theorem proving systems, no matter which kind, was and still is to contribute to the realisation of Leibniz' dream, i.e. to mechanise formal reasoning by a computer program. At present the most prominent application domains are mathematics, formal software development, (logical) programming developments, program synthesis, deductive databases, and teaching mathematics. But logics for "practical reasoning" are abundant and mechanized. Just as a calculus alone does not provide a theorem proving system, the successful implementation of a theorem proving system is not sufficient to use these systems for all of these purposes. In order to provide theorem proving environments that are suitable for practical applications, many additional techniques need to be developed and integrated into these systems. For example, the theorem proving system needs to be integrated with the specification of domains and problems, such as mathematical theories, lemmata and theorems, or specifications of software and their safety and security properties. Research in that direction has been executed mainly in the context of formal software development systems [Heisel *et al*, 1991, Autexier *et al*, 1998, Hutter, 2000b, Autexier *et al*, 2002] and the migration of these techniques into the application domain of mathematics has been realised inter alia in [Franke & Kohlhase, 1999, Kohlhase, 2000].

Another shortcoming of theorem proving systems is that their use requires specialist knowledge about the systems, for instance how parameter settings influence the behaviour of the search procedures for the case of automated theorem provers, or how to force the proof steps of the system into these a user would like to perform in the case of interactive theorem provers. Although there has

been some work on the development of such user-interfaces the research on that topic is still in its infancy. Examples of user-interfaces for theorem proving systems are LΩUI [Siekmann *et al*, 1999], XBarnacle [Duncan & Lowe, 1997], IsaWin [Lüth *et al*, 1999], and they can also be found in systems like VSE [Autexier *et al*, 1998], KIV [Heisel *et al*, 1991], INKA [Hutter & Sengler, 1996], and the mathematics teaching systems ACTIVEMATH [Melis *et al*, 2001]. Mechanisms that support the user by automatically generating suggestions as the possible next steps can be found in [Benzmüller & Sorge, 1999, Benzmüller & Sorge, 2000]. However, the usability of these interfaces still falls far beyond that of computer programs in other domains. Research in that direction is nevertheless essential, as it is evident that theorem proving for problems arising in real application domains still requires a great deal of user interaction. There is now a special series of workshops devoted to this topic (*User Interfaces for Theorem Provers*).

Finally, the proofs obtained both in automated and interactive theorem proving are usually in the format imposed by the calculus underlying these systems. This style of proof is far from the style by which proofs are performed by a mathematician. Work on high-level, natural language presentation of these proofs, like proofs in mathematical textbooks, can be found in [Fiedler, 2001, Dahn *et al*, 1997].

Contextual Reasoning

–

Part II

Chapter 3

Syntax, Semantics and Uniform Notation

We will now present the basic notions upon which the CoRe proof theory is built. Section 3.1 is concerned with the definition of the many-sorted, simply typed lambda-calculus as the underlying representation language for terms and formulas of the logics. The syntax and semantics of all logics considered in this thesis is presented in Sections 3.2 and 3.3. In order to ease the presentation of the proof theory (Chapter 4 and 5) we unify the notation of the syntax and semantics into a uniform terminology. The uniform terminology is further extended in Section 3.4 by recalling the uniform notation for formulas from [Wallen, 1990, Fitting, 1972, Smullyan, 1968] and extending it for our purposes. The result is the notion of syntax and semantics of signed formulas, which is the key technique that smooths the way for a uniform and simple presentation of the proof theory. We will end with an observation why functional and boolean extensionality needs to be handled in the proof theory (Section 3.5).

3.1 Terms and Subterm Occurrences

We use a many-sorted, simply typed λ-calculus as our basic representation language, as adequacy and conciseness of representation is an important aspect with respect to the area of application of the CoRe-system. Sorts are a simple tool to support a consise representation, and we allow for arbitrary many base types (i.e. sorts) of individuals, instead of using only two types ι for individuals and o for truth-values. We refrain from using a subtyping concept in order to keep the type inference simple, avoiding the intrinsic problems with subtyping.

Definition 3.1.1 (Many Sorted Higher order types) Let $\mathcal{C}$ be a set of type constants and o not in $\mathcal{C}$. The set $\mathcal{T}$ of *many sorted higher order types* induced by $\mathcal{C}$ is:

- $\mathcal{C} \subseteq \mathcal{T}$ the base types,
- $o \in \mathcal{T}$ the type for truth values,
- If $\tau_1, \tau_2 \in \mathcal{T}$, then $\tau_1 \to \tau_2 \in \mathcal{T}$ is a function type.

As usual, we assume that the functional type constructor $\to$ associates to the right, e.g. $\tau_1 \to \tau_2 \to \tau_3$ denotes $\tau_1 \to (\tau_2 \to \tau_3)$. Furthermore, we use $\tau_1 \times \ldots \times \tau_n \to \tau_0$ as an abbreviation for $\tau_1 \to (\ldots \to (\tau_n \to \tau_0))$, where $\tau_0 \in \mathcal{C} \dot{\cup} \{\mathrm{o}\}$.

We say that a type τ is a *first-order type* if, and only if, either $\tau \in \mathcal{C} \dot{\cup} \{\mathrm{o}\}$ or if it is of the form $\tau_1 \to \tau_2$ where $\tau_1 \in \mathcal{C}$ and τ_2 is a first-order type. ■

We annotate constants f_τ and variables x_τ with types τ from $\mathcal{T}$ to indicate their type. A higher-order signature $\Sigma = (\mathcal{T}, \mathcal{F}, \mathcal{V})$ consists of types $\mathcal{T}$, constants $\mathcal{F}$ and variables $\mathcal{V}$, both typed over $\mathcal{T}$. The typed λ-calculus is standard and is defined over a given higher-order signature of types $\mathcal{T}$, typed constants $\mathcal{F}$, and typed-variables $\mathcal{V}$.

Definition 3.1.2 (λ-Terms) Let $\Sigma = (\mathcal{T}, \mathcal{F}, \mathcal{V})$ be a higher-order signature. Then the typed λ-terms $\mathcal{T}_{\Sigma,\mathcal{V}}$ over Σ and $\mathcal{V}$ are:

- For all $x_\tau \in \mathcal{V}$, $x \in \mathcal{T}_{\Sigma,\mathcal{V}}$ is a term of type τ (also denoted by $x : \tau$). We say that this term x is a *variable term of type* τ.
- For all $c_\tau \in \mathcal{C}$, $c \in \mathcal{T}_{\Sigma,\mathcal{V}}$ is a term of type τ (also denoted by $c : \tau$). We say c is a *constant term of type* τ.
- If $t : \tau, t' : \tau \to \tau' \in \mathcal{T}_{\Sigma,\mathcal{V}}$ are typed terms, then $(t'\ t) \in \mathcal{T}_{\Sigma,\mathcal{V}}$ is an *application term* of type τ',
- if $x : \tau \in \mathcal{V}$ and $t : \tau' \in \mathcal{T}_{\Sigma,\mathcal{V}}$, then $\lambda x_\tau . t \in \mathcal{T}_{\Sigma,\mathcal{V}}$ is an *abstraction term* of type $\tau \to \tau'$. We say t is abstracted over x. ■

In the following we assume that terms are always well-typed and terms of type o are also called *formulas*.

Definition 3.1.3 (Substitutions) Let $\Sigma = (\mathcal{T}, \mathcal{F}, \mathcal{V})$ be a higher-order signature. A *substitution* is a type preserving function[1] $\sigma : \mathcal{V} \to \mathcal{T}_{\mathcal{F},\mathcal{V}}$ that is the identity function on $\mathcal{V}$ except for finitely many elements from $\mathcal{V}$. This allows for a finite representation of a substitution as a set of pairs:

$$\sigma := \{t_1/x_1, \ldots, t_n/x_n\}$$

where $\sigma(y) = y$ if $\forall 1 \leq i \leq n, y \neq x_i$. The *homomorphic extension* of σ to terms, i.e. the application of σ to a term, is defined by

$$\sigma(t) := \begin{cases} \sigma(x) & \text{if } t \in \mathcal{V} \\ (\sigma(t_0)\ \sigma(t_1)) & \text{if } t = (t_0\ t_1) \\ \lambda x_\tau . \sigma[x/x](t') & \text{if } t = \lambda x_\tau . t' \end{cases}$$

where $\sigma[x/t]$ denotes the function that behaves like σ except for x on which it yields t. ■

Remark 3.1.4 A substitution σ is *idempotent* if, and only if, its homomorphic extension to terms is idempotent, i.e. $\sigma(\sigma(t)) = \sigma(t)$ for all $t \in \mathcal{T}_{\Sigma,\mathcal{V}}$.

Given a substitution σ we denote by $dom(\sigma)$ the set of all variables for which $\sigma(x) \neq x$, i.e. the *domain* of σ.

Throughout the rest of this thesis we assume that every substitution is idempotent, as this can always be achieved. Higher-order λ-terms usually come with a certain set of reduction rules. We use the so-called β and η reduction rules (cf. [Barendregt, 1984]), which give rise to the $\beta\eta$ long normal form, which is unique up to renaming of bound variables. Throughout the rest of this thesis we assume that all terms are in $\beta\eta$ long normal form. Application terms in this normal form are always of the form $(c\ t_1 \ldots t_n)$ where c is either a constant from $\mathcal{F}$ or a variable from $\mathcal{V}$.

[1] i.e. for all variables $x : \tau$, $\sigma(x)$ also has type τ.

Lemma 3.1.5 (Properties of Terms in $\beta\eta$ Long Normal Form) Let t be a term in $\beta\eta$ long normal form. Then t is of one of the following forms:

- either t is a typed variable or a typed constant,
- or it is of the form $(c\ t_1\ \ldots\ t_n)$, where c is a variable or constant of type $\tau_1 \times \ldots \times \tau_n \to \tau_0$, and all t_i are in $\beta\eta$ long normal form,
- or it is of the form $\lambda x_\tau . t'$ and t' is in $\beta\eta$ long normal form. ■

Proof. Assume t is a subterm in $\beta\eta$ long normal form and fails to have any of the properties stated in the Lemma. The critical case to consider are the application terms. Thus, assume there is such a subterm s in t. There are two cases to consider:

1. *s is of the form $(t_0\ t_1 \ldots t_m)$, the type of t_0 is $\tau_1 \times \ldots \times \tau_n \to \tau_0$ and $n > m$. In this case the η-expansion rule could be applied to obtain the term $\lambda x^n_{\tau_n} . \ldots \lambda x^{m+1}_{\tau_{m+1}} . (t_0\ t_1 \ldots\ t_m\ x^{m+1} \ldots\ x^n)$ that is in normal form, which contradicts the assumption that t was in $\beta\eta$ long normal form.*
2. *s is of the form $(t_0\ t_1 \ldots t_n)$, the type of t_0 is $\tau_1 \times \ldots \times \tau_n \to \tau_0$, but t_0 is an abstraction term. In this case the β-reduction rule could be applied, which contradicts the assumption that t was in $\beta\eta$ long normal form.* □

Imposing the $\beta\eta$ normal form has the advantage that it allows for a straightforward definition of *subterm occurrences* for λ-terms.

Definition 3.1.6 (Subterm Occurrences) A *subterm occurrence* is a possibly empty list of natural numbers $[i_0, \ldots, i_n]$, ($i_j > 0$ for all $0 \leq j \leq n$). For any term t in $\beta\eta$ long normal form the set of *valid subterm occurrences* for t is the smallest set fullfilling the following properties:

- If t is a typed variable or constant, then $[]$ is the only valid subterm occurrence for t.
- If $t = (c\ t_1\ \ldots\ t_n)$ and ρ' is a valid subterm occurrence for t_i then $\rho = [i, \rho']$ is a valid subterm occurrence for t, $1 \leq i \leq n$.
- If $t = \lambda x_\tau . t'$ and ρ' is a valid subterm occurrence for t', then $\rho = [1, \rho']$ is a valid subterm occurrence for t.

If ρ is a subterm occurrence for t, we denote by $t_{|\rho}$ the subterm denoted by ρ, i.e.

- $s_{|[]} := s$,
- $(t_0\ t_1\ \ldots\ t_n)_{|[i,\rho']} := t_{i|\rho'}$, and
- $(\lambda x . t')_{|[1,\rho']} := t'_{|\rho'}$. ■

For the terms in $\beta\eta$ long normal form we define the following notion of subterms.

Definition 3.1.7 (Subterms of $\beta\eta$-Normal Terms) Let t be a term in $\beta\eta$ long normal form. The *subterms of t, $Subterms_t$,* is the smallest set that contains t and that is closed under the following rules:

- if $(c\ t_1, \ldots, t_n) \in Subterms_t$, then $t_1, \ldots, t_n \in Subterms_t$,
- if $\lambda x_\tau . t' \in Subterms_t$, then $t' \in Subterms_t$ ■

For the representation of formulas we will use the constants $\neg : o \to o$ for negation, and $\vee$ for disjunction, $\wedge$ for conjunction, $\Rightarrow$ for implication and $\Leftrightarrow$ for equivalence, all of type $o \times o \to o$. Equality over arbitrary types, including o, is denoted by $=: \tau \times \tau \to o$. We use higher-order abstract syntax (cf. [Pfenning & Elliott, 1988]) to encode quantification over object variables and introduce $\forall, \exists : (\tau \to o) \to o$ for universal and existential quantification for all types τ. To ease readability we may use $\forall x_0, \ldots, x_n . \varphi$ as an abbreviation for $\forall(\lambda x_0 . \ldots \forall(\lambda x_n . \varphi))$. Additionally, we use $\Box, \Diamond : o \to o$ for the modal operators "necessarily" and "possibly".

3.2 Syntax

The CORE-system presented in this thesis is a uniform proof system for a fixed set of logics, namely classical propositional, first-order and higher-order logics as well as classical propositional and first-order modal logics K, K4, D, D4, T, and S4 with constant domains. The reason for restricting ourselves to these logics comes from the context of this thesis, namely theorem proving for mathematics and formal software engineering. For these application areas, especially higher-order logic as well as first-order classical and modal logics are of interest. Aspiring at a uniform framework for contextual reasoning for all these logics led to the matrix characterisations for classical and modal logics by Wallen [Wallen, 1990] and for classical higher-order logics by Andrews [Andrews, 1981], Miller [Miller, 1983] and Pfenning [Pfenning, 1987]. Although the class of modal logics considered by Wallen is slightly larger, we restrict ourselves to the aforementioned in order to keep the definition of the proof theory simple.

In this section we introduce the syntax of the formulas for those logics. Since the underlying term language is the many-sorted λ-calculus with $\beta\eta$-equality the definition of the logic specific syntax consists of the definition of the admissible signatures and terms.

Definition 3.2.1 (Syntax) For each of the logic $\mathcal{L}$ we define the signatures $\Sigma_{\mathcal{L}} = (\mathcal{T}_{\mathcal{L}}, \mathcal{F}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ consisting of the base types $\mathcal{T}_{\mathcal{L}}$, the predefined constants $\mathcal{F}_{\mathcal{L}}$, and the set $\mathcal{V}_{\mathcal{L}}$ of typed variables for $\mathcal{L}$.

CPL - Classical Propositional Logic: A CPL signature is $\Sigma_{CPL} := (\{o\}, \{True, False : o, \neg : o \to o, \vee, \wedge, \Rightarrow, \Leftrightarrow : o \times o \to o\} \dot\cup C\}, \emptyset)$, where C contains only constants of the form $A : o$. The CPL terms are the subset of the λ-terms over Σ_{CPL} for which all subterms (cf. Definition 3.1.7) are of base type o.

CPML - Classical Propositional Modal Logic: A CPML signature is the pairwise disjoint union of Σ_{CPL} with $(\emptyset, \{\Box, \Diamond : o \to o\}, \emptyset)$, i.e. $\Sigma_{CPML} := \Sigma_{CPL} \dot\cup (\emptyset, \{\Box, \Diamond : o \to o\}, \emptyset)$. The CPML terms are the subset of the λ-terms over Σ_{CPML} for which all subterms (cf. Definition 3.1.7) are of base type o.

CFOL - Classical First Order Logic: A CFOL signature is $\Sigma_{CFOL} := (\{o\} \dot\cup \mathcal{S}, \{\neg : o \to o, \vee, \wedge, \Rightarrow, \Leftrightarrow : o \times o \to o, \forall, \exists : (\tau \to o) \to o$ and $=: \tau \times \tau \to o$ for all $\tau \in \mathcal{S}\} \dot\cup C\}, V)$ where the constants in C have only first-order types and the variables in V have only base types from $\mathcal{S}$. The CFOL terms are the subset of the λ-terms over Σ_{CFOL} for which all subterms (cf. Definition 3.1.7) are of some base type from $\{o\} \dot\cup \mathcal{S}$, except the direct subterms t of $\forall(t)$ and $\exists(t)$ which are of type $\tau \to o$ where τ is from $\mathcal{S}$.

CFOML - Classical First Order Modal Logic: A CFOML is the pairwise disjoint union of Σ_{CFOL} with $(\emptyset, \{\Box, \Diamond : o \to o\}, \emptyset)$, i.e. $\Sigma_{CPML} := \Sigma_{CFOL} \dot\cup (\emptyset, \{\Box, \Diamond : o \to o\}, \emptyset)$. The CFOML terms are the subset of the λ-terms over Σ_{CFOML} for which all subterms (cf. Definition 3.1.7) are of some

$\mathcal{L}$	Condition on R
K	no conditions
K4	transitive
D	seriality
D4	seriality, transitive
T	reflexive
S4	reflexive, transitive

Table 3.1: Conditions on accessibility relations.

---

base type from $\{o\} \dot{\cup} \mathcal{S}$, except the direct subterms t of $\forall(t)$ and $\exists(t)$ which are of type $\tau \to o$ where τ is from $\mathcal{S}$.

CHOL - Classical Higher-Order Logic: Assume $\mathcal{T}$ is the set of higher-order types over an arbitrary set of base types $\mathcal{S}$ and additional base type o (cf. Definition 3.1.1). Then a CHOL signature is $\Sigma_{CHOL} := (\mathcal{T}, \{True, False : o, \neg : o \to o, \vee, \wedge, \Rightarrow, \Leftrightarrow : o \times o \to o, \forall, \exists : (\tau \to o) \to o$ and $=: \tau \times \tau \to o$ for all $\tau \in \mathcal{T}\}, V)$. The CHOL terms are the λ-terms over Σ_{CHOL}. ■

3.3 Semantics

This section defines the model-theoretic semantics for the logics under consideration as follows: on the one hand we define the semantics for first-order modal logics with constant domains (CFOML), which encompass the semantics for CPL, CPML, and CFOL. On the other hand we define higher-order logic with primitive equality, as well as boolean and functional extensionality.

3.3.1 Semantics for Classical First Order Modal Logic

We follow [Wallen, 1990] for the definition of the semantics of classical first order modal logics. Let G be a non-empty set and R a binary relation on G. We denote the elements of G as the *possible worlds* and R as the *accessibility relation*. If R satisfies the conditions outlined in table 3.1, then the pair $\langle G, R \rangle$ is an $\mathcal{L}$*-frame*. Thereby the reflexivity and transitivity properties are standard, i.e. R is reflexive if, and only if, for all $v \in G$ it holds vRv and the transitivity is analogous. The *seriality*[2] *property* holds for a binary relation R over G if, and only if, for each $w \in G$ there is some $v \in G$ such that wRv holds. Note that reflexivity implies seriality.

Assume now that $\langle \mathcal{S}, \mathcal{F}, \mathcal{V} \rangle$ is a CFOML-signature. A *many sorted first-order frame* for this signature is a 4-tuple $\langle G, R, D, \overline{D} \rangle$ where $\langle G, R \rangle$ is an $\mathcal{L}$-frame, D is a sort indexed family of non-empty sets $(D_s)_{s \in \mathcal{S}}$ and $\overline{D}$ a mapping from G to a sort indexed family of non-empty subsets of the respective D_s. $\overline{D}(w)$ is the family of sets of individuals over which range quantifiers of variables with the respective types. Furthermore we require for every base type s that $D_s = \cup_{w \in G} \overline{D}(w)_s$. In the rest of this thesis we stick to the *constant domain* variants of the modal logics under consideration, i.e. we consider only first-order frames for which $\overline{D}(w) = \overline{D}(v)$ holds for all $w, v \in G$.

Definition 3.3.1 (First Order Kripke Models) A constant domain $\mathcal{L}$-frame $M = \langle G, R, D, \overline{D} \rangle$ is a first-order Kripke model if, and only if, for all $w \in G$, all assignments ρ of variables of type s to

[2] Note that in [Wallen, 1990] seriality is called idealisation.

elements of $\overline{D}(w)_s$ the following holds:

1. $\overline{D}(w)_o$ is a binary set $\{\top,\bot\}$, $\top$ for truth and $\bot$ for falsehood,
2. $M^\rho_w(\mathit{True}) = \top$, $M^\rho_w(\mathit{False}) = \bot$, and the logical functions $\neg,\wedge,\vee,\Rightarrow,\Leftrightarrow$, and $=$ have the classical interpretation,
3. $M^\rho_w(\lambda x_s \,.\, t_o)$ is the function from $\overline{D}(w)_s$ to $\overline{D}(w)_o$ that maps elements $e \in \overline{D}(w)_s$ to $M^{\rho[x/e]}_w(t)$,
4. $M^\rho_w(\forall(t_{s\to o})) = \begin{cases} \top & \text{if for all } e \in \overline{D}(w)_s\ M^\rho_w(t)(e) = \top \\ \bot & \text{otherwise} \end{cases}$
5. $M^\rho_w(\exists(t_{s\to o})) = \begin{cases} \top & \text{if there exists an } e \in \overline{D}(w)_s\ M^\rho_w(t)(e) = \top \\ \bot & \text{otherwise} \end{cases}$
6. $M^\rho_w(\Box(t)) = \begin{cases} \top & \text{if for all } v \text{ such that } wRv \text{ it holds } M^\rho_v(t) = \top \\ \bot & \text{otherwise} \end{cases}$
7. $M^\rho_w(\Diamond(t)) = \begin{cases} \top & \text{if there exists } v \text{ such that } wRv \text{ and it holds } M^\rho_v(t) = \top \\ \bot & \text{otherwise} \end{cases}$
8. Otherwise $M^\rho_w(c(t_1,\ldots,t_n)) = M_w(c)(M^\rho_w(t_1),\ldots,M^\rho_w(t_n))$, $n \geq 0$, where $M_w(c_{s_1\times\ldots\times s_n\to s_0}) \in \overline{D}(w)_{s_1} \times \ldots \overline{D}(w)_{s_n} \to \overline{D}(w)_{s_n}$. A constant $c_{s_1\times\ldots\times s_n\to s_0}$ is *rigid* if, and only if, for all possible worlds w, w' it holds $M_w(c) = M_{w'}(c)$; otherwise $c_{s_1\times\ldots\times s_n\to s_0}$ is *flexible*.

Given a CFOML-formula φ and a first-order Kripke model M, we say M *satisfies* φ if, and only if, for any world w and assignment ρ it holds $M^\rho_w(\varphi) = \top$. A CFOML-formula φ is valid, if, and only if, every first-order Kripke model satisfies φ. ■

3.3.2 Semantics for Classical Higher Order Logic

For the semantics of higher-order logic we use the general models from [Henkin, 1950] by taking into account the corrections from [Andrews, 1972]. It is based on the notion of *frames* that is a τ-indexed family $\{\overline{D}_\tau\}_{\tau\in\mathcal{T}}$ of nonempty domains, such that $\overline{D}_o = \{\top,\bot\}$[3] and $\overline{D}_{\tau_1\to\tau_2}$ is a collection of functions mapping $\overline{D}_{\tau_1}$ into $\overline{D}_{\tau_2}$. The members of $\overline{D}_o$ are called *truth values* and the members of $\overline{D}_S$, $S \in \mathcal{S}$, are called *S-individuals*.

Given a frame $\{\overline{D}_\tau\}_{\tau\in\mathcal{T}}$, an *assignment* is a function ρ on $\mathcal{V}$ such that for each variable x_τ holds $\rho(x_\tau) \in \overline{D}_\tau$. Given an assignment ρ, a variable x_τ and an element $e \in \overline{D}_\tau$ we denote by $\rho[x/e]$ that assignment ρ' such that $\rho'(x_\tau) = e$ and $\rho'(y_{\tau'}) = \rho(y_{\tau'})$, if $y_{\tau'} \neq x_\tau$.

For the definition of the function spaces in a frame we use $\Lambda x_\tau \,.\, e_{\tau'}$ to denote a function from $\overline{D}_\tau$ into $\overline{D}_{\tau'}$ in order to distinguish it from the syntax.

Definition 3.3.2 (Extensional General Models) A frame $\{\overline{D}_\tau\}_\tau$ is an extensional general model in the sense of [Andrews, 1972] if, and only if, it satisfies the following conditions:

(a_0) For each $\tau \in \mathcal{T}$, $\overline{D}_{\tau\times\tau\to o}$ contains the identity relation q on $\overline{D}_{\tau\times\tau\to o}$,

(a_1) $\overline{D}_{o\to o}$ contains the negation function n such that $n(\top) = \bot$ and $n(\bot) = \top$,

[3] Analogous to Definition 3.3.1.

(a_2) $\overline{D}_{o\to o}$ contains $\Lambda x_o \,.\, \top$ and $\Lambda x_o \,.\, x_o$. Also, $\overline{D}_{o\to o}$ contains the alternation function a such that $a(\top) = \Lambda x_o \,.\, \top$ and $a(\bot) = \Lambda x_o \,.\, x_o$,

(a_3) For each $\tau \in \mathcal{T}$, $\overline{D}_{(\tau\to o)\to o}$ contains a function $\pi_{(\tau\to o)\to o}$ such that for all $g \in \overline{D}_{\tau\to o}$ $\pi_{(\tau\to o)\to o}(g) = \top$ if, and only if, $g = \Lambda x_\tau \,.\, \top$,

(b) For all τ, τ' and all $e \in \overline{D}_\tau$ the function $\Lambda x_{\tau'} \,.\, e$ is in $\overline{D}_{\tau'\to\tau}$,

(c) For all τ, τ' the function $\Lambda x_\tau \,.\, \Lambda y_{\tau'} \,.\, x_\tau$ is in $\overline{D}_{\tau\times\tau'\to\tau}$,

(d) For all τ, τ', τ'', all $x \in \overline{D}_{\tau\times\tau'\to\tau''}$ and all $y \in \overline{D}_{\tau\to\tau'}$ the function $\Lambda z_\tau \,.\, x(z, y(z))$ is in $\overline{D}_{\tau\to\tau''}$,

(e) For all τ, τ', τ'' and all $x \in \overline{D}_{\tau\times\tau'\to\tau''}$ the function $\Lambda y_{\tau\to\tau'} \,.\, \Lambda z_\tau \,.\, x(z, y(z))$ is in $\overline{D}_{(\tau\to\tau')\times\tau\to\tau''}$,

(f) For all τ, τ', τ'' the function $\Lambda x_{\tau\times\tau'\to\tau''} \,.\, \Lambda y_{\tau\to\tau'} \,.\, \Lambda z_\tau \,.\, x(z, y(z))$ is in $\overline{D}_{(\tau\times\tau'\to\tau'')\times(\tau\to\tau')\times\tau\to\tau''}$.

The interpretation of a λ-term t by an extensional general model $N := \{\overline{D}_\tau\}_\tau$ and with respect to an assignment ρ is the usual interpretation defined by:

- $M(o) := \overline{D}_o = \{\top, \bot\}$,
- $M^\rho(\mathit{True}) := \top, M^\rho(\mathit{False}) := \bot$, $M^\rho(\neg) := n$, $M^\rho(=_{\tau\times\tau\to o}) := q \in \overline{D}_{\tau\times\tau\to o}$, and the logical functions $\wedge, \vee, \Rightarrow$, and $\Leftrightarrow$ have the classical interpretation,
- $M^\rho(\forall_{(\tau\to o)\to o}) := \pi \in \overline{D}_{(\tau\to o)\to o}$, and $M^\rho(\exists_{(\tau\to o)\to o}) := M^\rho(\lambda x_{\tau\to o} \,.\, \neg(\forall(\lambda y_\tau \,.\, \neg(xy))))$,
- $M^\rho(c_\tau) \in \overline{D}_\tau$, for any constant c_τ,
- $M^\rho(x_\tau) := \rho(x_\tau) \in \overline{D}_\tau$, for any variable x_τ,
- $M^\rho(t_0\, t_1, \ldots, t_n) := M^\rho(t_0)(M^\rho(t_1), \ldots, M^\rho(t_n))$,
- $M^\rho(\lambda x_\tau \,.\, t_{\tau'})$ is the function from $\overline{D}_\tau$ to $\overline{D}_{\tau'}$ that maps every element $e \in \overline{D}_\tau$ to $M^{\rho[e/x]}(t)$. ■

3.3.3 Unifying Notations

For every logic $\mathcal{L}$ mentioned in the previous section we assume there is a unique uniform notation for signed formulas with respect to $\mathcal{L}$. Furthermore, we agree to omit type information on variables and constants, unless they are required. In order to simplify the presentation of the meta proof theory for the whole class of logics, the semantics of the different logics are unified to a uniform notion of a model.

Definition 3.3.3 ($\mathcal{L}$-Models) Let $\mathcal{L}$ be one of the logics under consideration. An $\mathcal{L}$-formula φ is $\mathcal{L}$-satisfiable if, and only if, there is an $\mathcal{L}$-model M such that for all variable assignments ρ and all worlds w it holds $M^\rho_w(\varphi) = \top$. An $\mathcal{L}$-formula φ is $\mathcal{L}$-valid if, and only if, it is $\mathcal{L}$-satisfiable in all $\mathcal{L}$-models. ■

α	α_0	α_1
$(\varphi \vee \psi)^+$	φ^+	ψ^+
$(\varphi \Rightarrow \psi)^+$	φ^-	ψ^+
$(\varphi \wedge \psi)^-$	φ^-	ψ^-
$(\neg\varphi)^+$	φ^-	$-$
$(\neg\varphi)^-$	φ^+	$-$

γ	$\gamma_0(c)$
$(\forall x \,.\, \varphi)^-$	$(\varphi[x/t])^-$
$(\exists x \,.\, \varphi)^+$	$(\varphi[x/t])^+$

ν	ν_0
$(\Box\varphi)^-$	φ^-
$(\Diamond\varphi)^+$	φ^+

β	β_0	β_1
$(\varphi \wedge \psi)^+$	φ^+	ψ^+
$(\varphi \vee \psi)^-$	φ^-	ψ^-
$(\varphi \Rightarrow \psi)^-$	φ^+	ψ^-

δ	$\delta_0(c)$
$(\forall x \,.\, \varphi)^+$	$(\varphi[x/c])^+$
$(\exists x \,.\, \varphi)^-$	$(\varphi[x/c])^-$

π	π_0
$(\Box\varphi)^+$	φ^+
$(\Diamond\varphi)^-$	φ^-

Figure 3.1: Uniform notation.

3.4 Uniform Notation

The meta proof-theory for the CORE framework relies on an extension of indexed formula trees which makes use of the concept of polarities and uniform notation (cf. [Wallen, 1990,Fitting, 1972,Smullyan, 1968]). Polarities are assigned to formulas and subformulas and are either positive ($+$) or negative ($-$). Intuitively, positive polarity of a subformula indicates that it occurs in the succedent of a sequent in a sequent calculus proof and negative polarity is for formulas occurring in the antecedent of a sequent.

Formulas annotated with polarities are called *signed formulas*. Uniform notation assigns *uniform types* to signed formulas which encode their "behaviour" in a sequent calculus proof: there are two propositional uniform types α and β, two types γ and δ for quantification over object variables, and two types π and ν for modal quantification. A signed formula is of type α if the subformulas obtained by application of the respective sequent calculus decomposition rule on the formula both occur in the same sequent. Signed formulas are of type β, if the decomposition of the signed formula gives rise to a split of the sequent calculus proof and the obtained subformulas occur in different sequents. γ-type signed formulas indicate that the bound variable is freely instantiable, while δ-type signed formulas are those for which the *Eigenvariable* condition must hold. We call γ-*variable* (resp. δ-*variable*) variables bound on some γ-type signed (sub-)formula (resp. δ-type). In Figure 3.1 we give the list of signed formulas for each uniform type.

The tables indicate also how the polarity of a signed formula is inherited to its respective subformulas. Furthermore, they define a recursion principle to annotate all subformulas of a signed formula with a polarity and a uniform type. This recursion principle is the basis for indexed formula trees, which are signed formulas where each subformula is annotated with its polarity and uniform type.

Polarities and uniform notation are sufficient to define a uniform notion of a logical context and to determine the usable rules from the logical context. Due to the uniform results in [Wallen, 1990] the framework can be instantiated to support the intuitive proof search with respect to a variety of logics, namely propositional, first-order classical logic, and some propositional and first-order modal logics. Indexed formula trees are closely related to expansion trees from [Andrews, 1981, Miller, 1983, Pfenning, 1987, Andrews, 1989] which allows the treatment of higher-order classical logics.

ε	ε_0	ε_1
$(s \Leftrightarrow t)^-$	s°	t°
$(s = t)^-$	s°	t°

ζ	ζ_0	ζ_1
$(s \Leftrightarrow t)^+$	s°	t°
$(s = t)^+$	s°	t°

Figure 3.2: Uniform types for equations and equivalences.

We are mainly interested in first-order versions of the logics as well as in higher-order logic, and in order to keep the framework simple, we restrict ourselves to first-order and higher-order classical logic, and first-order modal logics with *constant domains*.

An important intuitive concept is equality and equivalence and we want to treat those as first-class citizens by supporting their use as rewrite rules. Example: given an equation $s = t$ and a formula $\varphi(s)$ it is natural to allow the rewrite of $\varphi(s)$ to $\varphi(t)$. Similarly we want to support the rewriting with equivalence, i.e. to apply $P \Leftrightarrow Q$ on $\varphi(P)$ to obtain $\varphi(Q)$. Note that we cannot assign polarities to P and Q in $P \Leftrightarrow Q$, while P in $\varphi(P)$ may well have a polarity. Furthermore, the uniform notion of rules obtained from uniform notation is restricted to logical refinement rules and does not capture equivalence rules. In order to capture equations and equivalence we introduce a third polarity $\circ$, *undefined*, and uniform types ε and ζ for negative and positive equations and equivalences.

Definition 3.4.1 (Polarities) We introduce three kinds of polarities: a *positive polarity* $+$ (intuitively succedent of a sequent), a *negative polarity* $-$ (intuitively antecedent of a sequent), and an *undefined polarity* $\circ$. We say that a polarity is *defined*, if it is either positive or negative. ■

The definition of the additional uniform types is given in Figure 3.2. The new uniform types extend the notion of a rule to capture uniformly logical equivalence rules, in contrast to logical refinement rules.

In the rest of this thesis we are mainly concerned with signed formulas. To ease the presentation we extend the notion of $\mathcal{L}$-satisfiability to signed formulas. In order to motivate this definition consider a sequent $\psi_1, \ldots, \psi_n \vdash \varphi$. It represents the proof status that we have to prove φ from the ψ_i. In terms of polarities, all the ψ_i have negative polarity while φ has positive polarity. The ψ_i are the assumptions and thus we consider the $\mathcal{L}$-models that satisfy those formulas and prove that those $\mathcal{L}$-models also satisfy φ. Hence, we define that an $\mathcal{L}$-model M satisfies a *negative* formula ψ_i^- if, and only if, M satisfies ψ_i. From there we derive the dual definition for positive formulas, namely that an $\mathcal{L}$-model M satisfies a positive formula φ^+, if, and only if, M *does not* satisfy φ. Formally:

Definition 3.4.2 ($\mathcal{L}$-Satisfiability of Signed Formulas) Let φ^p be a signed $\mathcal{L}$-formula of defined polarity p and M an $\mathcal{L}$-model. Then:

$$M \models_{\mathcal{L}} \varphi^+ \quad \text{if, and only if,} \quad M \not\models_{\mathcal{L}} \varphi$$
$$M \models_{\mathcal{L}} \varphi^- \quad \text{if, and only if,} \quad M \models_{\mathcal{L}} \varphi$$

■

From this definition we can infer for each uniform type $\alpha, \beta, \gamma, \delta, \nu$, and π the relationship between the satisfiability of a signed formula of this type and the satisfiability of its signed components.

Lemma 3.4.3 Let M be an $\mathcal{L}$-model, ρ a variable assignment, w, v worlds, and φ^p a signed $\mathcal{L}$-formula of polarity p. Then it holds:

1. If $\varphi^p = \alpha^p(\varphi_1^{p_1}, \varphi_2^{p_2})$, then

$$M_w^\rho \models_{\mathcal{L}} \alpha^p(\varphi_1^{p_1}, \varphi_2^{p_2}) \text{ if, and only if, } M_w^\rho \models_{\mathcal{L}} \varphi_1^{p_1} \text{ and } M_w^\rho \models_{\mathcal{L}} \varphi_2^{p_2}$$

2. If $\varphi^p = \beta^p(\varphi_1^{p_1}, \varphi_2^{p_2})$, then

$$M_w^\rho \models_{\mathcal{L}} \beta^p(\varphi_1^{p_1}, \varphi_2^{p_2}) \text{ if, and only if, } M_w^\rho \models_{\mathcal{L}} \varphi_1^{p_1} \text{ or } M_w^\rho \models_{\mathcal{L}} \varphi_2^{p_2}$$

3. If $\varphi^p = \gamma^p x \,.\, \varphi_1^p$, then

$$M_w^\rho \models_{\mathcal{L}} \gamma^p x \,.\, \varphi_1^p \text{ if, and only if, for all } a \in \mathcal{D}_\tau,\ M_w^{\rho[a/x]} \models_{\mathcal{L}} \varphi_1^p$$

4. If $\varphi^p = \delta^p x \,.\, \varphi_1^p$, then

$$M_w^\rho \models_{\mathcal{L}} \delta^p x \,.\, \varphi_1^p \text{ if, and only if, there is an } a \in \mathcal{D}_\tau \text{ such that } M_w^{\rho[a/x]} \models_{\mathcal{L}} \varphi_1^p$$

5. If $\varphi^p = \nu^p(\varphi_1^p)$, then

$$M_w^\rho \models_{\mathcal{L}} \nu^p(\varphi_1^p) \text{ if, and only if, for all } v \text{ such that } wRv \text{ and } M_v^\rho \models_{\mathcal{L}} \varphi_1^p$$

6. If $\varphi^p = \pi^p(\varphi_1^p)$, then

$$M_w^\rho \models_{\mathcal{L}} \pi^p(\varphi_1^p) \text{ if, and only if, there is a } v \text{ such that } wRv \text{ and } M_v^\rho \models_{\mathcal{L}} \varphi_1^p$$

■

Proof.

1. *α-type formulas: we show the proof if φ^p is of the form $(\varphi_1^- \Rightarrow \varphi_2^+)^+$. The other cases can be proved analogously:*

$$\begin{array}{rcl} M_w^\rho \models_{\mathcal{L}} (\varphi_1 \Rightarrow \varphi_2)^+ & \Leftrightarrow & M_w^\rho \not\models_{\mathcal{L}} (\varphi_1 \Rightarrow \varphi_2) \\ & \Leftrightarrow & M_w^\rho \not\models_{\mathcal{L}} \neg\varphi_1 \text{ and } M_w^\rho \not\models_{\mathcal{L}} \varphi_2 \\ & \Leftrightarrow & M_w^\rho \models_{\mathcal{L}} \varphi_1 \text{ and } M_w^\rho \not\models_{\mathcal{L}} \varphi_2 \\ & \Leftrightarrow & M_w^\rho \models_{\mathcal{L}} \varphi_1^- \text{ and } M_w^\rho \models_{\mathcal{L}} \varphi_2^+ \end{array}$$

2. *β-type formulas: we show the proof if φ^p is of the form $(\varphi_1^+ \Rightarrow \varphi_2^-)^-$. The other cases can be proved analogously:*

$$\begin{array}{rcl} M_w^\rho \models_{\mathcal{L}} (\varphi_1 \Rightarrow \varphi_2)^- & \Leftrightarrow & M_w^\rho \models_{\mathcal{L}} (\varphi_1 \Rightarrow \varphi_2) \\ & \Leftrightarrow & M_w^\rho \models_{\mathcal{L}} \neg\varphi_1 \text{ or } M_w^\rho \models_{\mathcal{L}} \varphi_2 \\ & \Leftrightarrow & M_w^\rho \not\models_{\mathcal{L}} \varphi_1 \text{ or } M_w^\rho \models_{\mathcal{L}} \varphi_2 \\ & \Leftrightarrow & M_w^\rho \models_{\mathcal{L}} \varphi_1^+ \text{ or } M_w^\rho \models_{\mathcal{L}} \varphi_2^- \end{array}$$

3. *γ-type formulas: we present the proof when φ^p is of the form $\forall^- x_\tau \,.\, \varphi'^-$. The case of $\exists^+ x_\tau \,.\, \varphi'^+$ is analogous.*

$$\begin{array}{rcl} & & M_w^\rho \models_{\mathcal{L}} \forall^- x_\tau \,.\, \varphi'^- \\ & \Leftrightarrow & M_w^\rho \models_{\mathcal{L}} \forall x_\tau \,.\, \varphi' \\ & \Leftrightarrow & \text{For all } a \in \mathcal{D}_\tau \text{ such that } M_w^{\rho[a/x]} \models_{\mathcal{L}} \varphi' \\ & \Leftrightarrow & \text{For all } a \in \mathcal{D}_\tau \text{ such that } M_w^{\rho[a/x]} \models_{\mathcal{L}} \varphi'^- \end{array}$$

4. δ-type formulas: *we present the proof when* φ^p *is of the form* $\exists^- x_\tau \,.\, \varphi'^-$. *The case of* $\forall^+ x_\tau \,.\, \varphi'^+$ *is analogous.*

$$\begin{array}{ll} & M_w^\rho \models_{\mathcal{L}} \exists^- x_\tau \,.\, \varphi'^- \\ \Leftrightarrow & M_w^\rho \models_{\mathcal{L}} \exists x_\tau \,.\, \varphi' \\ \Leftrightarrow & \text{There exists } a \in \mathcal{D}_\tau \text{ such that } M_w^{\rho[a/x]} \models_{\mathcal{L}} \varphi' \\ \Leftrightarrow & \text{There exists } a \in \mathcal{D}_\tau \text{ such that } M_w^{\rho[a/x]} \models_{\mathcal{L}} \varphi'^- \end{array}$$

5. ν-type formulas: *we present the proof when* φ^p *is of the form* $\Box^- \varphi'^-$. *The case of* $\Diamond^+ \varphi'^+$ *is analogous.*

$$\begin{array}{ll} & M_w^\rho \models_{\mathcal{L}} \Box^- \varphi'^- \\ \Leftrightarrow & M_w^\rho \models_{\mathcal{L}} \Box \varphi' \\ \Leftrightarrow & \text{For all } v \text{ with } wRv \text{ it holds } M_v^\rho \models_{\mathcal{L}} \varphi' \\ \Leftrightarrow & \text{For all } v \text{ with } wRv \text{ it holds } M_v^\rho \models_{\mathcal{L}} \varphi'^- \end{array}$$

6. π-type formulas: *we present the proof when* φ^p *is of the form* $\Diamond^- \varphi'^-$. *The case of* $\Box^+ \varphi'^+$ *is analogous.*

$$\begin{array}{ll} & M_w^\rho \models_{\mathcal{L}} \Diamond^- \varphi'^- \\ \Leftrightarrow & M_w^\rho \models_{\mathcal{L}} \Diamond \varphi' \\ \Leftrightarrow & \text{There exists } v \text{ with } wRv \text{ such that } M_v^\rho \models_{\mathcal{L}} \varphi' \\ \Leftrightarrow & \text{There exists } v \text{ with } wRv \text{ such that } M_v^\rho \models_{\mathcal{L}} \varphi'^- \end{array}$$

□

3.5 Preliminary Remarks

Before presenting the formal definition of indexed formula trees we have a quick look at the problems of rewriting with equations and equivalences as the theories of indexed formula trees [Wallen, 1990] and expansion trees [Andrews, 1981, Miller, 1983, Pfenning, 1987, Andrews, 1989] do not provide a direct support for rewriting with equations and equivalences. In [Pfenning, 1987] primitive equality is handled by expanding an equation $s_\tau = t_\tau$ into Leibniz' equality $\forall P_{\tau\to o} \,.\, P(s) \Rightarrow P(t)$. For rewriting equations and equivalences the idea is to use Leibniz' equality in the following manner: assume we have an equation from the definition of addition over natural numbers

$$\forall(\lambda x \,.\, \forall(\lambda y \,.\, (\mathrm{s}(x) + y) = \mathrm{s}(x + y))) \tag{3.1}$$

where s is the successor on natural numbers. Rewriting a formula $Q(\mathrm{s}(a) + b)$ with this equation results in $P(\mathrm{s}(a + b))$ and we can encode this by replacing (3.1) with

$$\forall(\lambda x \,.\, \forall(\lambda y \,.\, \forall(\lambda P \,.\, P(\mathrm{s}(x) + y) \Rightarrow P(\mathrm{s}(x + y))))) \tag{3.2}$$

and instantiating P with $\lambda u \,.\, Q(u)$ in order to obtain

$$Q(\mathrm{s}(x) + y) \Rightarrow Q(\mathrm{s}(x + y)). \tag{3.3}$$

Applying (3.3) on $Q(\mathrm{s}(a) + b)$ we obtain $Q(\mathrm{s}(a + b))$, which is the desired result. Thus, the idea is to have rewriting as a primitive rule and to encode it internally as a sequence of expansion, instantiation and application of Leibniz' equality. However, this already fails for equations in the presence of quantified formulas that have undefined polarities. Example: assume we want to apply (3.1) on the formula

$$\forall(\lambda z \,.\, R(\mathrm{s}(z) + b)) \Leftrightarrow S. \tag{3.4}$$

Intuitively (3.1) is applicable on (3.4) and it should rewrite it into

$$\forall(\lambda z \,.\, R(\mathrm{s}(z+b))) \Leftrightarrow S \tag{3.5}$$

Expanding (3.1) we obtain again (3.2), but we fail to construct an instance t for P such that the $\beta\eta$ long normal form for $t(\mathrm{s}(x)+y)$ is unifiable by some σ with $\forall(\lambda z \,.\, R(\mathrm{s}(z)+b)) \Leftrightarrow S$ *and* the $\beta\eta$ long normal form $\sigma(t(\mathrm{s}(x+y)))$ is equal to (3.5). This problem shows up because we want to rewrite within the scope of quantifiers without polarity. The way around that problem is to have an extensionality rule which transforms (3.1) into

$$\forall(\lambda y \,.\, (\lambda x \,.\, (\mathrm{s}(x)+y) = \lambda x \,.\, \mathrm{s}(x+y))) \tag{3.6}$$

and to use that equation for the rewriting [4] . The Leibniz' equality for (3.6) is

$$\forall(\lambda y \,.\, \forall(\lambda P \,.\, P(\lambda x \,.\, (\mathrm{s}(x)+y)) \Rightarrow P(\lambda x \,.\, \mathrm{s}(x+y)))). \tag{3.7}$$

Now we can instantiate P with $\lambda u \,.\, (\forall(\lambda z \,.\, u(z))) \Leftrightarrow S$. Applying the instantiation on (3.7) we obtain

$$\forall(\lambda y \,.\, [\forall(\lambda z \,.\, \mathrm{s}(z)+y) \Leftrightarrow S] \Rightarrow [\forall(\lambda z \,.\, \mathrm{s}(z+y)) \Leftrightarrow S]. \tag{3.8}$$

Now $(\forall(\lambda z \,.\, \mathrm{s}(z)+y)) \Leftrightarrow S$ is unifiable with (3.4) by instantiating y with b and we obtain the desired result from (3.5). Thus, our solution to support intuitive rewriting will be to encode it via a sequence of extensionality steps, expansion of primitive equality into Leibniz' equality, and standard logical refinement.

3.5.1 Sketch of the CoRe Proof-Theory

The proof-theoretical framework for CoRe relies entirely on the (extended) uniform notation and is based on a notion of a proof state that consists of two complementary parts: the first is an indexed formula tree for the initial conjecture and this is used to check the validity of substitutions. The second part is the working copy of the first indexed formula tree which is actively transformed by the CoRe calculus rules. The working copy is an indexed formula tree with free variables, and there is a one-to-one mapping between the free variables and binding positions in the first indexed formula tree. The secondary uniform type of the binding positions in the first indexed formula tree also indicates the types γ or δ of the free variables.

The uniform notation is the basis for a uniform definition of a logical context and replacement rules in free variable indexed formula trees. The reasoning rules are (1) contraction, (2) weakening, (3) permutations of modal quantifiers over logical connectives, (4) resolution replacement rule application, (5) propositional simplification, (6) expansion of ε- and ζ-type formulas into Leibniz' equalities, (7) extensionality over γ-variables for particular ε-type signed formulas and δ-variables for particular ζ-type signed formulas, (8) the boolean ζ-expansion rule that expands positive equivalences into the conjunction of two implications, (9) instantiation, (10) the increase of multiplicities of γ-and ν-type quantifiers, (11) the application of rewriting replacement rules, and (12) Cut.

[4] This extensionality rule is the (ξ)-rule from [Hindley & Seldin, 1986].

Chapter 4

Indexed Formula Trees

A CoRe proof state consists of an indexed formula tree and a free variable copy of the indexed formula tree. This chapter introduces indexed formula trees, which are a combination of the indexed formula trees in [Wallen, 1990] with a variant of the expansion tree proofs in [Miller, 1983, Pfenning, 1987].

We introduce the notion of an indexed formula tree in two steps: first we define the indexed formula tree obtained initially from a formula, which we denote by *initial indexed formula tree*. In a second step we add nodes that represent the introduction of Leibniz' equality, extensionality introduction for ε- and ζ-type formulas, boolean ζ-expansion as well as the introduction of cut. Furthermore we define the application of substitutions as well as the handling of new variables, for instance those generated by higher-order unification.

In the following we agree to denote by $\alpha^p(\alpha_1^{p_1}, \alpha_2^{p_2})$ a signed formula of polarity p, uniform type α, and subformulas α_i with respective polarities p_i according the tables in Figure 3.1 (p. 26). By abuse of notation we also allow the replacement of the α_i by new formulas. Example: if $\alpha^+(\alpha_1^{p_1}, \alpha_2^{p_2})$ is $(A^- \Rightarrow B^+)^+$, then $\alpha^p(C, \alpha_2^{p_2})$ denotes $(C^- \Rightarrow B^+)^+$. We use an analogous notation for formulas of the other uniform types. Furthermore we agree to denote by $\overline{\alpha}^p(\varphi_1^{p_1}, \ldots, \varphi_n^{p_n})$ either the single signed formula $\varphi_1^{p_1}$, if $n = 1$, and otherwise the signed formula $\alpha^p(\varphi_1^{p_1}, \overline{\alpha}^{p_2}(\varphi_2^{p_2}, \ldots, \varphi_n^{p_n}))$. Analogously we define $\overline{\beta}$.

4.1 Initial Indexed Formula Trees

Definition 4.1.1 (Initial Indexed Formula Tree) We define *initial indexed formula trees* inductively over the structure of formulas. Each node of the tree has a formula as label, a polarity, and a uniform type. All nodes, except for the root node have also secondary uniform types, which is the uniform type of their parent nodes.

1. If A^p is a signed atom of polarity p and without uniform type, then $Q = A^p_-$ is an initial indexed formula tree of polarity p and no uniform type, which is indicated by the subscript $-$. Those literal nodes are leaves of indexed formula trees and $\mathbf{Label}(Q) := A$.

2. If $\varepsilon(s,t)^p$ is a signed formula of polarity p and uniform type ε, then $Q = \varepsilon(s,t)^p_\varepsilon$ is an initial indexed formula tree of polarity p and uniform type ε. They are also leaves of indexed formula trees and $\mathbf{Label}(Q) := \varepsilon(s,t)$.

3. If $\zeta(s,t)^p$ is a signed formula of polarity p and uniform type ζ, then $Q = \zeta(s,t)^p_\zeta$ is an initial indexed formula tree of polarity p and uniform type ζ. They are the last kind of leaves of indexed formula trees and $\mathbf{Label}(Q) := \zeta(s,t)$.

4. Let Q' be an initial indexed formula tree of polarity p and $\mathbf{Label}(Q') = \varphi$ and $\alpha^{-p}(\varphi^p)$ a signed formula with the opposite polarity $-p$. Then

$$Q = \begin{array}{c} \alpha(\varphi)^{-p}_\alpha \\ | \\ Q' \end{array}$$

is an initial indexed formula tree with $\mathbf{Label}(Q) := \alpha(\varphi)$, of polarity $-p$ and uniform type α. The secondary type of Q' is α_1.

5. Let Q_1, Q_2 be initial indexed formula trees with respective polarities p_1 and p_2, and assume a signed formula $\alpha^p(\mathbf{Label}(Q_1)^{p_1}, \mathbf{Label}(Q_2)^{p_2})$ of polarity p. Then

$$Q = \begin{array}{c} \alpha(\mathbf{Label}(Q_1), \mathbf{Label}(Q_2))^p_\alpha \\ \diagup \quad \diagdown \\ Q_1 \qquad Q_2 \end{array}$$

is an initial indexed formula tree with $\mathbf{Label}(Q) := \alpha(\mathbf{Label}(Q_1), \mathbf{Label}(Q_2)$, polarity p, and uniform type α. The secondary types of Q_1 and Q_2 are α_1 and α_2.

6. Let Q_1, Q_2 be initial indexed formula trees with respective polarities p_1 and p_2, and assume a signed formula $\beta^p(\mathbf{Label}(Q_1)^{p_1}, \mathbf{Label}(Q_2)^{p_2})$ of polarity p. Then

$$Q = \begin{array}{c} \beta(\mathbf{Label}(Q_1), \mathbf{Label}(Q_2))^p_\beta \\ \diagup \quad \diagdown \\ Q_1 \qquad Q_2 \end{array}$$

is an initial indexed formula tree with $\mathbf{Label}(Q) := \beta(\mathbf{Label}(Q_1), \mathbf{Label}(Q_2)$, of polarity p and uniform type β. The secondary types of Q_1 and Q_2 are β_1 and β_2.

7. Let $\gamma^p x \,.\, \varphi(x)$ be a signed formula of polarity p, and $Q_i, 1 \leq i \leq n$ initial indexed formula trees with $\mathbf{Label}(Q_i) = \varphi(X_i)$ where the X_i are new (meta) variables. Then

$$Q = \begin{array}{c} \gamma^p x \,.\, \varphi(x)^p_\gamma \\ \diagup \quad | \quad \diagdown \\ Q_1 \quad \ldots \quad Q_n \end{array}$$

is an initial indexed formula tree with $\mathbf{Label}(Q) := \gamma^p x \,.\, \varphi(x)$, of polarity p and uniform type γ. All the Q_i then have secondary type γ_0. The multiplicity of Q is n.
For each $1 \leq i \leq n$ we say that Q_i is the binding node for X_i. We also call a meta variable X_i a γ-variable.

8. Let $\delta^p x \,.\, \varphi(x)$ be a signed formula of polarity p, and Q' an initial indexed formula trees with $\mathbf{Label}(Q') = \varphi(x)$ where x is a new parameter. Then

$$Q = \begin{array}{c} \delta^p x \,.\, \varphi(x)^p_\delta \\ | \\ Q' \end{array}$$

is an initial indexed formula tree with $\mathbf{Label}(Q) := \delta^p x \,.\, \varphi(x)$, of polarity p and uniform type δ. The secondary type of Q' is δ_0.
We say that Q' is the binding position for x. We also call a parameter x a δ-variable.

9. Let $Q_i, 1 \leq i \leq n$ be initial indexed formula trees all with the same labels up to renaming of bound variables and the same polarity p. Further let $\nu^p(\mathbf{Label}(Q_1))$ be a signed formula of polarity p. Then

$$Q = \begin{array}{c} \nu(\mathbf{Label}(Q_1))^p_\nu \\ \diagup \;\; | \;\; \diagdown \\ Q_1 \quad \ldots \quad Q_n \end{array}$$

is an initial indexed formula tree with $\mathbf{Label}(Q) := \nu(\mathbf{Label}(Q_1))$, of polarity p, and uniform type ν. All the Q_i then have secondary type ν_0. The multiplicity of Q is n.

10. Let Q' be an initial indexed formula trees of polarity p and $\pi^p(\mathbf{Label}(Q'))$ a signed formula of polarity p. Then

$$Q = \begin{array}{c} \pi(\mathbf{Label}(Q))^p_\delta \\ | \\ Q' \end{array}$$

is an initial indexed formula tree with $\mathbf{Label}(Q) := \pi(\mathbf{Label}(Q'))$, of polarity p and uniform type π. The secondary type of Q' is π_0. ■

Example 4.1.2 As an example for an initial indexed formula tree we consider the formula about natural numbers

$$\begin{array}{l}(\forall x_{Nat} \,.\, 0 + x = x) \wedge (\forall x_{Nat} \,.\, \forall y_{Nat} \,.\, \neg(x = 0) \Rightarrow x + y = \mathrm{s}(\mathrm{p}(x) + y)) \\ \Rightarrow \forall p_{Nat \to \mathrm{o}} \,.\, \forall v_{Nat} \,.\, p(\mathrm{s}(\mathrm{s}(0)) + v) \Leftrightarrow P(\mathrm{s}(\mathrm{s}(v)))\end{array}$$

where *Nat* denotes the type of natural numbers, 0_{Nat} is the zero of natural numbers, $\mathrm{s}_{Nat \to Nat}$ and $\mathrm{p}_{Nat \to Nat}$ denote repsectively the successor and predecessor of some natural number. Then the initial indexed formula tree for the positive formula is viewed in Figure 4.1 (p. 34). In the following sections we will use this initial indexed formula tree to illustrate different rules on indexed formula trees.

Notation 4.1.3 (Γ_0, Δ_0, ∇_0 **and** Π_0) For a given (initial) indexed formula tree we denote the set of nodes of secondary type γ_0 by Γ_0, and analogously we define the sets Δ_0, ∇_0 and Π_0.

The proof-theory defined for CoRe is a meta proof-theory in that it supports a variety of logics. To this end (following [Wallen, 1990]) there are two kinds of substitutions: one for the instantiation of meta variables and one that deals with the modal part of the considered logics. The former is called a *variable substitution*, denoted by σ_Q, and instantiates meta variables bound on nodes of secondary type γ_0. Following [Wallen, 1990, Miller, 1983, Pfenning, 1987] the occur-check is realised as an acyclicity check of a directed graph obtained from the structure of the indexed formula tree and additional edges between binding nodes of the instantiated meta variable X and binding nodes of parameters occurring in $\sigma_Q(X)$. We first introduce the *structural ordering* $\prec$ induced by the structure of an indexed formula tree and the *quantifier ordering* $\prec_V$ induced by a variable substitution.

Definition 4.1.4 (Structural Ordering) Let Q be an indexed formula tree. The *structural ordering* $\prec_Q$ is a binary relation among the nodes in Q defined by: $Q_1 \prec_Q Q_2$ iff Q_1 dominates Q_2 in Q. ■

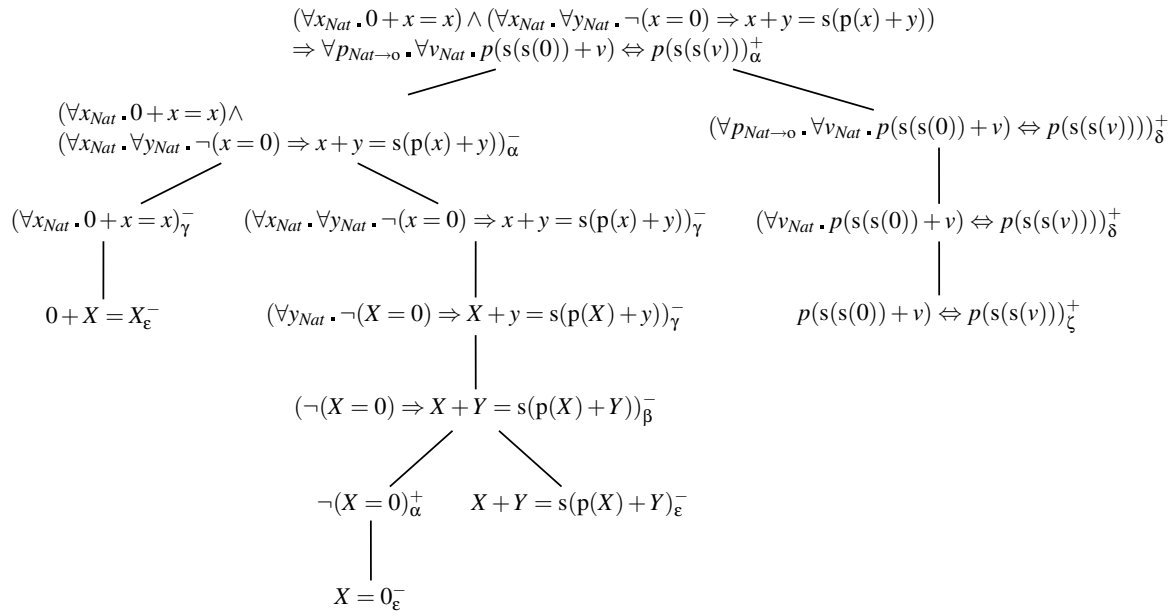


Figure 4.1: Initial indexed formula tree for the running example.

Definition 4.1.5 (Quantifier Ordering) Let Q be an indexed formula tree and σ an (idempotent) substitution for meta variables bound on γ_0-type positions in Q by terms containing only meta variables and parameters bound in Q. The *quantifier ordering* $\prec_V$ induced by σ is the binary relation defined by: $Q_0 \prec_V Q_1$ iff there is an $X \in dom(\sigma)$ bound on Q_1 and there occurs in $\sigma(X)$ a parameter bound on Q_0. ∎

The second kind of substitutions is called a *modal substitution* and is denoted by σ_M. In order to define it properly, we introduce the notion of *modal prefixes* for the modal logics under consideration, i.e. the modal logics K, K4, D, D4, T, and S4.

For these modal logics the modal prefix is defined for some node in the indexed formula tree and is the sequence of nodes from $\mathbf{V}_0$ and Π_0 that govern that node.

Definition 4.1.6 (Modal Prefix) Let Q be a node in an indexed formula tree. The *modal prefix* $pre(Q)$ of Q is a sequence $\langle Q_1, \ldots, Q_n \rangle \in (\mathbf{V}_0 \cup \Pi_0)^*$ and is defined as follows:

$$pre(Q) := \begin{cases} \langle\rangle & \text{if } Q \text{ has no parent node} \\ \langle Q' :: pre(Q') \rangle & \text{if } Q \text{ has parent node } Q' \in \mathbf{V}_0 \cup \Pi_0 \\ pre(Q') & \text{if } Q \text{ has parent node } Q' \notin \mathbf{V}_0 \cup \Pi_0 \end{cases}$$

where $\langle Q :: pre(Q') \rangle$ denotes the sequence obtained from adding Q as first element to the sequence $pre(Q')$. ∎

In order to identify two nodes the modal prefixes of both nodes must be equal. To this end the nodes from $\mathbf{V}_0$ are treated like meta variables and a modal substitution is a substitution of nodes from $\mathbf{V}_0$ by sequences of nodes from $\mathbf{V}_0 \cup \Pi_0$. Additionally, a modal substitution must respect the so-called

Property	Condition	
general	$p\ R_0\ pu$	$p \in (\mathbf{V}_0 \cup \Pi_0)^*, u \in (\mathbf{V}_0 \cup \Pi_0)$
reflexive	$p\ R_0\ p$	$p \in (\mathbf{V}_0 \cup \Pi_0)^*$
transitive	$p\ R_0\ pq$	$p \in (\mathbf{V}_0 \cup \Pi_0)^*, Q \in (\mathbf{V}_0 \cup \Pi_0)^+$

Table 4.1: Prefix conditions.

$\mathcal{L}$	Properties of R_0
K, D	general
T	general, reflexive
K4, D4	general, transitive
S4	general, reflexive, transitive

Table 4.2: Accessibility relations on prefixes.

accessibility relation of the modal logic under consideration, which is given in Tables 4.1 and 4.2 taken from [Wallen, 1990].

Before introducing the notion of modal substitutions we introduce the interpretation of nodes of type Π_0 and the *modal assignment for nodes of type* $\mathbf{V}_0$. Consider an $\mathcal{L}$-model M for some modal logic $\mathcal{L}$ with accessibility relation R and w a possible world. The interpretation of some node $Q \in \Pi_0$ depends on w and denotes the set of all worlds w' for which wRw' holds. Thus, $M_w(Q) = \{w' \mid wRw'\}$. A *modal assignment* ρ_M maps a world w and some node $Q \in \mathbf{V}_0$ to some world w' for which wRw' holds.

Assume an $\mathcal{L}$-model M, a possible world w, and a modal assignment ρ_M. The interpretation of some modal prefix p is a set of possible worlds that are described by the prefix.

Definition 4.1.7 (Semantics of Modal Prefixes) Let M be an $\mathcal{L}$-model, R the $\mathcal{L}$-accessibility relation, and let w be a possible world. An *interpretation* of a constant node $Q \in \Pi_0$ with respect to M_w is the set of all worlds w' for which wRw' holds. A *modal assignment* ρ_M is a mapping from $G \times \mathbf{V}_0$ into G, such that $wR\rho_M(w,Q)$ holds. The *interpretation of a modal prefix* p with respect to M, w, and some ρ_M is a set of possible worlds defined by:

$$M_w^{\rho_M}(p) = \begin{cases} \{w\} & \text{if } p = \langle\rangle \\ \bigcup_{w' \in M_w(Q)} M_{w'}^{\rho_M}(p') & \text{if } p = \langle Q :: p' \rangle \text{ and } Q \in \Pi_0 \\ M_{\rho_M(w,Q)}^{\rho_M}(p') & \text{if } p = \langle Q :: p' \rangle \text{ and } Q \in \mathbf{V}_0 \end{cases}$$

■

Lemma 4.1.8 (Accuracy of Definition 4.1.7) Let M be an $\mathcal{L}$-model, ρ_Q an assignment of object variables, ρ_M a modal assignment, and φ an $\mathcal{L}$-formula. For any polarity p and sequence mq of modal quantifiers from $\{\Box, \Diamond\}$ it holds:

$$M_w^{\rho_Q,\rho_M} \models_{\mathcal{L}} (mq(\varphi))^p \iff \text{for all } w' \in M_w^{\rho_M}(mp)\ M_{w'}^{\rho_Q,\rho_M} \models_{\mathcal{L}} \varphi^p$$

where mp is the modal prefix of φ obtained from mq, where each ν-type modal quantifier corresponds to some (new) variable node and each π-type modal quantifier to some new constant node. ■

Proof. We prove the statement by induction over the length n of mq:

$\mathbf{n = 0}$: In this case the statement trivially holds, since $M_w^{\rho_M}(\langle\rangle) = \{w\}$.
$\mathbf{n \to n+1}$: There are two cases to consider:

(A) The signed formula $(mq\varphi)^p$ is of the form $(\nu\{\nu,\pi\}^*\varphi)^p$: let $Q \in \mathbf{V}_0$ be the variable node obtained for the leading ν-type modal quantifier.

$$\begin{array}{rl} & M_w^{\rho_Q,\rho_M} \models_{\mathcal{L}} (\nu\{\nu,\pi\}^*\varphi)^p \\ \Leftrightarrow & \text{there exists } w', wRw', \text{ such that } M_{w'}^{\rho_Q,\rho_M} \models_{\mathcal{L}} (\{\nu,\pi\}^*\varphi)^p \\ \overset{IH}{\Leftrightarrow} & \text{there exists } w', wRw', \text{ such that for all } w'' \in M_{w'}^{\rho_M}(mp)\ M_{w''}^{\rho_Q,\rho_M} \models_{\mathcal{L}} \varphi^p \\ \overset{\rho_M(w,Q) \text{ is such a } w'}{\Leftrightarrow} & \text{for all } w'' \in M_w^{\rho_M}(\langle Q :: mp\rangle)\ M_{w''}^{\rho_Q,\rho_M} \models_{\mathcal{L}} \varphi^p \end{array}$$

(B) The signed formula is of the form $(\pi\{\nu,\pi\}^*\varphi)^p$: let $Q \in \Pi_0$ be the constant node obtained for the leading π-type modal quantifier.

$$\begin{array}{rl} & M_w^{\rho_Q,\rho_M} \models_{\mathcal{L}} (\pi\{\nu,\pi\}^*\varphi)^p \\ \Leftrightarrow & \text{for all } w', wRw' \text{ it holds } M_{w'}^{\rho_Q,\rho_M} \models_{\mathcal{L}} (\{\nu,\pi\}^*\varphi)^p \\ \overset{IH}{\Leftrightarrow} & \text{for all } w', wRw' \text{ it holds for all } w'' \in M_{w'}^{\rho_M}(mp)\ M_{w''}^{\rho_Q,\rho_M} \models_{\mathcal{L}} \varphi^p \\ \Leftrightarrow & \text{for all } w' \in M_w(Q) \text{ it holds for all } w'' \in M_{w'}^{\rho_M}(mp)\ M_{w''}^{\rho_Q,\rho_M} \models_{\mathcal{L}} \varphi^p \\ \Leftrightarrow & \text{for all } w'' \in M_w^{\rho_M}(\langle Q :: mp\rangle)\ M_{w''}^{\rho_Q,\rho_M} \models_{\mathcal{L}} \varphi^p \end{array}$$

□

The above semantics of modal prefixes allows the definition of *prefixed formulas* and their semantics, which is a convenient tool for many soundness and safeness arguments. A prefixed formula of a node in an indexed formula tree is of the form $mp.\varphi^p$ where mp is the modal prefix of Q, p the polarity of Q, and φ its label. Its semantics with respect to an $\mathcal{L}$-model M, a possible world w and assignment ρ is the valuation of φ^p in the possible worlds described by $M_w^{\rho}(mp)$.

Definition 4.1.9 (Modal Substitution) Let Q be an indexed formula tree. A *modal substitution* is a mapping $\sigma_M : \mathbf{V}_0 \to (\mathbf{V}_0 \cup \Pi_0)^*$. The homomorphic extension of σ_M to sequences $p \in (\mathbf{V}_0 \cup \Pi_0)^*$ is defined by

$$\overline{\sigma}_M(p) := \begin{cases} \langle\rangle & \text{if } p = \langle\rangle \\ \sigma_M(Q) \oplus \overline{\sigma}_M(q) & \text{if } p = \langle Q :: q\rangle \text{ and } Q \in \mathbf{V}_0 \\ \langle Q :: \overline{\sigma}_M(q)\rangle & \text{if } p = \langle Q :: q\rangle \text{ and } Q \in \Pi_0 \end{cases}$$

where $\sigma_M(Q) \oplus \overline{\sigma}_M(q)$ denotes the concatenation of the two sequences $\sigma_M(Q)$ and $\overline{\sigma}_M(q)$. A modal substitution σ_M is idempotent, iff $\overline{\sigma}_M$ is idempotent, i.e. if $\overline{\sigma}_M(\overline{\sigma}_M(p)) = \overline{\sigma}_M(p)$. ■

Throughout the rest of this thesis we assume that all modal substitutions are idempotent. The domain of a modal substitution σ_M is the set $\mathbf{V}_0$ of nodes Q of secondary type ν_0, for which $\sigma_M(Q) \neq \langle Q\rangle$ holds; the domain of σ_M is denoted by $dom(\sigma_M)$. Analogously to substitutions of meta variables, a modal substitution induces an ordering among nodes (from [Wallen, 1990]):

Definition 4.1.10 (Modal Ordering) Let Q be an indexed formula tree and σ_M a modal substitution for Q. The *modal substitution* σ_M induces a binary relation $\prec_M \in \mathbf{V}_0 \times (\mathbf{V}_0 \cup \Pi_0)$ defined by: $Q_0 \prec_M Q_1$ iff $Q_0 \in \sigma_M(Q_1)$. ■

We have now all notions required for the definition of a substitution for each logic $\mathcal{L}$ considered in this thesis and its induced *reduction relation* $\lhd_{\mathcal{L}}$. We present a unified notion for substitutions, in order to stress the uniformity of the foundations laid in [Wallen, 1990, Miller, 1983, Pfenning, 1987] as a basis for the uniformity of the framework proposed in this thesis.

Definition 4.1.11 ($\mathcal{L}$-Substitution & Reduction-Relation $\lhd_{\mathcal{L}}$) Let Q be an indexed formula tree for some logic $\mathcal{L}$.

CPL: If $\mathcal{L}$ is classical propositonal logic (CPL), then there are no substitutions. The *reduction relation* $\lhd_{\mathcal{L}}$ is the transitive closure of $\prec_Q$, i.e. $\lhd_{\mathcal{L}} := \prec_Q^+$.

CPML: If $\mathcal{L}$ is classical propositional modal logic (CPML), then an $\mathcal{L}$-substitution consists of a *modal substitution* σ_M. The domain of such a substitution is $dom(\sigma_M)$. The *reduction relation* $\lhd_{\mathcal{L}}$ is the transitive closure of the union of $\prec_Q$ and $\prec_M$, i.e. $\lhd_{\mathcal{L}} := (\prec_Q \cup \prec_M)^+$.

CFOML: If $\mathcal{L}$ is classical first-order modal logic (CFOML), then an $\mathcal{L}$-substitution consists of a *variable substitution* σ_Q and a *modal substitution* σ_M, denoted by $\langle \sigma_Q, \sigma_M \rangle$. The domain of such a substitution is the pair $\langle dom(\sigma_Q), dom(\sigma_M) \rangle$. The *reduction relation* $\lhd_{\mathcal{L}}$ is the transitive closure of the union of $\prec_Q$, $\prec_V$, and $\prec_M$, i.e. $\lhd_{\mathcal{L}} := (\prec_Q \cup \prec_V \cup \prec_M)^+$.

CFOL & CHOL: If $\mathcal{L}$ is classical first-order or higher-order logic (CHOL), then an $\mathcal{L}$-substitution consists only of a *variable substitution* σ_Q. The domain of this substitution is $dom(\sigma_Q)$. The *reduction relation* $\lhd_{\mathcal{L}}$ is the transitive closure of the union of $\prec_Q$ and $\prec_V$, i.e. $\lhd_{\mathcal{L}} := (\prec_Q \cup \prec_V)^+$. ∎

Remark 4.1.12 Throughout the rest of this thesis we agree that if σ is some $\mathcal{L}$-substitution and Q is the binding node of some γ-variable X, then if $X \in dom(\sigma)$, we say that Q *is instantiated* and otherwise Q *is uninstantiated*. If Q is instantiated, then we may denote the instance $\sigma(X)$ also by $\sigma(Q)$. Similarly, if Q is some node of secondary type ν_0, then if $Q \in dom(\sigma)$ then Q *is instantiated* and otherwise Q *is uninstantiated*.

Based on the uniform notion of an $\mathcal{L}$-substitution, we can uniformly define when an $\mathcal{L}$-substitution is admissible with respect to the logic $\mathcal{L}$.

Definition 4.1.13 ($\mathcal{L}$-Admissible Substitutions) Let Q be an indexed formula tree for some logic $\mathcal{L}$, $\sigma_{\mathcal{L}}$ an $\mathcal{L}$-substitution, and $\lhd_{\mathcal{L}}$ the respective reduction relation. $\sigma_{\mathcal{L}}$ is $\mathcal{L}$-admissible, if and only if

CPL: If $\mathcal{L}$ is classical propositional logic, then $\lhd_{\mathcal{L}}$ ($:= \prec_Q^+$) must be irreflexive. This is always fulfilled by construction of the indexed formula tree.

CPML: If $\mathcal{L}$ is classical propositional modal logic, then $\sigma_{\mathcal{L}} := \sigma_M$ and it must hold

1. σ_M respects the $\mathcal{L}$-accessibility relation R_0 on $(\mathbf{V}_0 \cup \Pi_0)^*$; i.e. for all modal prefixes p, q in Q, $p\ R_0\ q$ implies $\overline{\sigma}_M(p)\ R_0\ \overline{\sigma}_M(q)$,
2. if $\mathcal{L}$ is a K-logic, then for any $Q \in \mathbf{V}_0$, such that $\sigma_M(Q) \neq \langle Q \rangle$, there must be a $Q' \in \Pi_0 \cup \mathbf{V}_0$ which occurs in $\sigma_M(Q)$,
3. and $\lhd_{\mathcal{L}} := (\prec_Q \cup \prec_M)^+$ must be irreflexive.

CFOML: If $\mathcal{L}$ is classical first-order modal logic, then $\sigma_{\mathcal{L}} := \langle \sigma_Q, \sigma_M \rangle$ and it must hold

1. σ_M respects the $\mathcal{L}$-accessibility relation R_0 as for CPML,

2. if $\mathcal{L}$ is a K-logic, then for any $Q \in \mathbf{V}_0$, such that $\sigma_M(Q) \neq \langle Q \rangle$, there must be a $Q' \in \Pi_0$ which occurs in $\sigma_M(Q)$,
3. and $\lhd_{\mathcal{L}} := (\prec_Q \cup \prec_V \cup \prec_M)^+$ must be irreflexive.

CFOL & CHOL: If $\mathcal{L}$ is classical first-order or higher-order logic, then $\lhd_{\mathcal{L}}$ ($:= (\prec_Q \cup \prec_V)^+$) must be irreflexive. ■

Note that the notion of admissibility of substitutions is equivalent to the notion defined in [Wallen, 1990] if substitutions σ are idempotent. With respect to [Miller, 1983, Pfenning, 1987] our notion corresponds to the *dependency relation* among the instances of γ-variables.

Lemma 4.1.14 (Relationship between assignment and substitution) Let Q be an actual indexed formula tree, and let $\langle \sigma_Q, \sigma_M \rangle$ be an $\mathcal{L}$-admissible $\mathcal{L}$-substitution. Then for every $\mathcal{L}$-model M, every possible world w, assignment ρ_Q, and modal assignment ρ_M:

$$M_w^{\rho_Q,\rho_M}(\sigma_M(pre(Q)).\sigma_Q(\mathbf{Label}(Q))) \Longrightarrow M_w^{\rho'_Q,\rho'_M}(pre(Q).\mathbf{Label}(Q))$$

where $\rho'_Q := \rho_Q[x/M_w^{\rho_Q,\rho_M}(\sigma_Q(x)) \mid x \in dom(\sigma_Q)]$
$\rho'_M := \rho_M[(w,Q)/M_{w'}^{\rho_Q,\rho_M}(\sigma_M(Q)) \mid Q \in dom(\sigma_M)]$ for some $w' \in M_w^{\rho_Q,\rho_M}(\sigma_M(Q))$ ■

Proof. We assume that the statement holds for empty prefixes, i.e. $pre(Q) = \langle\rangle$ (this proof is simply first by structural induction over terms and then over the formula $\mathbf{Label}(Q)$*. The statement is proved by induction over the length n of $pre(Q)$.*

$\mathbf{n = 0}$:

$$\begin{array}{rl} & M_w^{\rho_Q,\rho_M}(\sigma_M(pre(Q)).\sigma_Q(\mathbf{Label}(Q))) \\ \Leftrightarrow & M_w^{\rho_Q,\rho_M}(\sigma_Q(\mathbf{Label}(Q))) \\ \Leftrightarrow & M_w^{\rho'_Q,\rho'_M}(\mathbf{Label}(Q)) \\ \Leftrightarrow & M_w^{\rho'_Q,\rho'_M}(pre(Q).\mathbf{Label}(Q)) \end{array}$$

$\mathbf{n \to n+1}$: *we have two cases to consider:*

(A). $pre(Q) = \langle Q' :: p \rangle$ and $Q' \in \mathbf{V}_0$:

$$\begin{array}{rl} & M_w^{\rho_Q,\rho_M}(\sigma_M(pre(Q)).\sigma_Q(\mathbf{Label}(Q))) \\ \Leftrightarrow & M_w^{\rho_Q,\rho_M}(\sigma_M(\langle Q' :: p \rangle).\sigma_Q(\mathbf{Label}(Q))) \\ \Leftrightarrow & M_w^{\rho_Q,\rho_M}(\sigma_M(Q') \oplus \sigma_M(p).\sigma_Q(\mathbf{Label}(Q))) \\ \overset{Lemma\ 4.1.8}{\Leftrightarrow} & \text{for all } w' \in M_w^{\rho_Q,\rho_M}(\sigma_M(Q')) \;.\; M_{w'}^{\rho_Q,\rho_M}(\sigma_M(p).\sigma_Q(\mathbf{Label}(Q))) \\ \overset{IH}{\Leftrightarrow} & \text{for all } w' \in M_w^{\rho_Q,\rho_M}(\sigma_M(Q')) \;.\; M_{w'}^{\rho'_Q,\rho'_M}(p.\mathbf{Label}(Q)) \\ \Rightarrow & M_w^{\rho'_Q,\rho''_M}(pre(Q).\mathbf{Label}(Q)) \end{array}$$

where $\rho''_M := \rho'_M[w''/(w,Q)]$, $w'' \in M_w^{\rho'_Q,\rho'_M}(\sigma_M(Q))$.

(B). $pre(Q) = \langle Q' :: p \rangle$ and $Q' \in \Pi_0$:

$$\begin{array}{rl} & M_w^{\rho_Q,\rho_M}(\sigma_M(pre(Q)).\sigma_Q(\mathbf{Label}(Q))) \\ \Leftrightarrow & M_w^{\rho_Q,\rho_M}(\sigma_M(\langle Q' :: p \rangle).\sigma_Q(\mathbf{Label}(Q))) \\ \Leftrightarrow & M_w^{\rho_Q,\rho_M}(Q' :: \sigma_M(p).\sigma_Q(\mathbf{Label}(Q))) \\ \overset{Lemma\ 4.1.8}{\Leftrightarrow} & \text{for all } w', wRw' \text{ it holds} \;.\; M_{w'}^{\rho_Q,\rho_M}(\sigma_M(p).\sigma_Q(\mathbf{Label}(Q))) \\ \overset{IH}{\Leftrightarrow} & \text{for all } w', wRw' \text{ it holds} \;.\; M_{w'}^{\rho'_Q,\rho'_M}(p.\mathbf{Label}(Q)) \\ \Rightarrow & M_w^{\rho'_Q,\rho'_M}(\langle Q'' :: p \rangle.\mathbf{Label}(Q)) \end{array}$$

□

Following [Wallen, 1990, Andrews, 1981] we define *(horizontal) paths* on indexed formula trees.

Definition 4.1.15 (Paths) Let Q be an indexed formula tree. Then a *path* in Q is a sequence $\ll Q_1, \ldots, Q_n \gg$ of α-related nodes in Q. The sets $\mathcal{P}(Q)$ of paths through Q is the smallest set containing $\{\ll Q \gg\}$ and which is closed under the following operations:

α-Decomposition: If Q' is a node of primary type α and subtrees Q_1, Q_2, and $P \cup \{\ll \Gamma, Q' \gg\} \in \mathcal{P}(Q)$, then $P \cup \{\ll \Gamma, Q_1, Q_2 \gg\} \in \mathcal{P}(Q)$.

β-Decomposition: If Q' is a node of primary type β and subtrees Q_1, Q_2, and $P \cup \{\ll \Gamma, Q' \gg\} \in \mathcal{P}(Q)$, then both $P \cup \{\ll \Gamma, Q_1 \gg\} \in \mathcal{P}(Q)$ and $P \cup \{\ll \Gamma, Q_2 \gg\} \in \mathcal{P}(Q)$.

γ-Decomposition: If Q' is a node of primary type γ and subtrees $Q_1, \ldots, Q_n$, and $P \cup \{\ll \Gamma, Q' \gg\} \in \mathcal{P}(Q)$, then $P \cup \{\ll \Gamma, Q_1, \ldots, Q_n \gg\} \in \mathcal{P}(Q)$.

δ-Decomposition: If Q' is a node of primary type δ and subtree Q', and $P \cup \{\ll \Gamma, Q' \gg\} \in \mathcal{P}(Q)$, then $P \cup \{\ll \Gamma, Q' \gg\} \in \mathcal{P}(Q)$.

ν-Decomposition: If Q' is a node of primary type ν and subtrees $Q_1, \ldots, Q_n$, and $P \cup \{\ll \Gamma, Q' \gg\} \in \mathcal{P}(Q)$, then $P \cup \{\ll \Gamma, Q_1, \ldots, Q_n \gg\} \in \mathcal{P}(Q)$.

π-Decomposition: If Q' is a node of primary type π and subtree Q'', and $P \cup \{\ll \Gamma, Q' \gg\} \in \mathcal{P}(Q)$, then $P \cup \{\ll \Gamma, Q'' \gg\} \in \mathcal{P}(Q)$. ■

Note the close relationship between the decomposition rules for paths for the different types of nodes and the relationship between the satisfiability of the main formulas with respect to its constituent formulas: a path containing an α-type node is replaced by a path containing both subnodes, while a path containing a β-type node is decomposed into two paths each containing one of the subnodes. Analogously the decomposition of γ-, δ-, ν- and π-type nodes corresponds to the relationship between the satisfiability of the signed formula of the respective type to its constituent formulas. This relationship together with the side conditions imposed by the requirements (1) that a modal substitution respects the $\mathcal{L}$-accessibility relation and (2) the acyclicity of the overall ordering $\lhd_{\mathcal{L}}$ induced by the $\mathcal{L}$-substitution entail that whenever we have obtained a set of paths where all paths are unsatisfiable, then the initial conjecture is $\mathcal{L}$-valid. Indeed, if all paths are unsatisfiable, we can apply the model satisfiability rules backwards from the constituent formulas to the main formulas and obtain that there is no $\mathcal{L}$-model M satisfying φ^+, i.e. $\forall M \,.\, M \not\models_{\mathcal{L}} \varphi^+$, and thus it holds by definition $\forall M \,.\, M \models_{\mathcal{L}} \varphi$. Hence φ is $\mathcal{L}$-valid.

Analysing the decomposition rules and the respective satisfiability relations in more detail, we observe that they are equivalence transformations which is necessary for our soundness and safeness results. Safeness means intuitively that no possible refutation is lost by such a transformation. The decomposition rules have this property and this allows to switch freely between the granularity of the decomposition of paths. However, the definition of further transformations on indexed formula trees requires a weaker property which only requires the preservation of satisfiable paths during the transformation, which is the general condition for *soundness*. Formally:

Definition 4.1.16 (Soundness & Safeness) Let Q, Q' be two indexed formula trees with respective $\mathcal{L}$-admissible substitutions σ and σ', where Q' has been obtained from Q by some transformation. The transformation is *sound* if, and only if, if there is an $\mathcal{L}$-satisfiable path Q then so there is in Q'. The transformation is *safe* if, and only if, if there is an $\mathcal{L}$-satisfiable path in Q' then so there is one in Q. ■

Indexed formula trees are obtained from an initial indexed formula tree by five additional rules: introduction of Leibniz' equality for equations and equivalences (Section 4.2), a rule for functional and boolean extensionality introduction (Section 4.3), a rule to expand positive equivalences into a conjunction of implications (Section 4.4), a substitution rule (Section 4.5), a rule to introduce cuts (Section 4.7), and a rule to increase the multiplicity of nodes of primary type γ or ν (Section 4.11).

4.2 Leibniz' Equality

The first rule expands an ε- or ζ-node into Leibniz' equality. This rule changes a given indexed formula tree by replacing the respective ε- or ζ-node Q_e (i.e. $\mathbf{Label}(Q_e) = \varepsilon(s,t)$ or $\mathbf{Label}(Q_e) = \zeta(s,t)$) by a so-called *Leibniz* node, which has the same label and polarity p as Q_e, is of primary type α, and has subtrees Q_e and an initial indexed formula tree for $(\forall P \,.\, P(s) \Rightarrow P(t))^p$.

Definition 4.2.1 (Leibniz' Equality Introduction) Let Q_e be a leaf node in some indexed formula of polarity p, uniform type $e \in \{\varepsilon, \zeta\}$, such that $\mathbf{Label}(Q_e) = \varepsilon(s_\tau, t_\tau)$ or $\mathbf{Label}(Q_e) = \zeta(s_\tau, t_\tau)$. Further let Q_L be an initial indexed formula tree for the signed formula $(\forall P_{\tau\to o} \,.\, P(s) \Rightarrow P(t))^p$. Then we can replace Q_e by

$$Q_{Leibniz} = \begin{array}{c} \mathbf{Label}(Q_e)^p_\alpha \\ \diagup \quad \diagdown \\ Q_e \qquad Q_L \end{array}$$

We call the new node a *Leibniz* node. ∎

Example 4.2.2 We illustrate the Leibniz' equality introduction rule with our running example from Example 4.1.2 (p. 33). The application of the rule to the ε-type subtree $0 + x = x^-_\varepsilon$ transforms the indexed formula tree from Figure 4.1 into the indexed formula tree in Figure 4.2.

Lemma 4.2.3 (Soundness & Safeness of Leibniz' Equality Introduction Rule) The Leibniz' equality introduction rule on indexed formula trees from Definition 4.2.1 is sound and safe. ∎

Proof. The rule operates on literal nodes. Since literal nodes are never of type γ, δ, ν, or π, no quantifiers with defined polarity are duplicated by that operation and thus the operation does not affect the overall substitution σ which remains $\mathcal{L}$-admissible. It remains to be shown for soundness (resp. safeness) that it also preserves the existence (resp. absence) of satisfiable paths. To this end we consider the paths affected by these transformations.

Let $Q = \overset{\varepsilon}{\underset{\zeta}{}}(s,t)$ be the ε- or ζ-type literal node on which the rule is applied. This rule transforms the paths as follows

$$\begin{array}{lll} \ll \Gamma, \varepsilon^-(s,t) \gg & \text{into} & \ll \Gamma, \varepsilon^-(s,t), \gamma^- P \,.\, \beta^-(P(s)^-, P(t)^+) \gg \text{ and} \\ \ll \Gamma, \zeta^+(s,t) \gg & \text{into} & \ll \Gamma, \zeta^+(s,t), \delta^+ P \,.\, \alpha^+(P(s)^+, P(t)^-) \gg. \end{array}$$

We show that for both transformations the former path is satisfiable if, and only if, the latter is also satisfiable. For the negative case, i.e. for $\varepsilon(s,t)$, we prove it when $\varepsilon(s,t)$ is $s = t^-$. The proof for

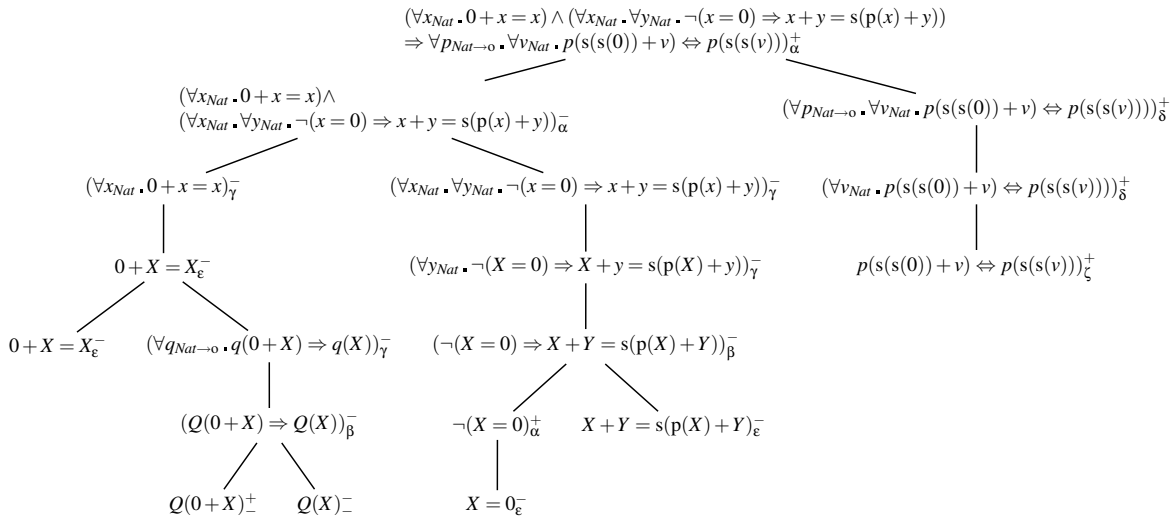


Figure 4.2: Indexed formula tree after introduction of Leibniz' equality.

$(s\Leftrightarrow t)^{-}$ *is analogous.*

$$
\begin{array}{ll}
 & M^{\rho}_{w}\models_{\mathcal{L}} s=t^{-}\\
\Leftrightarrow & M^{\rho}_{w}\models_{\mathcal{L}} s=t\\
\Leftrightarrow & M^{\rho}_{w}\models_{\mathcal{L}} s=t \text{ and forall } p\in\mathcal{D}_{\tau\to o}\ M^{\rho[p/P]}_{w}\models_{\mathcal{L}} P(s) \text{ or } M^{\rho}_{w}\not\models_{\mathcal{L}} P(s)\\
\Leftrightarrow & M^{\rho}_{w}\models_{\mathcal{L}} s=t \text{ and forall } p\in\mathcal{D}_{\tau\to o}\ M^{\rho[p/P]}_{w}\models_{\mathcal{L}} P(s) \text{ or } M^{\rho}_{w}\not\models_{\mathcal{L}} P(t)\\
\Leftrightarrow & M^{\rho}_{w}\models_{\mathcal{L}} s=t \text{ and forall } p\in\mathcal{D}_{\tau\to o}\ M^{\rho[p/P]}_{w}\models_{\mathcal{L}} P(s)^{-} \text{ or } M^{\rho}_{w}\models_{\mathcal{L}} P(t)^{+}\\
\Leftrightarrow & M^{\rho}_{w}\models_{\mathcal{L}} s=t \text{ and forall } p\in\mathcal{D}_{\tau\to o}\ M^{\rho[p/P]}_{w}\models_{\mathcal{L}} \beta^{-}(P(s)^{-},P(t)^{+})\\
\Leftrightarrow & M^{\rho}_{w}\models_{\mathcal{L}} s=t^{-} \text{ and } M^{\rho}_{w}\models_{\mathcal{L}} \gamma^{-}P\,.\,\beta^{-}(P(s)^{-},P(t)^{+})
\end{array}
$$

For the positive case $\zeta^{+}(s,t))$ *we prove it when* $\zeta^{+}(s,t)$ *is* $s=t^{+}$*. The proof for* $(s\Leftrightarrow t)^{+}$ *is analogous.*

$$
\begin{array}{ll}
 & M^{\rho}_{w}\models_{\mathcal{L}} s=t^{+}\\
\Leftrightarrow & M^{\rho}_{w}\not\models_{\mathcal{L}} s=t\\
\Leftrightarrow & M^{\rho}_{w}\not\models_{\mathcal{L}} s=t \text{ and } M^{\rho}_{w}\models_{\mathcal{L}} s=s \text{ and } M^{\rho}_{w}\not\models_{\mathcal{L}} s=t\\
\Leftrightarrow & M^{\rho}_{w}\not\models_{\mathcal{L}} s=t \text{ and } M^{\rho}_{w}\models_{\mathcal{L}} (\lambda x\,.\,s=x)s \text{ and } M^{\rho}_{w}\not\models_{\mathcal{L}} \lambda x\,.\,(s=x)t\\
\Leftrightarrow & M^{\rho}_{w}\not\models_{\mathcal{L}} s=t \text{ and there exists a } p\in\mathcal{D}_{\tau\to o}\ (p=M^{\rho}_{w}(\lambda x\,.\,(s=x)))\\
 & M^{\rho[p/P]}_{w}\models_{\mathcal{L}} P(s) \text{ and } M^{\rho[p/P]}_{w}\not\models_{\mathcal{L}} P(t)\\
\Leftrightarrow & M^{\rho}_{w}\not\models_{\mathcal{L}} s=t \text{ and there exists a } p\in\mathcal{D}_{\tau\to o}\ (p=M^{\rho}_{w}(\lambda x\,.\,(s=x)))\\
 & M^{\rho[p/P]}_{w}\models_{\mathcal{L}} P(s)^{-} \text{ and } M^{\rho[p/P]}_{w}\models_{\mathcal{L}} P(t)^{+}\\
\Leftrightarrow & M^{\rho}_{w}\not\models_{\mathcal{L}} s=t \text{ and there exists a } p\in\mathcal{D}_{\tau\to o}\ (p=M^{\rho}_{w}(\lambda x\,.\,(s=x)))\\
 & M^{\rho[p/P]}_{w}\models_{\mathcal{L}} \alpha^{+}(P(s)^{-},P(t)^{+})\\
\Leftrightarrow & M^{\rho}_{w}\models_{\mathcal{L}} s=t^{+} \text{ and } M^{\rho}_{w}\models_{\mathcal{L}} \delta^{+}P\,.\,\alpha^{+}(P(s)^{-},P(t)^{+})
\end{array}
$$

□

4.3 Extensionality

The second rule deals with the functional and boolean extensionality of ε and ζ-formulas, and the respective introduction rule can be sketched as follows:

$$\frac{\overset{\varepsilon}{\zeta}(s(x),t(x))}{\overset{\varepsilon}{\zeta}(\lambda x\,.\,s(x),\lambda x\,.\,t(x))}\ \textit{Ext-I}$$

For this rule we need some additional restrictions to ensure its soundness: if we have an ε-formula $\varepsilon(s(X),t(X))$, then intuitively we can only abstract over meta variables bound in some γ-type node and moreover it must be possible to move its quantifier in front of $\varepsilon(s(X),t(X))$. However, in order to simplify the theorem proving process we refrain from forcing to actually have to move the quantifiers in front of the ε-formula and rather define a sufficient condition ensuring that the quantifier could *in principle* be moved in front of the ε-formula. For ε-formulas $\varepsilon(s(X),t(X))$ with an occurrence of some γ-variable X the condition expresses that the γ-*variable* X *is local for* $\varepsilon(s(X),t(X))$. Analogously, ζ-formulas can only be abstracted over variables bound in δ-nodes (i.e. parameters) and must also be "local". We formalise these conditions by defining the notion of a local variable for ε and ζ formulas (cf. Definition 4.3.1), and use this as the condition for the rule.

Definition 4.3.1 (Local Variables) Given an indexed formula tree Q and a node Q' inside Q whose label contains a free variable x. If x is bound in some γ-type node in Q, then x is γ-*local for* Q' if, and only if, Q' is the binding position for x or Q has a direct parent node Q'' such that

- Q'' is of primary type β, x does not occur in the label of the sibling of Q', and x is γ-local for Q'', or
- Q' is of either primary type α or γ and x is γ-local for Q''.

The dual property of a δ-local variable x for some Q' is defined for variables bound in a δ-type node and holds if, and only if, Q' is the binding node for x or Q' has a direct parent node Q'' such that

- Q'' is of type α, x does not occur in the label of the sibling of Q', and x is δ-local for Q'', or
- Q'' is of either type β or δ and x is δ-local for Q''. ∎

Before presenting the extensionality introduction rule we prove a property about the local variables that is used in the soundness and safeness proof of the extensionality introduction rule.

Lemma 4.3.2 (Local Variables are Local) Let $Q = \varphi(x')^p_t$ be a subtree of an indexed formula tree and Q_x the parent node of the binding node of x'. If x' is γ-local to Q, then the label of Q_x is of the form $\gamma^{p'}x\,.\,\Psi(\varphi(x)^p_t)$ and there is a Ψ' such that for every $\mathcal{L}$-model M we have:

$$\text{(i)}\ \ M^\rho_w \models_{\mathcal{L}} \gamma^{p'}x\,.\,\Psi(\varphi(x)^p_t) \Leftrightarrow M^\rho_w \models_{\mathcal{L}} (\Psi'(\gamma^p x\,.\,\varphi(x)^p_t))^{p'}$$

If x is δ-local to Q, then the label of Q_x is of the form $\delta^{p'}x\,.\,\Psi(\varphi(x)^p_t)$ and there is a Ψ' such that for every $\mathcal{L}$-model M:

$$\text{(ii)}\ \ M^\rho_w \models_{\mathcal{L}} \delta^{p'}x\,.\,\Psi(\varphi(x)^p_t) \Leftrightarrow M^\rho_w \models_{\mathcal{L}} (\Psi'(\delta^p x\,.\,\varphi(x)^p_t))^{p'}$$

∎

Proof. The proof of (i) is by induction over the distance from Q to Q_x. The proof of (ii) is similar.

Base Case: *For the base case we have the situation that Q_x is the parent node of Q and has the label $\gamma^p x \,.\, \varphi(x)$. Hence the statement holds trivially where Ψ' is $\lambda x_o \,.\, x$.*

Induction Step: *Assume the statement holds for the parent node Q_p of Q. We proceed by case analysis over the primary type ut of Q_p:*

1. *$ut = \beta$, the label of the parent node Q_p is $\beta^{p_0}(\varphi(x)^p, \psi^{p_1})$ and x does not occur in ψ. Then by induction hypothesis there is a Ψ'' such that it holds:*

$$M^\rho_w \models_\mathcal{L} \gamma^{p'} \Psi \,.\, (\beta^{p_0}(\varphi(x)^p, \psi^{p_1})) \overset{IH}{\Leftrightarrow} M^\rho_w \models_\mathcal{L} \Psi''(\gamma^{p_0} x \,.\, \beta^{p_0}(\varphi(x)^p, \psi^{p_1}))$$

Now x is still local to $\varphi(x)^p$ and it remains to prove that

$$M^\rho_w \models_\mathcal{L} \gamma^{p_0} x \,.\, \beta^{p_0}(\varphi(x)^p, \psi^{p_1}) \Leftrightarrow M^\rho_w \models_\mathcal{L} \beta^{p_0}(\gamma^{p_0} x \,.\, \varphi(x)^p, \psi^{p_1})$$

$$\begin{array}{ll} & M^\rho_w \models_\mathcal{L} \gamma^{p_0} x \,.\, \beta^{p_0}(\varphi(x)^p, \psi^{p_1}) \\ \Leftrightarrow & \text{for all } a \in \mathcal{D}_\tau \text{ it holds } M^{\rho[a/x]}_w \models_\mathcal{L} \beta^{p_0}(\varphi(x)^p, \psi^{p_1}) \\ \Leftrightarrow & \text{for all } a \in \mathcal{D}_\tau \text{ it holds } M^{\rho[a/x]}_w \models_\mathcal{L} \varphi(x)^p \text{ or } M^{\rho[a/x]}_w \models_\mathcal{L} \psi^{p_1} \\ \overset{x \notin \psi}{\Leftrightarrow} & \text{for all } a \in \mathcal{D}_\tau \text{ it holds } M^{\rho[a/x]}_w \models_\mathcal{L} \varphi(x)^p \text{ or } M^\rho_w \models_\mathcal{L} \psi^{p_1} \\ \Leftrightarrow & (\text{for all } a \in \mathcal{D}_\tau \text{ it holds } M^{\rho[a/x]}_w \models_\mathcal{L} \varphi(x)^p) \text{ or } M^\rho_w \models_\mathcal{L} \psi^{p_1} \\ \Leftrightarrow & M^\rho_w \models_\mathcal{L} \gamma^p x \,.\, \varphi(x)^p \text{ or } M^\rho_w \models_\mathcal{L} \psi^{p_1} \\ \Leftrightarrow & M^\rho_w \models_\mathcal{L} \beta^{p_0}(\gamma^{p_0} x \,.\, \varphi(x)^p, \psi^{p_1}) \end{array}$$

The Ψ' for this case is then $\lambda F_o \,.\, \Psi''(\beta^{p_0}(F, \psi^{p_1}))$.

2. *$ut = \alpha$, the label of the parent node Q_p is $\alpha^{p_0}(\varphi(x)^p, \psi^{p_1})$. Then by induction hypothesis there is a Ψ'' such that it holds:*

$$M^\rho_w \models_\mathcal{L} \gamma^{p'} \Psi \,.\, (\alpha^{p_0}(\varphi(x)^p, \psi^{p_1})) \overset{IH}{\Leftrightarrow} M^\rho_w \models_\mathcal{L} \Psi''(\gamma^{p_0} x \,.\, \alpha^{p_0}(\varphi(x)^p, \psi^{p_1}))$$

Now x is still local to $\varphi(x)^p$ and it remains to prove

$$M^\rho_w \models_\mathcal{L} \gamma^{p_0} x \,.\, \alpha^{p_0}(\varphi(x)^p, \psi^{p_1}) \Leftrightarrow M^\rho_w \models_\mathcal{L} \alpha^{p_0}(\gamma^{p_0} x \,.\, \varphi(x)^p, \psi^{p_1})$$

$$\begin{array}{ll} & M^\rho_w \models_\mathcal{L} \gamma^{p_0} x \,.\, \alpha^{p_0}(\varphi(x)^p, \psi^{p_1}) \\ \Leftrightarrow & \text{for all } a \in \mathcal{D}_\tau \text{ it holds } M^{\rho[a/x]}_w \models_\mathcal{L} \alpha^{p_0}(\varphi(x)^p, \psi^{p_1}) \\ \Leftrightarrow & \text{for all } a \in \mathcal{D}_\tau \text{ it holds } M^{\rho[a/x]}_w \models_\mathcal{L} \varphi(x)^p \text{ and } M^{\rho[a/x]}_w \models_\mathcal{L} \psi^{p_1} \\ \overset{x \notin \psi}{\Leftrightarrow} & \text{for all } a \in \mathcal{D}_\tau \text{ it holds } M^{\rho[a/x]}_w \models_\mathcal{L} \varphi(x)^p \text{ and} \\ & \text{for all } a \in \mathcal{D}_\tau \text{ it holds } M^{\rho[a/x]}_w M^{\rho[a/x]}_w \models_\mathcal{L} \psi^{p_1} \\ \Leftrightarrow & M^\rho_w \models_\mathcal{L} \gamma^p x \,.\, \varphi(x)^p \text{ and } M^\rho_w \models_\mathcal{L} \gamma^{p_1} x \,.\, \psi^{p_1} \\ \Leftrightarrow & M^\rho_w \models_\mathcal{L} \alpha^{p_0}(\gamma^{p_0} x \,.\, \varphi(x)^p, \gamma^{p_1} x \,.\, \psi^{p_1}) \end{array}$$

The Ψ' for this case is then $\lambda F_o \,.\, \Psi''(\alpha^{p_0}(F, \gamma^{p_1} x \,.\, \psi^{p_1}))$.

3. *$ut = \gamma$ and the label of the parent node Q_p is $\gamma^{p_0} y \,.\, \varphi(x)^p$, $x \neq y$. Then by induction hypothesis there exists a Ψ'' such that it holds:*

$$M^{\rho}_w \models_{\mathcal{L}} \gamma^{p'} x \,.\, \Psi(\gamma^{p_0} y \,.\, (\varphi(x)^p)) \overset{IH}{\Leftrightarrow} M^{\rho}_w \models_{\mathcal{L}} \Psi'(\gamma^{p'} x \,.\, \gamma^{p_0} y \,.\, \varphi(x)^p)$$

Now x is still local to $\varphi(x)^p$ and it remains to prove

$$M^{\rho}_w \models_{\mathcal{L}} \gamma^{p'} x \,.\, \gamma^{p_0} y \,.\, \varphi(x)^p \Leftrightarrow M^{\rho}_w \models_{\mathcal{L}} \gamma^{p_0} y \,.\, \gamma^{p'} x \,.\, \varphi(x)^p$$

$$\begin{array}{ll} & M^{\rho}_w \models_{\mathcal{L}} \gamma^{p'} x \,.\, \gamma^{p_0} y \,.\, \varphi(x)^p \\ \overset{x \neq y}{\Leftrightarrow} & \text{for all } a,b \in \mathcal{D}_\tau \text{ it holds } M^{\rho[a/x,b/y]}_w \models_{\mathcal{L}} \varphi(x)^p \\ \Leftrightarrow & M^{\rho}_w \models_{\mathcal{L}} \gamma^{p_0} y \,.\, \gamma^{p} x \,.\, \varphi(x)^p \end{array}$$

The Ψ' in this case is $\lambda F_{\mathrm{o}} \,.\, \Psi''(\gamma^{p_0} y \,.\, F)$.

4. *Otherwise x is not local to Q.* □

The extensionality introduction rule changes a given indexed formula tree by replacing an ε or ζ-node Q_e by a so-called *Extensionality-Introduction* node, which has the same label and polarity p than Q_e, is of primary type α and with subtrees Q_e and an initial indexed formula tree for $(\lambda x \,.\, s(x) = \lambda x \,.\, t(x))^p$. Of course, *Ext-I* is only applicable if x is *local for* Q_e.

Definition 4.3.3 (Extensionality Introduction) Let Q_e be a leaf in some indexed formula of polarity p, uniform type $e \in \{\varepsilon, \zeta\}$, such that $\mathbf{Label}(Q_e) = \overset{\xi}{\xi}(s,t)$. Let further be x a variable that is *local for* Q_e, and Q_{Ext} be an initial indexed formula tree for the signed formula $(\lambda x \,.\, s = \lambda x \,.\, t)^p$. Then we can replace Q_e by

$$Q_{Ext-I} = \begin{array}{c} \mathbf{Label}(Q_e)^p_\alpha \\ \diagup \quad \diagdown \\ Q_e \qquad Q_{Ext} \end{array}$$

We call the new node an *Extensionality introduction* node. ■

Example 4.3.4 We illustrate the extensionality rule with our running example from Example 4.1.2 (p. 33). The application of the rule with the γ-local variable y to the ε-type subtree $x + y = \mathrm{s}(\mathrm{p}(x) + y)^-_\varepsilon$ transforms the indexed formula tree from Figure 4.1 into the indexed formula tree in Figure 4.3.

Lemma 4.3.5 (Soundness & Safeness of Extensionality Introduction) The Extensionality-Introduction rule on indexed formula trees from Definition 4.3.3 is sound and safe. ■

Proof. The rule operates on literal nodes and thus does not affect the substitution which remains $\mathcal{L}$-admissible. It remains to be shown for soundness (resp. safeness) that it preserves the existence (resp. absence) of satisfiable paths. To this end we consider the paths affected by these transformations.

Let $Q = \overset{\zeta}{\varepsilon}(s(x),t(x))$ be the ε- or ζ-type literal node on which the rule is applied and x the λ-abstracted free variable that is local to Q (cf. Definition 4.3.1). This rule transforms the paths as follows

$$\begin{array}{lll} \ll \Gamma, \varepsilon(s(x),t(x)) \gg & \textit{into} & \ll \Gamma, \varepsilon(s(x),t(x)), \varepsilon(\lambda x \,.\, s(x), \lambda x \,.\, t(x)) \gg \textit{ and} \\ \ll \Gamma, \zeta(s(x),t(x)) \gg & \textit{into} & \ll \Gamma, \zeta(s(x),t(x)), \zeta(\lambda x \,.\, s(x), \lambda x \,.\, t(x)) \gg. \end{array}$$

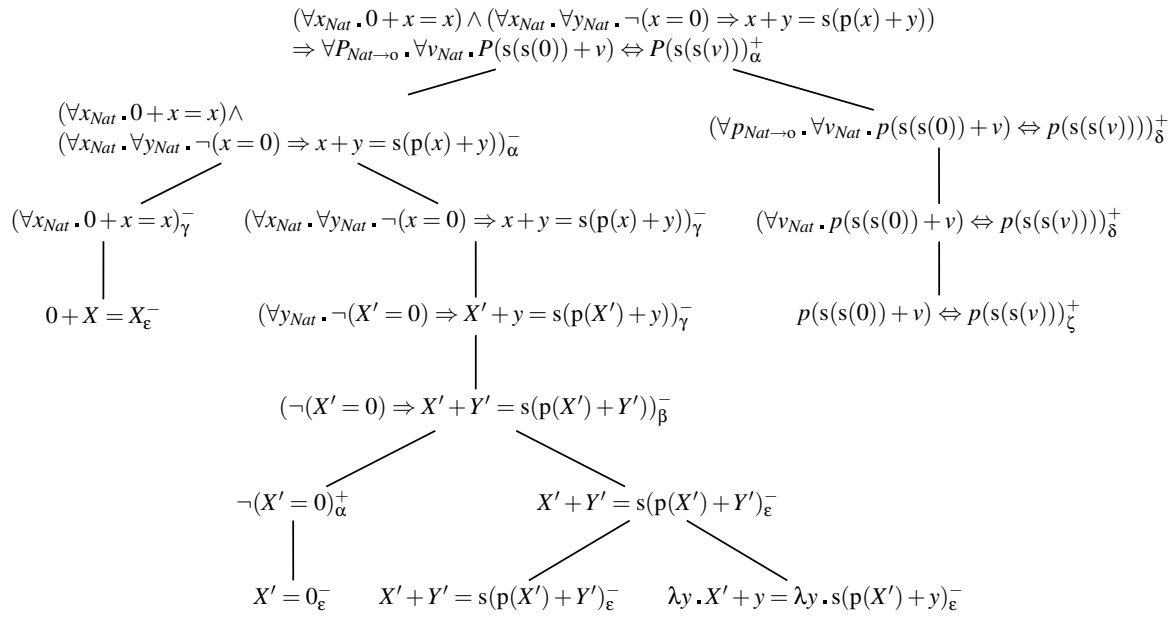


Figure 4.3: Indexed formula tree after extensionality introduction.

---

where in the first case x is γ-local to Q and in the second case x is δ-local to Q.

We show that for both transformations the former path is satisfiable if, and only if, the latter is also satisfiable. For the negative case we prove it when $\varepsilon(s(x),t(x))$ is $s(x)=t(x)^{-}$. The proof for the case when $\varepsilon(s(x),t(x))$ is $s(x)\Leftrightarrow t(x)^{-}$ is analogous. Note however, that for $s(x)\Leftrightarrow t(x)^{-}$ we introduce the ε-type signed formula $(\lambda x\,.\,s(x)=\lambda x\,.\,t(x))^{-}$.

$$
\begin{array}{lll}
M^{\rho}_{w}\models_{\mathcal{L}} s(x)=t(x)^{-} & \Leftrightarrow & M^{\rho}_{w}\models_{\mathcal{L}} s(x)=t(x) \\
 & \Leftrightarrow & M^{\rho}_{w}\models_{\mathcal{L}} s(x)=t(x) \text{ and } M^{\rho}_{w}\models_{\mathcal{L}} s(x)=t(x) \\
 & \overset{x \text{ is } \gamma\text{-local}}{\Leftrightarrow} & M^{\rho}_{w}\models_{\mathcal{L}} s(x)=t(x) \text{ and for all } a\in\mathcal{D}_{\tau}\ M^{\rho[a/x]}_{w}\models_{\mathcal{L}} s(x)=t(x) \\
 & \overset{\text{Extensionality}}{\Leftrightarrow} & M^{\rho}_{w}\models_{\mathcal{L}} s(x)=t(x) \text{ and } M^{\rho}_{w}\models_{\mathcal{L}} \lambda x\,.\,s(x)=\lambda x\,.\,t(x) \\
 & \Leftrightarrow & M^{\rho}_{w}\models_{\mathcal{L}} s(x)=t(x)^{-} \text{ and } M^{\rho}_{w}\models_{\mathcal{L}} \lambda x\,.\,s(x)=\lambda x\,.\,t(x)^{-}
\end{array}
$$

For the positive case we prove it when $\zeta(s(x),t(x))$ is $s(x)=t(x)^{+}$. The proof for the case when $\zeta(s(x),t(x))$ is $s(x)\Leftrightarrow t(x)^{+}$ is analogous. Again, note that for $s(x)\Leftrightarrow t(x)^{+}$ we introduce the ζ-type signed formula $(\lambda x\,.\,s(x)=\lambda x\,.\,t(x))^{+}$.

$$
\begin{array}{lll}
 & & M^{\rho}_{w}\models_{\mathcal{L}} s(x)=t(x)^{+} \\
 & \Leftrightarrow & M^{\rho}_{w}\not\models_{\mathcal{L}} s(x)=t(x) \\
 & \Leftrightarrow & M^{\rho}_{w}\not\models_{\mathcal{L}} s(x)=t(x) \text{ and } M^{\rho}_{w}\not\models_{\mathcal{L}} s(x)=t(x) \\
 & \overset{x \text{ is } \delta\text{-local}}{\Leftrightarrow} & M^{\rho}_{w}\not\models_{\mathcal{L}} s(x)=t(x) \text{ and there is an } a\in\mathcal{D}_{\tau}\ M^{\rho[a/x]}_{w}\not\models_{\mathcal{L}} s(x)=t(x) \\
 & \Leftrightarrow & M^{\rho}_{w}\not\models_{\mathcal{L}} s(x)=t(x) \text{ and not for all } a\in\mathcal{D}_{\tau}\ M^{\rho[a/x]}_{w}\models_{\mathcal{L}} s(x)=t(x) \\
 & \overset{\text{Extensionality}}{\Leftrightarrow} & M^{\rho}_{w}\not\models_{\mathcal{L}} s(x)=t(x) \text{ and not } M^{\rho}_{w}\models_{\mathcal{L}} \lambda x\,.\,s(x)=\lambda x\,.\,t(x) \\
 & \Leftrightarrow & M^{\rho}_{w}\not\models_{\mathcal{L}} s(x)=t(x) \text{ and } M^{\rho}_{w}\not\models_{\mathcal{L}} \lambda x\,.\,s(x)=\lambda x\,.\,t(x) \\
 & \Leftrightarrow & M^{\rho}_{w}\models_{\mathcal{L}} s(x)=t(x)^{+} \text{ and } M^{\rho}_{w}\models_{\mathcal{L}} \lambda x\,.\,s(x)=\lambda x\,.\,t(x)^{+}
\end{array}
$$

□

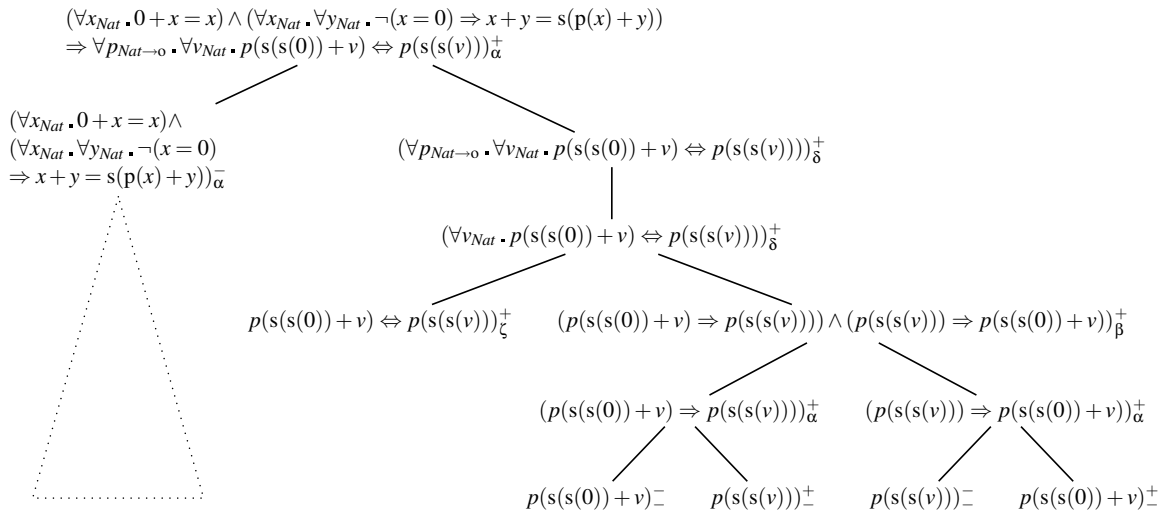


Figure 4.4: Indexed formula tree after boolean ζ-expansion.

4.4 Boolean ζ-Expansion

The third rule consists of the expansion of positive equations and equivalences $\zeta(A,B)$ over formulas A_o and B_o into $((A\Rightarrow B)\wedge(B\Rightarrow A))^{+}$. The rule replaces the respective ζ-type node Q_ζ of label $\zeta(A,B)$ with a so-called ζ-*expansion node* of the same label and polarity than Q_ζ, but of primary type α and with the subtrees Q_ζ and an initial indexed formula tree for the signed formula $((A\Rightarrow B)\wedge(B\Rightarrow A))^{+}$.

Definition 4.4.1 (Boolean ζ-Expansion) Let Q_ζ be a leaf node in some indexed formula tree of positive polarity, uniform type ζ, and label $\zeta(A,B)$, where A and B are of type o. Let further be Q_E be an initial indexed formula tree for the signed formula $((A\Rightarrow B)\wedge(B\Rightarrow A))^{+}$. Then we can replace Q_ζ by

$$Q_{\zeta\text{Expansion}} = \begin{array}{ccc} & \mathbf{Label}(Q_\zeta)^{+}_{\alpha} & \\ \swarrow & & \searrow \\ Q_\zeta & & Q_E \end{array}$$

We call the new node a ζ-*expansion node*. ■

Note that this rule corresponds to the *b*-rule in [Benzmüller *et al*, 2002b].

Example 4.4.2 We illustrate the boolean ζ-expansion rule with our running example from Example 4.1.2 (p. 33). The application of the rule on the ζ-type subtree $p(\mathrm{s}(\mathrm{s}(0))+v)\Leftrightarrow p(\mathrm{s}(\mathrm{s}(v)))^{+}_{\zeta}$ transforms the indexed formula tree from Figure 4.1 into the indexed formula tree in Figure 4.4.

We now prove the soundness and safeness of the boolean ζ-expansion rule.

Lemma 4.4.3 (Soundness & Safeness of Boolean ζ-Expansion) The boolean ζ-Expansion rule on indexed formula tree is sound and safe. ■

Proof.

The rule operates on literal nodes and thus does not affect the substitution which remains $\mathcal{L}$-admissible. It remains to be shown for soundness (respectively safeness) that it preserves also

the existence (respectively absence) of satisfiable paths. The rule is applied to a literal node $Q_\zeta = \zeta(A,B)$*, which is of positive polarity, where A and B are formulas. The new node introduced by that rule denotes the signed formula* $((A \Rightarrow B) \wedge (B \Rightarrow A))^+$*, which using uniform notation is* $\beta(\alpha(A^-,B^+),\alpha(B^-,A^+))$*. The rule transforms the paths as follows:*

$$\ll \Gamma, \zeta(A,B) \gg \quad \textit{into} \quad \ll \Gamma, \zeta(A,B), \beta(\alpha(A^-,B^+)^+, \alpha(B^-,A^+)^+)^+ \gg$$

We show that the former path is satisfiable if, and only if, the latter path is satisfiable.

$$\begin{array}{rcll}
M^\rho_w \models_{\mathcal{L}} \zeta(A_o, B_o) & \Leftrightarrow & M^\rho_w \not\models_{\mathcal{L}} A = B & \\
& \Leftrightarrow & M^\rho_w(A) \neq M^\rho_w(B) & \\
& \Leftrightarrow & (M^\rho_w(A) = \top \text{ and } M^\rho_w(B) = \bot) \text{ or} & \\
& & (M^\rho_w(A) = \bot \text{ and } M^\rho_w(B) = \top) \qquad \text{as} \quad M^\rho_w(o) = \{\top, \bot\} & \\
& \Leftrightarrow & (M^\rho_w \models_{\mathcal{L}} A \text{ and } M^\rho_w \not\models_{\mathcal{L}} B) \text{ or} & \\
& & (M^\rho_w \not\models_{\mathcal{L}} A \text{ and } M^\rho_w \models_{\mathcal{L}} B) & \\
& \Leftrightarrow & (M^\rho_w \models_{\mathcal{L}} A^- \text{ and } M^\rho_w \models_{\mathcal{L}} B^+) \text{ or} & \\
& & (M^\rho_w \models_{\mathcal{L}} A^+ \text{ and } M^\rho_w \models_{\mathcal{L}} B^-) & \\
& \Leftrightarrow & M^\rho_w \models_{\mathcal{L}} \alpha(A^-,B^+)^+ \text{ or } M^\rho_w \models_{\mathcal{L}} \alpha(B^-,A^+)^+ & \\
& \Leftrightarrow & M^\rho_w \models_{\mathcal{L}} \beta(\alpha(A^-,B^+)^+, \alpha(B^-,A^+)^+)^+ &
\end{array}$$

□

4.5 Substitutions

The fourth rule is the application of admissible $\mathcal{L}$-substitutions to indexed formula trees. Given an indexed formula tree Q and an admissible $\mathcal{L}$-substitution σ, we can apply a new $\mathcal{L}$-substitution σ' if and only if $dom(\sigma) \cap dom(\sigma') = \emptyset$ and $\sigma' \circ \sigma$ is an $\mathcal{L}$-admissible substitution. If σ' is applicable and if σ' contains a variable substitution, we must apply it to all subtrees of Q. This is required especially for the higher-order logic case when instantiating *set variables*, i.e. variables of type $\tau \to o$ where τ is arbitrary.

Definition 4.5.1 (Instantiation of Indexed Formula Trees) Let Q be an indexed formula tree and X a γ-variable that occurs free in $\mathbf{Label}(Q)$. The instantiation of X by t in Q is defined as

- if Q is a leaf node, then we replace Q by an initial indexed formula tree for $\{t/X\}(\mathbf{Label}(Q))$.
- Otherwise, apply $\{t/X\}$ to the label of Q, and recursively apply it to the subnodes of Q. ■

Definition 4.5.2 ($\mathcal{L}$-Substitution Application on Indexed Formula Trees) Let Q be an indexed formula tree, σ its actual $\mathcal{L}$-substitution, and σ' a new substitution. If σ' is applicable on Q with σ, then we apply σ' to Q. The result of the substitution application is the (instantiated) indexed formula tree together with the new substitution $\sigma' \circ \sigma$. ■

Example 4.5.3 Take as an example the indexed formula tree for the positive formula

$$(\forall p_{\iota\to o} \,.\, \forall q_{\iota\to o} \,.\, \exists r_{\iota\to o} \,.\, \forall x_\iota \,.\, (p(x) \vee q(x)) \Rightarrow r(x))^+.$$

The initial indexed formula tree is viewed on the left-hand side of Figure 4.5 and the actual substitution is the empty substitution. Instantiation of the γ-variable $R_{\iota\to o}$ with $\lambda y_\iota \,.\, p(y) \vee q(y)$ results in the

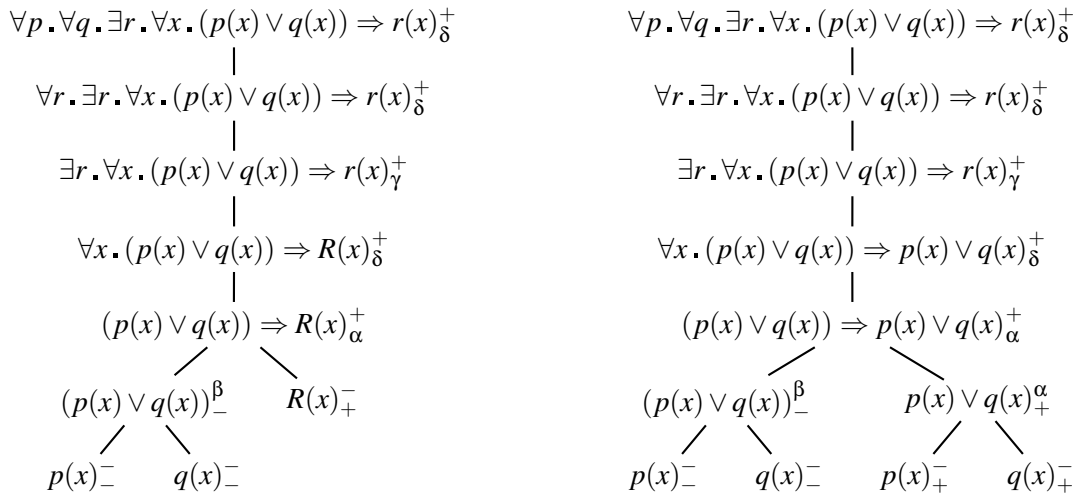


Figure 4.5: Example substitution application on indexed formula trees.

indexed formula tree on the right-hand side of Figure 4.5. Note how the leaf node $R(x)^{-}_{+}$ in the initial indexed formula tree is replaced by an initial indexed formula tree for the positive $\beta\eta$-normalised formula $(\{\lambda y_{\iota}\,.\,P(y)\vee Q(y)/R\}R(x)))^{+}$.

Lemma 4.5.4 The application of an $\mathcal{L}$-substitution is sound. ∎

Proof. *The new overall substitution is $\mathcal{L}$-admissible, i.e. the modal substitution respects the $\mathcal{L}$-accessibility relation and the overall ordering is $\lhd_{\mathcal{L}}$ is irreflexive. It remains to be shown that if there was an $\mathcal{L}$-satisfiable path before instantiation, then so there is afterwards. To this end consider a subtree Q of polarity p and label φ that has been affected by the instantiation. If Q was a literal node then it has been replaced by a subtree of label $\sigma'(\varphi)$; otherwise the label of Q has been replaced by $\sigma'(\varphi)$. Thus, in any case the prefixed formula $pre(Q).\varphi^{p}$ has been replaced by the prefixed formula $\sigma'(pre(Q)).\sigma'(\varphi)^{p}$. Assume Q was on an $\mathcal{L}$-satisfiable path, then there exists an $\mathcal{L}$-model M such that for all w and ρ it holds $M^{\rho}_{w}\models_{\mathcal{L}} pre(Q).\varphi^{p}$. On the other hand from Lemma 4.1.14 there exists an ρ' for σ', such that for all possible worlds w holds $M^{\rho'}_{w}(pre(Q).\varphi^{p}) = M^{\rho''}_{w}(\sigma'(pre(Q)).\sigma'(\varphi)^{p})$. Thus, if Q was $\mathcal{L}$-satisfiable before, then $\sigma(Q)$ still is.* □

Obtaining Substitutions. Finding a substitution is essential for automating proof search. For the instantiation rule two kinds of substitutions must be determined: one for γ-variables in order to instantiate the labels of the nodes and one for node variables in order to adjust the modal prefix of nodes.

A standard procedure for finding object variable substitutions is a unification procedure which computes for two terms t_1, t_2 a substitution σ_Q such that $\sigma_Q(t_1)$ and $\sigma_Q(t_2)$ are syntactically equal. For the CoRe framework we use the higher-order unification procedure from [Snyder & Gallier, 1989]. The procedure generates a list of substitutions σ each possibly accompanied by a set of flex-flex constraints of the form $H(t_1,\ldots,t_n) = G(s_1,\ldots,s_m)$, where H and G are higher-order variables. Only one of these substitutions and flex-flex constraints can be applied. However, undoing of substitution application is supported in CoRe which enables backtraking over unifiers.

The check for the admissibility of the substitutions is deferred to the acyclicity check of the new overall substitution. In case the actual logic is CHOL two additional problems must be tackled: first, the higher-order unification procedure may generate new variables, for which there are no binding nodes in the actual indexed formula tree. Second, the flex-flex constraints must be taken into account as additional conditions.

A variable H generated by higher-order unification is a γ-variable, but it is unclear how the binding node for H structurally relates to existing binding nodes. The exact location for the binding of H can only be determined when H itself gets instantiated since its location depends on the δ-variables that occur in H's instantiation. This instantiation being unknown at the time H is generated requires a "lazy" mechanism to introduce a binding node for H that is structurally independent of the existing quantifiers and only becomes related when H is instantiated. This mechanism is defined in Section 4.6.

The flex-flex constraints generated by higher-order unification are constraints that still need to be proved in order to equalise two formulas or terms. The main problem consists of inserting these constraints into the indexed formula tree at appropriate nodes. This depends on their role and we define in Section 4.9 a general technique to integrate flex-flex constraints.

The second part of an $\mathcal{L}$-substitution is a modal substitution. In [Otten & Kreitz, 1996] the idea of a unification procedure for terms is carried over to (modal) prefixes of indexed formula trees called the *T-string unification* procedure. The T-string unification procedure is parameterized over a set of unification rules that are specific for each modal logic. Each instantiation of the unification procedure has the property that it generates only substitutions that respect the $\mathcal{L}$-accessibility relation of the respective modal logic. Thus, the unification procedure already deals with the first conditions for admissible modal substitutions (cf. the cases **CPML** and **CFOML** in Definition 4.1.13). The check of the other conditions is deferred to the instantiation rule.

4.6 Binding Generated Variables

As discussed in the previous section we want to support the introduction of new variables, e.g. those generated by higher-order unification. They act like meta-variables, i.e. the same substitution restrictions apply to them as for any other γ-variable, and we want to treat them alike.

Assume Q is the root node of an indexed formula tree with label φ and polarity p and H_τ is a new variable, i.e. that is not bound in Q. To integrate a binding position for H we create an initial indexed formula tree Q' for the signed formula $(\forall H_\tau \,.\, True)^-$. Then Q' is of the form

$$\begin{array}{c} (\forall H \,.\, True)^-_\gamma \\ | \\ True^-_\_ \end{array}$$

and the node $True^-_\_$ is the binding node for H. Then we connect Q and Q' by some new α-type node of polarity p, and label $\alpha^p(\varphi^p, (\forall H \,.\, True)^-)$.

Definition 4.6.1 (Insertion of new variables) Let Q be an indexed formula tree, H_τ a variable not bound in Q, and let Q' be an initial indexed formula tree for $(\forall H_\tau \,.\, True)^-$. *Inserting a binding position for* H_τ transforms Q into

$$\begin{array}{c} \alpha^p(\mathbf{Label}(Q)^p, (\forall H \,.\, True)^-) \\ \diagup \quad \diagdown \\ Q \qquad Q' \end{array}$$

■

Lemma 4.6.2 (Soundness & Safeness of Insertion of new variables) The insertion of binding positions for new variables is sound and safe. ■

Proof. Since the substitution is not affected by the insertion of a new variable, the overall ordering $\lhd_{\mathcal{L}}$ remains irreflexive. It remains to prove that the former indexed formula tree is $\mathcal{L}$-satisfiable if, and only if, the latter indexed formula tree is $\mathcal{L}$-satisfiable.

Let M be an $\mathcal{L}$-model , w a possible world, and ρ an assignment. Then:

$$\begin{aligned}
& M^{\rho}_{w} \models_{\mathcal{L}} \alpha^{p}(\mathbf{Label}(Q)^{p}, (\forall H \,.\, True)^{-}) \\
\Leftrightarrow\ & M^{\rho}_{w} \models_{\mathcal{L}} \mathbf{Label}(Q)^{p} \text{ and } M^{\rho}_{w} \models_{\mathcal{L}} (\forall H \,.\, True)^{-} \\
\Leftrightarrow\ & M^{\rho}_{w} \models_{\mathcal{L}} \mathbf{Label}(Q)^{p} \text{ and } M^{\rho}_{w} \models_{\mathcal{L}} \forall H \,.\, True \\
\Leftrightarrow\ & M^{\rho}_{w} \models_{\mathcal{L}} \mathbf{Label}(Q)^{p} \text{ and for all } a \,.\, M^{\rho[a/x]}_{w} \models_{\mathcal{L}} True \\
\Leftrightarrow\ & M^{\rho}_{w} \models_{\mathcal{L}} \mathbf{Label}(Q)^{p}
\end{aligned}$$

□

4.7 Cut

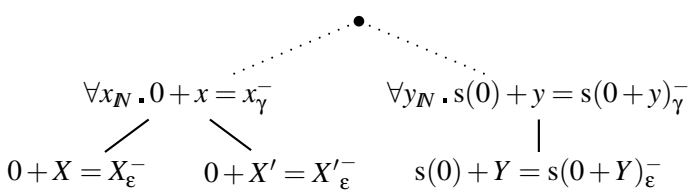
As usual the cut rule can be used in an arbitrary formula φ with free variables at some arbitrary subtree Q' of the whole indexed formula tree. We introduce the cut on Q' by α-relating the (signed) β-formula $\beta^{-}(\varphi^{+}, \varphi^{-})$ to Q'. For the free variables in φ we additionally universally quantify those that are not bound on some parent node of Q' in order to obtain a valid indexed formula tree. Thus, if the free γ- and δ-variables in φ and not bound above Q' are $\vec{x}$, let $\vec{x'}$ be variables of the same type that are new with respect to Q. Then there is a renaming ρ from $\vec{x}$ to $\vec{x'}$. The actual cut-formula that is α-inserted on Q' is then the negative formula $(\forall \vec{x'} \,.\, \rho(\beta^{-}(\varphi^{+}, \varphi^{-})))^{-}$ (respectively $(\Box \forall \vec{x'} \,.\, \rho(\beta^{-}(\varphi^{+}, \varphi^{-})))^{-}$ in case the actual logic $\mathcal{L}$ is a modal logic). In order to "restore" the old variables $\vec{x}$ in that formula, we instantiate the $\vec{x'}$ afterwards using ρ^{-1}.

Definition 4.7.1 (Cut) Let Q be an indexed formula tree, Q' a subtree of Q, φ an $\mathcal{L}$-formula with free γ- and δ-variables $\vec{x}$ that are not bound above Q' in Q, $\vec{x'}$ new variables, and $\rho = \{\vec{x'}/\vec{x}\}$. The *insertion of a cut over* φ consists in creating an initial indexed formula tree Q_c for the negative formula $\psi = (\forall \vec{x'} \,.\, \rho(\varphi \Rightarrow \varphi))^{-}$ (respectively $(\Box \forall \vec{x'} \,.\, \rho(\varphi \Rightarrow \varphi))^{-}$ for modal logics), α-relate Q_c to Q' and apply the substitution ρ^{-1}.

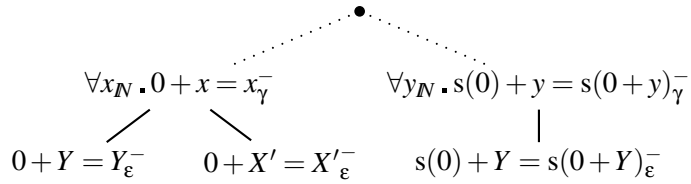
$$\begin{array}{c} \alpha^{p}(\psi, \mathbf{Label}(Q)^{p}) \\ \diagup \quad \diagdown \\ Q_c \qquad Q \end{array}$$

The new actual $\mathcal{L}$-substitution is then $\rho^{-1} \circ \sigma$. ■

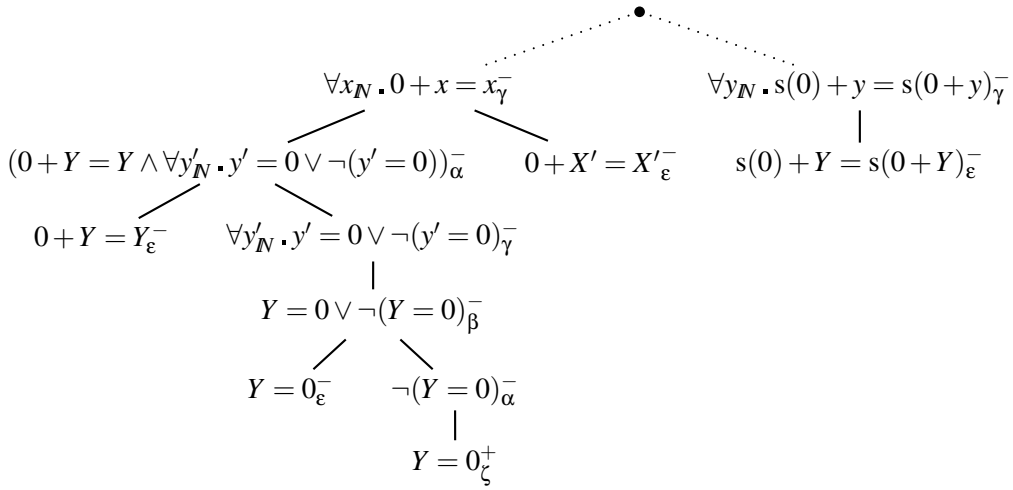
Example 4.7.2 We illustrate the cut rule for indexed formula trees with the following axioms about addition over natural numbers: $\forall x_{\mathbb{N}} \,.\, 0 + x = x$ and $\forall y_{\mathbb{N}} \,.\, \mathrm{s}(0) + y = \mathrm{s}(0 + y)$. The axioms occur negatively in some larger formula and we initially assign a multiplicity of 2 for the γ-quantifier of $x_{\mathbb{N}}$. The initial indexed formula tree is then as follows:



Assume further we instantiate X with Y which results in the indexed formula tree



Note that the actual substitution is now $\sigma := \{Y/X\}$. Finally, we perform a cut on $0+Y=Y^-_\varepsilon$ over the formula $Y=0$. To that end we consider the variables in that formula, which are not bound above $0+Y=Y^-_\varepsilon$. The only such variable is Y and thus we create the cut formula $\forall y'_{\mathbb{N}} . y'=0 \vee \neg(y'=0)$ and obtain the renaming $\rho := \{Y'/Y\}$ where Y' is the meta-variable introduced for y' during the creation of an initial indexed formula tree for the cut formula. Inserting the cut and applying ρ^{-1} results in the indexed formula tree



The new overall substitution is $\rho^{-1} \circ \sigma = \{Y/X, Y/Y'\}$.

Lemma 4.7.3 (Soundness & Safeness of Cut) The insertion of cut over some formula φ is sound and safe. ∎

Proof. Let M be an $\mathcal{L}$-model. To prove soundness and safeness we have to show that (1) the overall ordering remains irreflexive and (2) M satisfies the signed formula $\mathbf{Label}(Q')^p$ *if, and only if, it satisfies the signed formula* $\alpha^p(\psi^-, \mathbf{Label}(Q')^p)$.

1. *Assume the ordering $\lhd_{\mathcal{L}}$ before rule application is irreflexive. To prove the irreflexivity of the resulting ordering, observe that the substitution ρ^{-1} does not introduce dependencies inside Q_c. The only dependencies introduced by ρ^{-1} are going from adjacent nodes of Q_c into Q_c. Furthermore, all parent nodes of Q_c are smaller with respect to $\prec_Q$. Thus, if there would be a cycle after insertion of the Cut, then there must have been a cycle before its insertion, which contradicts the assumption.*

2. *For the second part we prove the statement only for the modal logic case, as it contains the other*

case. Thus, $\mathbf{Label}(Q') = (\Box\forall\vec{x'}.\rho(\varphi \Rightarrow \varphi))^-$.

$$
\begin{array}{ll}
 & M_w^\rho \models_{\mathcal{L}} \alpha^p(\psi^-, \mathbf{Label}(Q')^p) \\
\Leftrightarrow & M_w^\rho \models_{\mathcal{L}} \psi^- \text{ and } M_w^\rho \models_{\mathcal{L}} \mathbf{Label}(Q')^p \\
\Leftrightarrow & \text{for all } v, wRv, \text{ for all } a_1,\ldots,a_k \,.\, M_v^{\rho[x_i'/a_i]} \models_{\mathcal{L}} (\varphi \Rightarrow \varphi)^- \\
 & \text{and } M_w^\rho \models_{\mathcal{L}} \mathbf{Label}(Q')^p \\
\Leftrightarrow & \text{for all } v, wRv, \text{ for all } a_1,\ldots,a_k \; M_v^{\rho[x_i'/a_i]} \models_{\mathcal{L}} \varphi^+ \text{ or } M_v^{\rho[x_i'/a_i]} \models_{\mathcal{L}} \varphi^- \\
 & \text{and } M_w^\rho \models_{\mathcal{L}} \mathbf{Label}(Q')^p \\
\Leftrightarrow & M_w^\rho \models_{\mathcal{L}} \mathbf{Label}(Q')^p
\end{array}
$$

□

Admissibility of Cut. The cut rule is admissible for all logics considered in this thesis, except for the considered higher-order logic. There it is required to prove the admissibility of the extensionality rule from [Pfenning, 1987] (see Section 4.9). Its non-admissibility for higher-order logic with Henkin semantics is further supported by the results from [Benzmüller *et al*, 2002b]. We will discuss the non-admissibility of the cut rule for the higher-order logic case in more detail in Section 5.5.

4.8 Connections and $\mathcal{L}$-Unsatisfiable Paths

A *connection* is a link between two nodes in Q, which are α-related with opposite polarities, have the same label and the same modal prefix with respect to the actual $\mathcal{L}$-substitution σ.

Definition 4.8.1 (Connections) Let Q be an indexed formula tree. A connection is a pair (Q', Q'') of nodes with the same label, the same modal prefix, and opposite polarities, such that there is a path $\ll \Gamma, Q', Q'' \gg$ in some $P \in \mathcal{P}(Q)$. ■

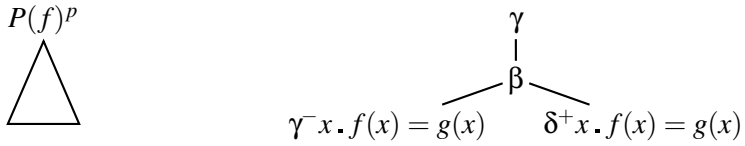
Note that we allow for connections between non-leaf nodes. A set $\mathcal{C}$ of connections is *spanning* for Q and σ, if there is a set P of paths $\mathcal{P}(Q)$, such that each path in P either contains a connection from $\mathcal{C}$, or contains $True^+$, $False^-$ or $t = t^+$. In this case each path in P is said to be $\mathcal{L}$-unsatisfiable with respect to σ, which is defined as follows:

Definition 4.8.2 ($\mathcal{L}$-Unsatisfiable Paths) Let Q be an indexed formula tree, σ an actual $\mathcal{L}$-admissible substitution for Q and $p \in P$ ($P \in \mathcal{P}(Q)$) a path through Q. The path p *is $\mathcal{L}$-unsatisfiable* if p contains either a positive node with label *True*, a negative node with label *False*, a positive node with label $t = t$, or two nodes that form a connection. ■

4.9 Cut Rule Applications

In this section we present two specific ways the cut rule from Section 4.7 is used, namely (i) that the extensionality rule from [Pfenning, 1987] is admissible and (ii) how flex-flex constraints that arise from higher-order unification can be integrated into an indexed formula tree.

The Extensionality Rule from [Pfenning, 1987] is Admissible. The extensionality rule defined for indexed formula trees in this thesis differs from the extensionality rule in [Pfenning, 1987]. In order to reduce the completeness proof of our calculus to the completeness proof in [Pfenning, 1987], we present how the extensionality rule from [Pfenning, 1987] can be simulated by combining our extensionality rule with the cut rule.

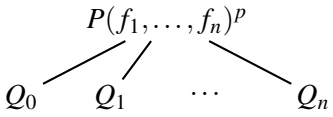


(a) Original indexed formula tree (b) Cut indexed formula tree

Figure 4.6:

Lemma 4.9.1 The Extensionality rule from [Pfenning, 1987] can be simulated by a combination of the cut rule, the extensionality introduction rule and Leibniz' expansion rule. ∎

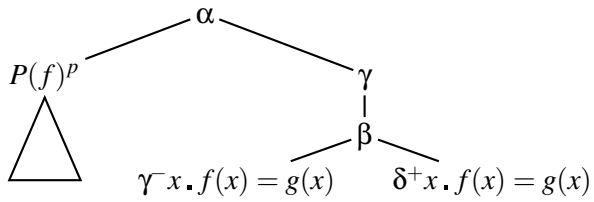
Proof. The extensionality rule from [Pfenning, 1987] is



where Q_0 is an indexed formula tree for $P(g_1,\ldots,g_n)^p$, and the Q_i are indexed formula trees for $\delta^p x_1,\ldots,x_n \,.\, f(x_1,\ldots,x_n) = g(x_1,\ldots,x_n)$. Without loss of generality we can use for our proof a rule where $n = 1$, since every application of the above simultaneous extensionality rule can be simulated by n applications of the unary extensionality rule.

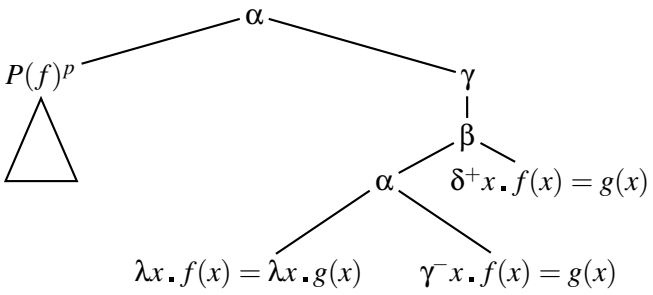
The proof sketch is as follows: we show that the cut-rule can be used to introduce the required statement about $\delta^p x \,.\, f(x) = g(x)$. This can be used to derive $\zeta^p(f,g)$ with the extensionality introduction rule, and subsequently expanded into the Leibniz equality to obtain $P(g)^p$ from $P(f)^p$.

The shape of the whole indexed formula tree before extensionality introduction is showed on the left-hand side of Figure 4.6: Q is the whole indexed formula tree (not shown) and the subtree on which we want to apply the extensionality rule is the subtree $P(f)^p$ (left-hand side of Figure 4.6). To simulate the extensionality rule from [Pfenning, 1987] we perform a cut over the formula $\forall x \,.\, f(x) = g(x)$. The cut corresponds to the indexed formula tree viewed on the right-hand side of Figure 4.6 and its insertion results in the following indexed formula tree:

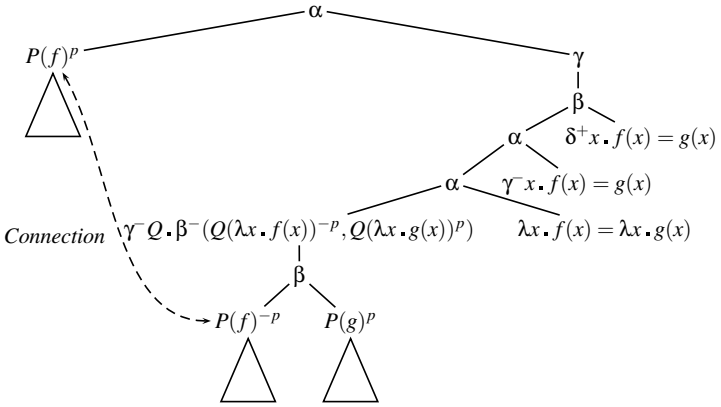


The positive occurrence of $\forall x \,.\, f(x) = g(x)$, i.e. $\delta^+ x \,.\, f(x) = g(x)$ represents the additional proof obligation to prove the cut-formula. The negative occurrence $\gamma^- x \,.\, f(x) = g(x)$ contains the ε-equation that is actually used to perform the extensionality proof step. Applying the extensionality introduction

rule of Definition 4.3.3 to $\gamma^{-}x\,.\,f(x) = g(x)$ *changes the whole indexed formula tree into*



Now we apply the Leibniz' equality introduction rule on the ε*-equation* $\lambda x\,.\,f(x) = \lambda x\,.\,g(x)$ *to obtain a subtree for* $\gamma^{-}Q\,.\,\beta^{-}(Q(\lambda x\,.\,f(x))^{-p}, Q(\lambda x\,.\,g(x))^{p})$*. After instantiation of the new* γ*-variable* Q *with* $\lambda h\,.\,P(h)$ *and* βη*-normalisation we obtain the indexed formula tree*



Finally we can draw a connection between the subtrees $P(f)^p$ *and* $P(f)^{-p}$ *to obtain the two new subgoals* $P(g)^p$ *and* $\delta^{+}x\,.\,f(x) = g(x)$*, which corresponds to the subgoals we would get when applying the extensionality rule from [Pfenning, 1987].* □

Inserting Flex-Flex Constraints. Flex-flex constraints are equations of the form $H(\vec{s}) = G(\vec{t})$ that result from higher-order unification. They represent additional constraints that need to be solved in order to make two formulas equal. Thus there is a need to adequately represent these additional goals in an indexed formula tree Q. However, the "exact" position for the constraints depends on its purpose, i.e. on the subtree from which a connection is to be introduced. This is addressed when we define the actual CORE reasoning rules. In this paragraph we show only how a flex-flex constraint can be attached to an arbitrary subtree Q' of Q by using the cut-rule. Assume $\vec{x}$ is the list of all γ- and δ-variables that are free in $H(\vec{s}) = G(\vec{t})$ and not bound on some parent node of Q' in Q and ρ a renaming of these variables into variables $\vec{x'}$ that are new with respect to Q. Then we introduce the flex-flex constraint by performing a cut over $H(\vec{s}) = G(\vec{t})$, which α-inserts an initial indexed formula tree for the negative formula $\forall\vec{x'}\,.\,\rho((H(\vec{s}) = G(\vec{t})) \Rightarrow (H(\vec{s}) = G(\vec{t})))$ on Q' and results in

$$\alpha^{p}(\gamma^{-}\vec{x'}\,.\,\rho((H(\vec{s}) = G(\vec{t})) \Rightarrow (H(\vec{s}) = G(\vec{t})))^{-}, \mathbf{Label}(Q'))$$

$$Q' \qquad Q_c$$

Since only those free variables are bound now that have not been bound above Q', the resulting indexed formula tree is again valid. After application of the cut-rule we apply the inverse substitution ρ^{-1} on the indexed formula tree. This is an admissible instantiation since ρ^{-1} cannot introduce cycles into the reduction ordering because we have only bound those variables in the constraints that are not bound above Q'. Hence the resulting indexed formula tree concisely represents the additional subgoal.

4.10 Soundness and Completeness

For any $\mathcal{L}$-valid formula φ there is an indexed formula tree for φ and an $\mathcal{L}$-substitution σ, which admits a spanning set of connections, this is the main result from [Andrews, 1981, Miller, 1983, Pfenning, 1987, Wallen, 1990] about proof search with indexed formula trees.

Theorem 4.10.1 (Soundness & Completeness of Indexed Formula Tree proofs) Let φ be a formula with respect to one of the logics $\mathcal{L}$ of Definition 3.2.1. φ is $\mathcal{L}$-valid if, and only if, we construct for φ an indexed formula tree using the rules from Definitions 4.1.1, 4.2.1, 4.3.3, 4.4.1, 4.5.2, and 4.7.1 which admits a spanning set of connections with respect to an $\mathcal{L}$-admissible substitution. ■

Proof. Follows for each of the considered logics from [Wallen, 1990], except for higher-order logic, where it follows from [Pfenning, 1987] and Lemma 4.9.1. The boolean ζ-expansion rule from Definition 4.4.1 is necessary in order to unfold positive equivalences, since equivalences are not treated in these approaches. There is no need for a negative variant of that rule, since for negative equivalences the expansion can be simulated by the Leibniz' equality introduction rule. □

4.11 Increase of Multiplicities

Proof search in indexed formula trees (cf. [Wallen, 1990, Pfenning, 1987, Andrews, 1981, Andrews, 1989]) proceeds by fixing the multiplicity of nodes of primary type γ and ν, and subsequently searching for an appropriate $\mathcal{L}$-admissible substitution and a spanning set of connections. However, setting the multiplicity beforehand is not adequate for an interactive proof search where multiplicities of nodes are determined on the fly. An example is the instantiation of some γ-variable x bound on some γ_0-type node Q of parent node Q_γ of label $\gamma^p y \,.\, \varphi(y)$: in order to design a complete proof procedure, we must be able to "copy" that meta variable before instantiating it. For the higher-order logic case the multiplicities can be adjusted during proof search by using the technique from [Issar, 1990]. It allows to increase the multiplicity of some γ-type node and localises its effect to some path for which no connection exists by copying and renaming that path adequately. However, that technique is not applicable in our context, since we have to copy the concerned indexed formula tree which prevents the localisation of the copying to some single path. Furthermore, adding a new initial indexed formula tree Q' for $\varphi(X')$ for some new meta variable X' and attaching it to Q, prevents to carry over all proof information onto Q': Leibniz' equality introductions and extensionality introductions are lost, and all connections that involved a subnode of Q are not present for subnodes in Q'. Finally, substitution information is not carried over onto Q', say when some γ-variable Z different from X is instantiated with a term containing X, then there is no copy of that Y which is instantiated with the equivalent term containing X'. Thus, all the proof information that was already established for Q must be redone for Q'. The same reasoning applies when "copying" ν_0-type node.

In the following we present a mechanism to constructively increase the multiplicity of some node which carries over all proof information to the new copy. First, note that the increase of multiplicity of some node may entail the increase of multiplicity of some other node. This is the case for example if

a meta variable to be copied occurs in the instance of some other meta variable. Intuitively we need a notion of self contained set of subtrees with respect to a substitution, in the sense that all γ-, δ-, ν-, and π-variables x that occur in the instance of some variable y, the set contains both the subtrees binding x and y.

We formalise this intuition but introducing the notion of a *convex set of subtrees with respect to some $\mathcal{L}$-admissible substitution* σ.

Definition 4.11.1 (Convex Set of Subtrees) Let Q be an indexed formula tree with $\mathcal{L}$-admissible substitution σ, $\mathcal{K}$ a set of independent[1] subtrees of Q.

Then $\mathcal{K}$ is *convex with respect to* σ if, and only if, for all $Q \in \mathcal{K}$ and for all γ-, δ-, ν-, π-variables x bound in Q we have: if x occurs in some instance $\sigma(y)$ for some y, then there exists some $Q' \in \mathcal{K}$ in which y is bound. ■

A trivial example for such a set is the set that consists of the whole indexed formula tree Q.

A convex set $\mathcal{K}$ of subtrees for an actual indexed formula tree Q has the property that it is not smaller with respect to $\lhd_{\mathcal{L}}$ than any other part of Q which is not in $\mathcal{K}$. In other words, there is no γ- or ν-variable bound outside $\mathcal{K}$ that is instantiated with some variable bound in $\mathcal{K}$. Consider the restriction $\sigma_{\mathcal{K}}$ of σ to those γ- and ν-variables that are bound in $\mathcal{K}$. Copying the subtrees in $\mathcal{K}$ yields a new set of subtrees $\mathcal{K}'$ and a renaming ρ of all[2] variables in $\mathcal{K}$. Then the renamed substitution $\sigma_{\mathcal{K}'} := \{\rho(\sigma_{\mathcal{K}}(x))/\rho(x) \mid x \in dom(\sigma_{\mathcal{K}})\}$ does not introduce any additional dependencies to parts not in $\mathcal{K}'$. Furthermore, if the original substitution was $\mathcal{L}$-admissible, then so is $\sigma_{\mathcal{K}'} \circ \sigma$. In the following lemma we formalise this observation:

Lemma 4.11.2 (Maximality of Convex Sets of Subtrees) Let Q be an indexed formula tree with $\mathcal{L}$-admissible substitution σ, $\mathcal{K}$ a convex set of subtrees of Q. For all $x \in dom(\sigma)$, if x is not bound in $\mathcal{K}$, then all variables that occur in $\sigma(x)$ are also not bound in $\mathcal{K}$. ■

Proof. The statement is proved by contradiction: assume there is an x not bound in $\mathcal{K}$ and a variable y bound in $\mathcal{K}$ that occurs in $\sigma(x)$. Then by Definition 4.11.1 x should be bound in $\mathcal{K}$, which is a contradiction. □

For the increase of multiplicities, we have to determine a convex set of subtrees of which we have to increase the multiplicities. We introduce a constructive mechanism to determine the minimal set of nodes whose multiplicities need to be increased when increasing the multiplicity of some given node.

Intuitively, if we have to copy a node Q_m, then we must copy Q_m and all its children. Furthermore, if Q_m is either the binding node of some γ-variable that occurs in the instance of some γ-variable of binding position Q' or it has secondary type ν_0 or π_0 and occurs in the instance of some other node Q' of secondary type ν_0, then we must copy Q' as well.

Definition 4.11.3 (Determining Nodes to Increase Multiplicities) Let Q be an indexed formula tree, and σ an $\mathcal{L}$-admissible substitution. Let Q_m be a node of secondary type γ_0 or ν_0. The subtrees to copy in order to increase the multiplicity of Q_m's parent are given by the predicate $\mu(Q_m)$ that is inductively defined:

$$\mu(Q_m) = \{Q_m\} \quad \cup \left(\bigcup_{Q' \mid Q_m \prec Q'} \mu(Q')\right) \cup \left(\bigcup_{Q' \in Inst_Q(Q_m)} \mu(Q')\right) \cup \left(\bigcup_{Q' \in Inst_M(Q_m)} \mu(Q')\right)$$

[1] i.e. no nested subtrees.

[2] i.e. γ, δ-, ν-, and π-variables.

where $Inst_Q(Q_m)$ is the set of binding nodes of variables x, such that y occurs in $\sigma(x)$ and y is bound on Q_m. If Q_m is not a binding node, then $Inst_Q(Q_m) = \emptyset$. $Inst_M(Q_m)$ is the analogous set for nodes of secondary type ν_0.

We denote by $\mu(Q_m)_{min}$ the subset of the minimal nodes with respect to $\lhd_{\mathcal{L}}$ of $\mu(Q_m)$. ■

Lemma 4.11.4 Let Q be an indexed formula tree of secondary type γ_0 or ν_0, and σ an actual $\mathcal{L}$-admissible substitution. Then

1. $\mu(Q)_{min}$ contains only nodes of secondary type γ_0 or ν_0.
2. $\mu(Q)_{min}$ is a convex set of subtrees with respect to σ. ■

Proof. Both parts of the lemma are easy consequences of the definition of $\mu(Q)$ and $\mu(Q)_{min}$. □

Having determined the minimal nodes that need to be copied, we copy the subtrees Q whose roots are these nodes and rename the variables if necessary. Note that we cannot just create initial indexed formula trees, since we must keep track of the applications of Leibniz' equality introductions and extensionality introductions during the copying process. From this copying process we obtain a renaming ρ of the copied variables and an isomorphic function ι between the original subtrees and their copies. We agree that ρ is a total function, which is the identity function for all variables not occurring in Q, and ι is a total function which is the identity function on all nodes, except those occurring in Q.

The new subtrees are of secondary type γ_0 or ν_0 and are inserted as further children on the respective parent node of primary type γ or ν, which increases their multiplicities. The renaming ρ and the node mapping ι are used in order to carry over the substitution information by enlarging the variable substitution σ_Q and the modal substitution σ_M:

– From ρ and σ_Q we create the substitution $\sigma'_Q := \{\rho(\sigma_Q(x))/\rho(x) \mid x \in dom(\rho)\}$.

– From ι and σ_M we create the substitution $\sigma'_M := \{\iota(\sigma_M(Q))/\iota(Q) \mid Q \in dom(\iota)\}$.

Finally, the information about established connections in $\mathcal{C}$ is carried over by enlarging $\mathcal{C}$ using ι, i.e. we add the following connections:

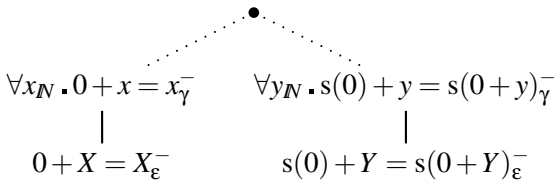
$$\{(\iota(c), \iota(c')) \mid (c, c') \in \mathcal{C}, c \text{ or } c' \in dom(\iota)\}$$

Definition 4.11.5 (Multiplicity Increase) Let Q be an indexed formula tree with actual $\mathcal{L}$-admissible substitution σ. Furthermore let Q_m be a node of secondary type γ_0 or ν_0. In order to increase the multiplicity of Q_m's parent we determine the set $\mu(Q_m)_{min}$. For each $Q' \in \mu(Q_m)_{min}$

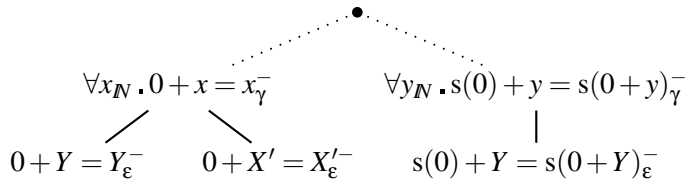
1. we copy Q' to obtain Q'' together with a variable renaming ρ and an isomorphic mapping from Q' to Q''. Subsequently we add Q'' to the parent node of Q';
2. we extend the variable and modal substitutions of σ respectively by

$$\{\rho(\sigma_Q(x))/\rho(x) \mid x \in dom(\rho)\} \text{ and } \{\iota(\sigma_M(Q))/\iota(Q) \mid Q \in dom(\iota)\}.$$ ■

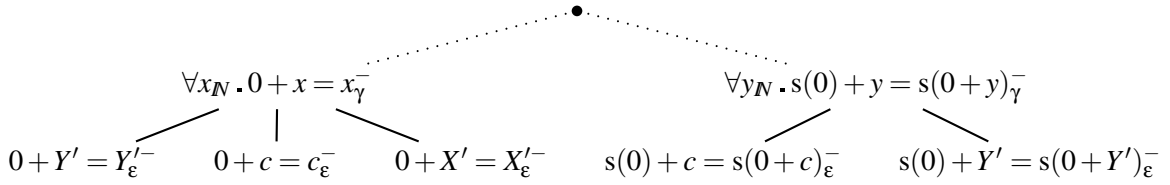
Example 4.11.6 We illustrate the dynamic increase of multiplicities in indexed formula trees with the following axioms: $\forall x_{\mathbb{N}} \,.\, 0+x=x$ and $\forall y_{\mathbb{N}} \,.\, \mathrm{s}(0)+y=\mathrm{s}(0+y)$. The axioms occur negatively in some larger formula, and thus the initial indexed formula tree is as follows:



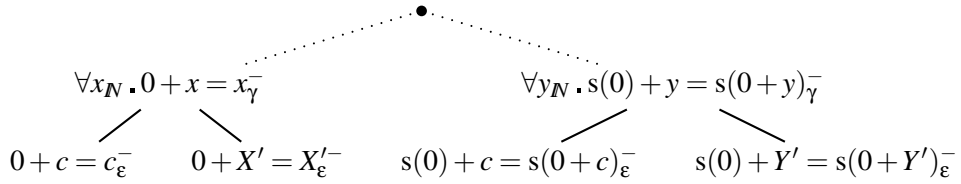
We first instantiate X with Y and therefore want to increase the multiplicity of $\forall x_{\mathbb{N}} \,.\, 0+x=x_\gamma^-$. That node is the only node of which we have to increase the multiplicity according to the determination of the convex set of subtrees. Increasing its multiplicity and subsequently applying the substitution $\{Y/X\}$ results in the following indexed formula tree:



Assume now we want to instantiate Y with some constant c and we want to increase the multiplicity of the parent node of Y's binding node. Determining the set of convex subtrees tells us that we have to increase the multiplicity not only of that node, but also the multiplicity of $\forall x_{\mathbb{N}} \,.\, 0+x=x_\gamma^-$, since Y occurs in the instantiation of a variable introduced by that node. Thus, increasing the multiplicities of both nodes and subsequently applying the substitution $\{c/Y\}$ results in



Note that without determining the convex set of subtrees would have resulted in the indexed formula tree



where the original, more "general" literal node $0+Y'=Y_\varepsilon'^-$ would have been lost, i.e. making the rule unsafe.

The multiplicity increasing rule is both sound and safe.

Lemma 4.11.7 (Soundness & Safeness of Multiplicity Increase) The constructive increase of multiplicities is sound and safe. ∎

Proof. The proof is achieved in two steps: first, we prove that the increase of multiplicites preserves the $\mathcal{L}$-admissibility of the substitution. Secondly, we prove that the indexed formula tree after increase of multiplicities has an $\mathcal{L}$-satisfiable path, if, and only if, the original indexed formula tree already had.

1. Preservation of $\mathcal{L}$-admissibility of substitution: By Lemma 4.11.4 the set $\mu(Q)_{min}$ is a convex set of subtrees from Q, and by Lemma 4.11.2 it follows that $\mu(Q)_{min}$ is maximal with respect to $\lhd_{\mathcal{L}}$. Thus the copies of $\mu(Q)_{min}$ are also maximal with respect to the new overall substitution. Furthermore, since the substitution before multiplicity increase was $\mathcal{L}$-admissible, the substitution extension $\{\rho(\sigma_Q(x))/\rho(x) \mid x \in dom(\rho)\}$ and $\{\iota(\sigma_M(Q))/\iota(Q) \mid Q \in dom(\iota)\}$ is irreflexive among the copies of $\mu(Q)_{min}$. Thus, the new overall substitution is $\mathcal{L}$-admissible.

2. For each $Q \in \mu(Q)_{min}$ the increase of multiplicities adds the copy Q' of Q to each path that contained Q. Thus, if Q has the prefix formula $pre(Q).\varphi^p$, then Q' has the label $pre(Q).\rho(\varphi^p)$. Since ρ is simply a renaming of variables, it trivially holds that $pre(Q).\varphi^p$ is $\mathcal{L}$-satisfiable if, and only if, $pre(Q).\rho(\varphi^p)$ is $\mathcal{L}$-satisfiable. □

Remark 4.11.8 The multiplicity increasing rule copies all Leibniz' equality and extensionality introductions. However, not all of them are necessary, since we are only interested in making an adequate copy of the actual proof state. The necessary introduction rules are those that introduced new subtrees in which actually occur connections. If no connection occurs in such a subtree we can disgard the rule application. However, for the purpose of the framework presented in Chapter 5 the current rule is adequate, because superflous parts obtained by copying can be removed using the weakening rule introduced in Chapter 5.

4.12 Soundness and Completeness Revisited

In order to obtain a complete proof procedure we have to search for the right multiplicity of γ- and ν-type nodes. An example is an iterative deepening over the maximum number of allowed multiplicities. But this means we have to restart the actual matrix proof search each time in every iteration, thus losing the information about substitutions and already established connections. The rule to increase the multiplicities overcomes that limitation and supports a demand-driven increase of the multiplicities. Furthermore it not only preserves existing substitutions and connections, but moreover carries this information over to the new subtrees that result from the increase of multiplicities. As an easy consequence the calculus that results from the addition of the multiplicity increasing rule is sound and complete.

Theorem 4.12.1 Let φ be a formula with respect to one of logics $\mathcal{L}$ of Definition 3.2.1. Then φ is $\mathcal{L}$-valid if, and only if, from an initial indexed formula tree for Q, we can derive an indexed formula tree Q' using the rules *instantiation*, *Leibniz' equality introduction*, *extensionality introduction*, *boolean* ζ*-expansion*, *cut*, and *multiplicity increase* such that the overall substitution is $\mathcal{L}$-admissible and there is some $P \in \mathcal{P}(Q')$ such that all paths in P are $\mathcal{L}$-unsatisfiable. ■

Proof. Follows from Theorem 4.10.1 and Lemma 4.11.7. □

4.13 Summary

The indexed formula trees introduced in this chapter are a generalised calculus that subsumes both the indexed formula trees in [Wallen, 1990] and the extensional expansion trees in [Pfenning, 1987].

The basic rules to manipulate them are the strict minimum to represent and check the $\mathcal{L}$-admissibility of substitutions for meta variables and node variables for the respective modal formulas, as well as the rules to deal with the introduction of Leibniz' equality for primitive equality and equivalence, the functional and boolean extensionality, the expansion of positive equivalences into implications, cut, and the dynamic increase of multiplicities.

As a result we obtain a sound and complete proof calculus for a whole class of logics. Although this calculus is not particularly intuitive, indexed formula trees are used in the proof theory of CoRe to represent quantifier dependencies. This calculus is the backbone of the framework with respect to the checks of the admissibility of the actual substitution. The intuitive part of the CoRe proof theory consists of free variable indexed formula trees which are added on top of indexed formula trees as presented in the next chapter.

Chapter 5

Free Variable Indexed Formula Trees

In this chapter we define free variable indexed formula trees on top of the underlying indexed formula trees. The working copy of an indexed formula tree is initially a free variable representation of the indexed formula tree. We use a "free variable" representation in order to ease the intuitive reading of trees. Indeed the application of rules usually requires the instantiation of variables, and it would be odd to instantiate variables for which there are still quantifiers around and the free variable representation avoids this. Since all introduced meta variables or parameters that result from γ- and δ-type nodes in the indexed formula tree are required to be new, this can be done.

Furthermore, free variable indexed formula trees provide support for the generation and application of rules from the logical context of some subformula. The working copy is manipulated by rule applications while still being linked with the original indexed formula tree to keep track of information about binding nodes of variables and modal prefixes of subformulas.

After the definition of initial free variable indexed formula trees in Section 5.1, we introduce in Section 5.2 a uniform notion for a logical context of subtrees as well as a uniform notion of a replacement rule inside a logical context. Based on these notions, we define in Section 5.3 the CoRe proof state and the actual rules for the manipulation of subtrees, i.e. the CoRe *calculus*. The CoRe calculus consists of 12 rules, namely instantiation, increase of multiplicities, Leibniz' equality introduction, extensionality introduction, contraction, weakening, modal permutation, an expansion rule for positive equivalences, resolution and rewriting replacement rule applications, propositional simplification, and cut. The soundness and safeness of the calculus rules is shown together with their definition. The chapter concludes with the completeness proof in Section 5.4 and a note about cut elimination in Section 5.5.

5.1 Initial Free Variable Indexed Formula Trees

Let us first introduce initial free variable indexed formula trees and subsequently add the rules to manipulate them. The reason for this is that we initially start the proof search with the conjecture, for which an initial indexed formula tree is created to represent the dependencies between quantifiers. Subsequently we initialise the proof state with that initial indexed formula tree and a working copy for it, which is an initial free variable indexed formula tree. Afterwards we define the calculus rules for the manipulation of such a proof state, which transform the working copy and possibly the indexed formula tree.

The definition of initial free variable indexed formula trees is essentially the straightforward rep-

resentation of an initial indexed formula tree without the object level quantifiers. Multiplicities of γ-type nodes are represented by α-nodes. However, the representation of modal quantification nodes, i.e. of ν- and π-type nodes, is less straightforward. Indeed, in order to have an intuitive representation, modal quantifications shall be explicitly represented – in contrast to object level quantification. Thus, a $\Box$-formula is represented as a $\Box$-formula. However, a problem arises from the multiplicities of ν-type nodes in Q. It is certainly not intuitive to represent e.g. a negative ν-type node of multiplicity n by a formula $\Box(\varphi_1 \wedge \ldots \wedge \varphi_n)$, since the proof theory of indexed formulas assigns to each φ_i a different (variable) position. In order to represent this a more adequate representation is $\Box(\varphi_1) \wedge \ldots \wedge \Box(\varphi_n)$. However, we need to reference the adjoined variable positions to each $\Box(\varphi_i)$, in order to be able to determine the modal prefix of some subformula.

Definition 5.1.1 (Initial Free Variable Indexed Formula Trees) We define *initial free variable indexed formula trees* R inductively over the structure of some given indexed formula tree Q. Each node of the tree has a formula as label, a polarity, a uniform type, and possibly the indexed formula tree node for which it is a working copy.

1. If $Q = A^p_-$ is a literal node, then $R = {}^p_- A_Q$ is a free variable indexed formula tree of the same label, polarity and uniform type than Q and a reference to Q. They are leaves of free variable indexed formula trees.

2. If $Q = \varepsilon(s,t)^p_\varepsilon$, then $R = {}^p_\varepsilon \varepsilon(s,t)_Q$ is a free variable indexed formula tree. They are leaves of free variable indexed formula trees.

3. If $Q = \zeta(s,t)^p_\zeta$, then $R = {}^p_\zeta \zeta(s,t)_Q$ is a free variable indexed formula tree. These are also leaves of free variable indexed formula trees.

4. If
$$Q = \begin{array}{c} \alpha(\mathbf{Label}(Q'))^p_\alpha \\ | \\ Q' \end{array}$$
is an indexed formula tree and R' is a free variable indexed formula tree for Q', then
$$R = \begin{array}{c} {}^p_\alpha\alpha(\mathbf{Label}(R')) \\ | \\ R' \end{array}$$
is a free variable indexed formula tree for Q.

5. If
$$Q = \begin{array}{c} \alpha(\mathbf{Label}(Q_1),\mathbf{Label}(Q_2))^p_\alpha \\ \diagup \quad \diagdown \\ Q_1 \qquad Q_2 \end{array}$$
is an indexed formula tree, R_1 and R_2 are free variable indexed formula trees for Q_1 and Q_2 respectively, then
$$R = \begin{array}{c} {}^p_\alpha\alpha(\mathbf{Label}(R_1),\mathbf{Label}(R_2)) \\ \diagup \quad \diagdown \\ R_1 \qquad R_2 \end{array}$$
is a free variable indexed formula tree for Q.

6. If

$$Q = \begin{array}{c} \beta(\mathbf{Label}(Q_1), \mathbf{Label}(Q_2))^p_\beta \\ \diagup \quad \diagdown \\ Q_1 \qquad Q_2 \end{array}$$

is an indexed formula tree, R_1 and R_2 are free variable indexed formula trees for Q_1 and Q_2 respectively, then

$$R = \begin{array}{c} {}^p_\beta\beta(\mathbf{Label}(R_1), \mathbf{Label}(R_2)) \\ \diagup \quad \diagdown \\ R_1 \qquad R_2 \end{array}$$

is a free variable indexed formula tree for Q.

7. If

$$Q = \begin{array}{c} \gamma^p x \,.\, \varphi(x)^p_\gamma \\ \diagup \quad | \quad \diagdown \\ Q_1 \quad \ldots \quad Q_n \end{array}$$

is an indexed formula tree and $R_1, \ldots, R_n$ are free variable indexed formula trees for $Q_1, \ldots, Q_n$ respectively, then let $R'_1 := R_1$, and

$$R'_{i+1} := \begin{array}{c} {}^p_\alpha\alpha(\mathbf{Label}(R'_i), \mathbf{Label}(R_{i+1})) \\ \diagup \quad \diagdown \\ R'_i \qquad R_{i+1} \end{array}$$

for $1 \leq i \leq (n-1)$. Then $R := R'_n$ is a free variable indexed formula tree for Q. Note that the R'_i do not have a reference to Q.

8. If

$$Q = \begin{array}{c} \delta^p x \,.\, \varphi(x)^p_\delta \\ | \\ Q' \end{array}$$

is an indexed formula tree and R is a free variable indexed formula tree for Q', then R is also a free variable indexed formula tree for Q.

9. If

$$Q = \begin{array}{c} \nu(\mathbf{Label}(Q_1))^p_\nu \\ \diagup \quad | \quad \diagdown \\ Q_1 \quad \ldots \quad Q_n \end{array}$$

is an indexed formula tree and $R_1, \ldots, R_n$ are free variable indexed formula trees for $Q_1, \ldots, Q_n$ respectively, then

$$R'_1 := \begin{array}{c} {}^p_\nu\nu(\mathbf{Label}(R_i))_{Q_1} \\ | \\ R_1 \end{array} \quad \text{and} \quad R'_{i+1} := \begin{array}{c} {}^\alpha_p\alpha(\mathbf{Label}(R'_i), \nu(\mathbf{Label}(R_{i+1}))) \\ \diagup \quad \diagdown \\ R'_i \qquad {}^p_\nu\nu(\mathbf{Label}(R_{i+1}))_{Q_{i+1}} \\ \qquad\qquad | \\ \qquad\qquad R_{i+1} \end{array}$$

for $1 \leq i \leq (n-1)$. Then $R := R'_n$ is a free variable indexed formula tree for Q.

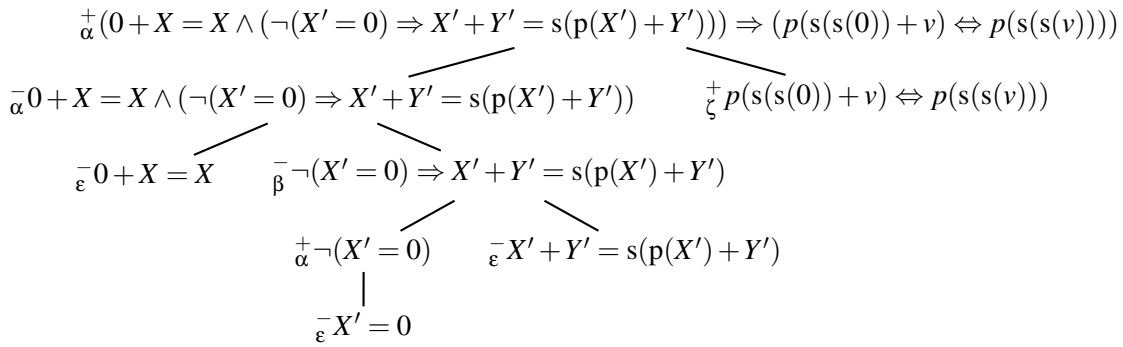


Figure 5.1: Initial free variable indexed formula tree for the running example

---

10. If

$$Q = \begin{array}{c}\pi(\mathbf{Label}(Q))^{p}_{\pi}\\ \big|\\ Q\end{array}$$

is an indexed formula tree and R is a free variable indexed formula tree for Q', then

$$R' = \begin{array}{c}\overset{p}{\delta}\pi(\mathbf{Label}(R))_{Q}\\ \big|\\ R'\end{array}$$

is a free variable indexed formula tree for Q. ■

Example 5.1.2 As an example consider the (initial) indexed formula tree from Example 4.1.2 (p. 33): The corresponding initial free variable indexed formula tree is shown in Figure 5.1.

It is convenient to introduce a notion of equality between free variable indexed formula trees that is defined over the α-equality of their labels. Note that we need α-equality, as there are still variable binders in the formula, for instance a usual λ-abstraction in higher-order logic, or universal quantification inside equivalences, which are not removed since they do not have a defined polarity.

Notation 5.1.3 Let R, R' be two free variable indexed formula trees. We say that R and R' are α-*equal* if, and only if, their labels are equal up to the renaming of bound variables. ■

It remains to define the modal prefix of some subtree of a free variable indexed formula tree.

Definition 5.1.4 (Prefix of Free Variable Indexed Formula Trees) Let Q be an indexed formula tree and R a node in a free variable indexed formula tree that belongs to Q. The modal prefix $pre(R)$

of R is a sequence nodes $\langle Q_1,\ldots,Q_n\rangle \in (\mathbf{V}_0 \cup \Pi_0)^*$ from Q and is defined as follows:

$$pre(R) := \begin{cases} \langle Q\rangle & \text{if } R \text{ has no parent node and is of primary type} \\ & \nu\text{- or } \pi\text{-type and associated node } Q \in \mathbf{V}_0 \cup \Pi_0 \\ \langle\rangle & \text{if } R \text{ has no parent node and is not of primary} \\ & \text{type } \nu\text{- or } \pi\text{-type} \\ \langle Q :: pre(R')\rangle & \text{if } R \text{ has parent node } R' \text{ and is of primary type} \\ & \nu\text{- or } \pi\text{-type and associated node } Q \in \mathbf{V}_0 \cup \Pi_0 \\ pre(R') & \text{if } R \text{ has parent node } R' \text{ and is not of primary} \\ & \text{type } \nu\text{- or } \pi\text{-type} \end{cases}$$

The *prefixed formula* of R is $pre(R).\varphi$, where φ is the label of R. ■

Remark 5.1.5 There is obviously a close relationship between the structure of a free variable indexed formula tree R and the term structure of $\mathbf{Label}(R)$. This allows us to define the *subtree occurrence* of a subtree R' of R as the subterm occurrence ρ of $\mathbf{Label}(R')$ within $\mathbf{Label}(R)$, i.e. $\mathbf{Label}(R)_{|\rho} = \mathbf{Label}(R')$. Each subtree R' of R is uniquely determined by this subterm occurrence and by abuse of notation we write $R_{|\rho}$ to denote R'. Note that not all subterm occurrences of $\mathbf{Label}(R)$ are subtree occurrences, since the tree structure does not extend below the literal node level.

Definition 5.1.6 (Proved and Disproved Free Variable Indexed Formula Trees) Let R be a literal free variable indexed formula tree. Then

- R is *proved*, if, and only if, either it has negative polarity and its label is *False*, or it has positive polarity and its label is *True*, or it is of primary type a ζ and the label is $\zeta(t,t)$.
- R is *disproved*, if, and only if, either it has positive polarity and its label is *False*, or it has negative polarity and its label is *True*, or it is of primary type ε and the label is $\varepsilon(t,t)$.

Let R be a free variable indexed formula tree. R is proved (resp. disproved), if, and only if,

- R is a proved (resp. disproved) literal free variable indexed formula tree,
- or R is of primary type α (resp. β) and some subtree is proved (resp. disproved),
- or R is of primary type β (resp. α) and all subtrees are proved (resp. disproved),
- or R is of primary type ν or π and its subtree is proved (resp. disproved). ■

Lemma 5.1.7 The definition of proved and disproved free variable indexed formula trees is accurate. ■

Proof. The intuition behind the above definition is that a free variable indexed formula tree R is proved if, and only if, the prefixed formula of R is $\mathcal{L}$-unsatisfiable. Similarly, R is disproved, if, and only if, the prefixed formula of R is $\mathcal{L}$-valid. We prove the accuracy by induction over the structure of R.

Base Case: *In this case R is a literal. By definition of $\mathcal{L}$-satisfiability of prefixed formulas, the prefixed formulas $pre(R).True^+$, $pre(R).False^-$, and $pre(R).\zeta(s,s)^+$ are not $\mathcal{L}$-satisfiable. Thus, those literal free variable indexed formula trees are proved. Analogously, the prefixed formulas $pre(R).True^-$, $pre(R).False^+$, and $pre(R).\varepsilon(s,s)^-$ are $\mathcal{L}$-valid. Thus those literal free variable indexed formula trees are disproved.*

Induction Step: *We prove the accuracy of "proved" by case analysis over the uniform type of R.*

A. $\mathbf{Label}(R)^p = \alpha^p(\mathbf{Label}(R_1)^{p_1}, \mathbf{Label}(R_2)^{p_2})$*: for all M, w, ρ :*

$$M_w^\rho \models_{\mathcal{L}} pre(R).\alpha^p(\mathbf{Label}(R_1)^{p_1}, \mathbf{Label}(R_2)^{p_2})$$
$$\Longleftrightarrow M_w^\rho \models_{\mathcal{L}} pre(R).\mathbf{Label}(R_1)^{p_1} \text{ and } M_w^\rho \models_{\mathcal{L}} pre(R).\mathbf{Label}(R_2)^{p_2})$$

Since $pre(R) = pre(R_i), i = 1,2$, R is proved, if, and only if, at least one of its subtrees is proved.

B. $\mathbf{Label}(R)^p = \beta^p(\mathbf{Label}(R_1)^{p_1}, \mathbf{Label}(R_2)^{p_2})$*: for all M, w, ρ :*

$$M_w^\rho \models_{\mathcal{L}} pre(R).\beta^p(\mathbf{Label}(R_1)^{p_1}, \mathbf{Label}(R_2)^{p_2})$$
$$\Longleftrightarrow M_w^\rho \models_{\mathcal{L}} pre(R).\mathbf{Label}(R_1)^{p_1} \text{ or } M_w^\rho \models_{\mathcal{L}} pre(R).\mathbf{Label}(R_2)^{p_2})$$

Since $pre(R) = pre(R_i), i = 1,2$, R is proved, if, and only if, both subtrees are proved.

C. $\mathbf{Label}(R)^p = \nu^p(\mathbf{Label}(R_1)^{p_1})$*: For all M, w, ρ :*

$$M_w^\rho \models_{\mathcal{L}} pre(R).\nu^p(\mathbf{Label}(R_1)^{p_1}) \Longleftrightarrow M_w^\rho \models_{\mathcal{L}} pre(R_1).\mathbf{Label}(R_1)^{p_1}$$

Thus, R is proved if, and only if, R_1 is proved.

D. $\mathbf{Label}(R)^p = \pi^p(\mathbf{Label}(R_2)^{p_1})$*: similar to the previous case.* □

5.1.1 Paths in Free Variable Indexed Formula Trees

Similarly to indexed formula trees we define (horizontal) paths for free variable indexed formula trees. We show that there is a correspondence between the paths through an initial indexed formula tree Q and the paths through the initial free variable indexed formula tree R for Q. This correspondence is exploited in order to establish the initial validity relationship between Q and its "working copy" R.

Definition 5.1.8 (Paths in Free Variable Indexed Formula Trees) Let R be a free variable indexed formula tree. A path in R is a sequence $\ll R_1, \ldots, R_n \gg$ of α-related nodes in R. The sets $\mathcal{P}(R)$ of paths through R is the smallest set containing $\{\ll R \gg\}$ and which is closed under the following operations:

α-Decomposition: If R' is a node of primary type α and subtrees R_1, R_2, and $P \cup \{\ll \Gamma, R' \gg\} \in \mathcal{P}(R)$, then $P \cup \{\ll \Gamma, R_1, R_2 \gg\} \in \mathcal{P}(R)$.

β-Decomposition: If R' is a node of primary type β and subtrees R_1, R_2, and $P \cup \{\ll \Gamma, R' \gg\} \in \mathcal{P}(R)$, then both $P \cup \{\ll \Gamma, R_1 \gg\} \in \mathcal{P}(R)$ and $P \cup \{\ll \Gamma, R_2 \gg\} \in \mathcal{P}(R)$.

ν-Decomposition: If R' is a node of primary type ν and subtree R_1, and $P \cup \{\ll \Gamma, R' \gg\} \in \mathcal{P}(R)$, then $P \cup \{\ll \Gamma, R_1 \gg\} \in \mathcal{P}(R)$.

π-Decomposition: If R' is a node of primary type π and subtree R_1, and $P \cup \{\ll \Gamma, R' \gg\} \in \mathcal{P}(R)$, then $P \cup \{\ll \Gamma, R_1 \gg\} \in \mathcal{P}(R)$. ■

Note the absence of decomposition rules for γ and δ-nodes since free variable indexed formula trees have no such nodes. Thus $\mathcal{P}(Q)$ constains sets of paths that do not occur in $\mathcal{P}(R)$. Note further that $\mathcal{P}(R)$ contains sets of paths, that contain nodes from R without reference to some node in Q.

Definition 5.1.9 (Connections in Free Variable Indexed Formula Trees) Let R be a free variable indexed formula tree. A connection is a pair (R', R'') of nodes with the same label, the same modal prefix under the actual modal substitution, and opposite polarities, such that there is a path $\ll \Gamma, R', R'' \gg$ in some $P \in \mathcal{P}(R)$. ■

Definition 5.1.10 ($\mathcal{L}$-Unsatisfiable and $\mathcal{L}$-Satisfiable Paths) Let R be a free variable indexed formula tree and $p \in P$ ($P \in \mathcal{P}(R)$) a path through R. The path p *is* $\mathcal{L}$*-unsatisfiable* if p contains either a positive node with label *True*, a negative node with label *False*, a positive node with label $\zeta(t,t)$, or two nodes that form a connection. If p is not $\mathcal{L}$-unsatisfiable, then p is said to be $\mathcal{L}$*-satisfiable*. ■

5.2 Logical Context and Replacement Rules

The free variable indexed formula tree contains all the information necessary to determine statically the *logical context*[1] logical context for any of its subtrees. Moreover it determines all possible rules within the logical context to manipulate the subtree under consideration. The key information is the annotated uniform type and the polarity. Indeed the logical context of some subtree R consists simply of all those subtrees that are connected with R via a node of uniform type α, and this can be checked statically from the free variable indexed formula tree. To see this, consider an α-type formula $(A \vee B)^+$: applying the respective decomposition rule on $(A \vee B)^+$ in some sequent calculus corresponds to the following inference step:

$$\frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash (A \vee B), \Delta} \vee\text{-}R$$

Obviously the two components A and B of the α-type formula occur in the same sequent and thus are in the same logical context. Generalising this observation for any two nested subformulas of A and B it is easy to see that given some subformula φ_1 in A and some subformula φ_2 in B, the successive application of all sequent calculus decomposition rules starting from $\Gamma \vdash (A \vee B), \Delta$ results in, among others, a sequent of the form $\Gamma' \vdash \varphi_1, \varphi_2, \Delta'$. We say φ_1 *and* φ_2 *are* α*-related*. Conversely, we say that two subformulas are β*-related*, if they are related by a β-type formula.

However, this is only valid for classical logics. Due to the modal connectives $\Diamond$ and $\Box$ in modal logics there might not be a sequent containing both formulas, even though they are α-related. But, nevertheless the converse is true, i.e. if two formulas are not α-related, they can never occur in a same sequent. Thus, the general pattern is that if two formulas are α-related then and only then can they be in the same logical context. Whether or not they really are in the same logical context can be statically checked by comparing the modal prefixes of both formulas: if there is an $\mathcal{L}$-admissible substitution that unifies both prefixes, then both formulas are indeed in the same logical context.

Having determined the formulas that are in the logical context of some formula, we are now concerned with the determination of the possible rules which can be generated from this context. To motivate this, consider the goal sequent $A \Rightarrow (B \Rightarrow C) \vdash C$. Applying $A \Rightarrow (B \Rightarrow C)$ to C means that the goal to prove C is replaced by the goal to prove A and B. In this case both occurrences of C have opposite polarities and are α-related via $\vdash$. Furthermore, the new subgoals, i.e. the positive occurrences of A and B, can be determined statically from the formula by collecting all the formulas that are β-related to the negative occurrence of C. This enables generating rules from a formula by fixing the left-hand side, e.g. the negative C. The right-hand side of the rule is then the list of all formulas that are β-related to the left-hand side, and we write

$$C^+ \rightarrow \langle A^+, B^+ \rangle$$

to indicate that this rule refines some positive C to the (positive) subgoals A and B. Analogously, if there is a negative equation or a negative equivalence in the context, i.e. an ε-type formula $\varepsilon(s,t)$, we

[1] Note that by logical context we mean the information beyond for instance the scope of variables which is already provided by the corresponding indexed formula tree.

obtain the rule

$$s^\circ \to \langle t^\circ, \varphi_1^{p_1}, \ldots, \varphi_n^{p_n} \rangle$$

where the $\varphi_i^{p_i}$ are, again, the formulas β-related to $\varepsilon(s,t)$. This rule contains the information, that some goal formula $\varphi(s)$ where s has an arbitrary polarity – even no polarity – can be refined to the subgoals $\varphi(t), \varphi_1^{p_1}, \ldots, \varphi_n^{p_n}$.

Before formalising the notion of replacement rules, we introduce the notion of a *weakened signed formula*. It consists of weakening α-connected parts of some indexed formula, and this is motivated by the following observation: consider the negative formula $(A \vee B) \Rightarrow C$ which using the uniform notation syntax is $\beta^-(\alpha^+(A^+, B^+), C^-)$. Applying this rule to some positive C^+ would refine C^+ to $\alpha^+(A,B)$. However, this is not necessary, since it would also be sound to refine C^+ only to either A^+ or B^+. These additional possibilities can be obtained from the β-related formula $\alpha^+(A^+, B^+)$ by *weakening* some α-parts. In order to include this into the formal definition of replacement rules, we define for some given free variable indexed formula tree Q with label φ^p the set of free variable indexed formula trees that can be obtained from Q by weakening.

Definition 5.2.1 (Weakening of Free Variable Indexed Formula Trees) Let R be a free variable indexed formula tree. The set $\mathit{Weakened}(R)$ of weakened free variable indexed formula trees for R is defined recursively over the structure of R:

$$\begin{array}{rcl}
\mathit{Weakened}(R) & = & \{R\} \text{ if, and only if, } R \text{ is a literal node} \\
\mathit{Weakened}(\alpha^p(R_1,R_2)) & = & \{\alpha^p(R_1^w, R_2^w) \mid R_i^w \in \mathit{Weakened}(R_i), i = 1,2\} \\
 & & \cup \mathit{Weakened}(R_1) \cup \mathit{Weakened}(R_2) \\
\mathit{Weakened}(\beta^p(R_1,R_2)) & = & \{\beta^p(R_1^w, R_2^w) \mid R_i^w \in \mathit{Weakened}(R_i), i = 1,2\} \\
\mathit{Weakened}(\nu^p(R)) & = & \{\nu^p(R^w) \mid R^w \in \mathit{Weakened}(R)\} \\
\mathit{Weakened}(\pi^p(R)) & = & \{\pi^p(R^w) \mid R^w \in \mathit{Weakened}(R)\}
\end{array}$$

■

Lemma 5.2.2 Let R be a free variable indexed formula tree of polarity p and $R' \in \mathit{Weakened}(R)$ of polarity p'. Then:

$$M_w^\rho \models_{\mathcal{L}} \mathit{pre}(R).\mathbf{Label}(R)^p \Longrightarrow M_w^\rho \models_{\mathcal{L}} \mathit{pre}(R).\mathbf{Label}(R')^p$$

Proof. *The proof is by induction over the structure of the formula R.*

Base Case: *In this case R is a literal and $\mathit{Weakened}(R) = \{R\}$. Thus the statement holds trivially.*
Induction Step: *We consider the four different cases for R:*

(A) $R = \alpha^p(R_1,R_2)$: then

$$\begin{array}{ll}
 & M_w^\rho \models_{\mathcal{L}} \mathit{pre}(R).\alpha^p(R_1,R_2) \\
\overset{\text{Lemma 4.1.8}}{\Leftrightarrow} & \text{for all } w' \in M_w^\rho(\mathit{pre}(R)) M_{w'}^\rho \models_{\mathcal{L}} \alpha^p(R_1,R_2) \\
\Leftrightarrow & \text{for all } w' \in M_w^\rho(\mathit{pre}(R)) M_{w'}^\rho \models_{\mathcal{L}} R_1 \text{ and } M_{w'}^\rho \models_{\mathcal{L}} R_2 \\
\overset{IH}{\Rightarrow} & \text{for all } w' \in M_w^\rho(\mathit{pre}(R)) M_{w'}^\rho \models_{\mathcal{L}} R_1' \text{ and } M_{w'}^\rho \models_{\mathcal{L}} R_2' \\
 & (\text{for any } R_1' \in \mathit{Weakened}(R_1) \text{ and } R_2' \in \mathit{Weakened}(R_2)) \\
\Leftrightarrow & M_w^\rho \models_{\mathcal{L}} \mathit{pre}(R).\alpha^p(R_1',R_2') \text{ and } M_w^\rho \models_{\mathcal{L}} \mathit{pre}(R).R_1' \\
 & \text{and } M_w^\rho \models_{\mathcal{L}} \mathit{pre}(R).R_2'
\end{array}$$

(B) $R = \beta^p(R_1, R_2)$*: then*

$$\begin{array}{rl} & M^{\rho}_w \models_{\mathcal{L}} pre(R).\beta^p(R_1,R_2) \\ \overset{Lemma\ 4.1.8}{\Leftrightarrow} & \text{for all } w' \in M^{\rho}_w(pre(R)) M^{\rho}_{w'} \models_{\mathcal{L}} \beta^p(R_1,R_2) \\ \Leftrightarrow & \text{for all } w' \in M^{\rho}_w(pre(R)) M^{\rho}_{w'} \models_{\mathcal{L}} R_1 \text{ or } M^{\rho}_{w'} \models_{\mathcal{L}} R_2 \\ \overset{IH}{\Rightarrow} & \text{for all } w' \in M^{\rho}_w(pre(R)) M^{\rho}_{w'} \models_{\mathcal{L}} R'_1 \text{ or } M^{\rho}_{w'} \models_{\mathcal{L}} R'_2 \\ & (\text{for any } R'_1 \in Weakened(R_1) \text{ and } R'_2 \in Weakened(R_2)) \\ \Leftrightarrow & M^{\rho}_w \models_{\mathcal{L}} pre(R).\beta^p(R'_1,R'_2) \end{array}$$

(C) $R = \nu^p(R')$*: then*

$$\begin{array}{rl} & M^{\rho}_w \models_{\mathcal{L}} pre(R).\nu^p(R') \\ \overset{Lemma\ 4.1.8}{\Leftrightarrow} & \text{for all } w' \in M^{\rho}_w(pre(R)) M^{\rho}_{w'} \models_{\mathcal{L}} \nu^p(R') \\ \Leftrightarrow & \text{for all } w' \in M^{\rho}_w(pre(R)) \text{ and for some } w'', w'Rw'' \; M^{\rho}_{w''} \models_{\mathcal{L}} R' \\ \overset{IH}{\Rightarrow} & \text{for all } w' \in M^{\rho}_w(pre(R)) \text{ and for some } w'', w'Rw'' \; M^{\rho}_{w''} \models_{\mathcal{L}} R'_w \\ & (\text{for any } R'_w \in Weakened(R')) \\ \Leftrightarrow & \text{for all } w' \in M^{\rho}_w(pre(R)) \; M^{\rho}_{w'} \models_{\mathcal{L}} \nu^p(R'_w) \\ \Leftrightarrow & M^{\rho}_w \models_{\mathcal{L}} pre(R).\nu^p(R'_w) \end{array}$$

(D) $R = \pi^p(R')$*: then*

$$\begin{array}{rl} & M^{\rho}_w \models_{\mathcal{L}} pre(R).\pi^p(R') \\ \overset{Lemma\ 4.1.8}{\Leftrightarrow} & \text{for all } w' \in M^{\rho}_w(pre(R)) M^{\rho}_{w'} \models_{\mathcal{L}} \pi^p(R') \\ \Leftrightarrow & \text{for all } w' \in M^{\rho}_w(pre(R)) \text{ and for all } w'', w'Rw'' \; M^{\rho}_{w''} \models_{\mathcal{L}} R' \\ \overset{IH}{\Rightarrow} & \text{for all } w' \in M^{\rho}_w(pre(R)) \text{ and for all } w'', w'Rw'' \; M^{\rho}_{w''} \models_{\mathcal{L}} R'_w \\ & (\text{for any } R'_w \in Weakened(R')) \\ \Leftrightarrow & \text{for all } w' \in M^{\rho}_w(pre(R)) \; M^{\rho}_{w'} \models_{\mathcal{L}} \pi^p(R'_w) \\ \Leftrightarrow & M^{\rho}_w \models_{\mathcal{L}} pre(R).\pi^p(R'_w) \end{array}$$

□

For the definition of replacement rules it is convenient to state the following corollary.

Corollary 5.2.3 (Connectable Free Variable Indexed Formula Trees) Let σ be an actual $\mathcal{L}$-admissible substitution, and let R and R' be two free variable indexed formula trees that have the same modal prefix with respect to σ but opposite polarities. If there exists an $R_w \in Weakened(R)$ which is α-equal (see Notation 5.1.3) to some $R'_w \in Weakened(R')$, then there exists no $\mathcal{L}$-model M which satisfies both R and R'. We say that R and R' are *connectable*. ■

Proof. Let the prefixed formulas of R and R' be respectively $w.\varphi^p$ and $w.\varphi'^{-p}$. Then R_w has the prefixed formula $w.\varphi^p_w$ while R'_w has the prefixed formula $w.\varphi^{-p}_w$. Assume there is an $\mathcal{L}$-model that satisfies both $w.\varphi^p$ and $w.\varphi'^{-p}$, i.e. for all possible worlds v and all assignments ρ it holds:

$$M^{\rho}_v \models w.\varphi^p \text{ and } M^{\rho}_v \models w.\varphi'^{-p}$$

By Lemma 5.2.2 it follows

$$M^{\rho}_v \models w.\varphi^p_w \text{ and } M^{\rho}_v \models w.\varphi^{-p}_w$$

Which is a contradiction. □

The consequence of the corollary is that two free variable indexed formula trees form a connection if, and only if, the intersection of their respective sets of weakened free variable indexed formula trees are non-empty.

In order to formalise the notion of a replacement rule, we first define the *conditions* of some subtree as the formal characterisation of the β-related formulas of some node.

Definition 5.2.4 (Node Conditions) Let R, c be nodes in some free variable indexed formula tree, such that c governs R. Let $R_1, \ldots, R_n$ be all maximal nodes that are below c and β-related to R. Then the conditions of R are given by the set $\mathcal{C}^c_R := \mathit{Weakened}(R_1) \times \ldots \times \mathit{Weakened}(R_n)$. ■

Replacement rules are of two kinds: the first kind are those where the left-hand side is a node with a polarity, and the second kind result from ε-type nodes. The former are called *resolution replacement rules*, while the latter are called *rewriting replacement rules*.

Definition 5.2.5 (Admissible Resolution Replacement Rules) Let R_0, R be nodes in some free variable indexed formula tree and σ the actual overall substitution. Then $R_0 \rightarrow \langle R_1, \ldots, R_n \rangle$ is an *admissible resolution replacement rule* for R, if, and only if,

1. R_0 and R have opposite polarities and are α-related by a node c,
2. the modal prefixes of R_0 and R are equal with respect to σ,
3. and $(R_1, \ldots, R_n) \in \mathcal{C}^c_{R_0}$. ■

Definition 5.2.6 (Admissible Rewriting Replacement Rules) Let R_0, R be nodes in some free variable indexed formula tree, R_0 of primary type ε and label $\varepsilon(s,t)$, and σ the actual overall substitution. Then $s \rightarrow \langle t, R_1, \ldots, R_n \rangle$ and $t \rightarrow \langle s, R_1, \ldots, R_n \rangle$ are *admissible rewriting replacement rules* for R, if, and only if,

1. R_0 and R are α-related by a node c,
2. the modal prefixes of R_0 and R are equal with respect to σ,
3. and it holds that $(R_1, \ldots, R_n) \in \mathcal{C}^c_{R_0}$. ■

5.3 CoRe Calculus Rules

In this section we finally present the basic rules of the CoRe framework. We first clarify the notion of a *proof state* in CoRe before presenting the twelve *basic rules*.

A CoRe proof state is parameterized over the actual logic $\mathcal{L}$ and consists of an indexed formula tree Q, an actual $\mathcal{L}$-substitution σ and a free variable indexed formula tree R. A proof state with respect to $\mathcal{L}$ is denoted by $[Q, \sigma \rhd_{\mathcal{L}} R]$. In order to prove some conjecture φ the initial proof state consists of the initial indexed formula tree Q_I for φ^+, the empty substitution, and the initial free variable indexed formula tree R_I for Q_I. The reasoning rules transform a proof state $[Q, \sigma \rhd_{\mathcal{L}} R]$ into a proof state $[Q', \sigma' \rhd_{\mathcal{L}} R']$. The transformation is *sound*, if, and only if, σ is $\mathcal{L}$-admissible for Q, σ' is $\mathcal{L}$-admissible for Q', and whenever there is an $\mathcal{L}$-satisfiable path in R, then there is an $\mathcal{L}$-satisfiable path in R'.

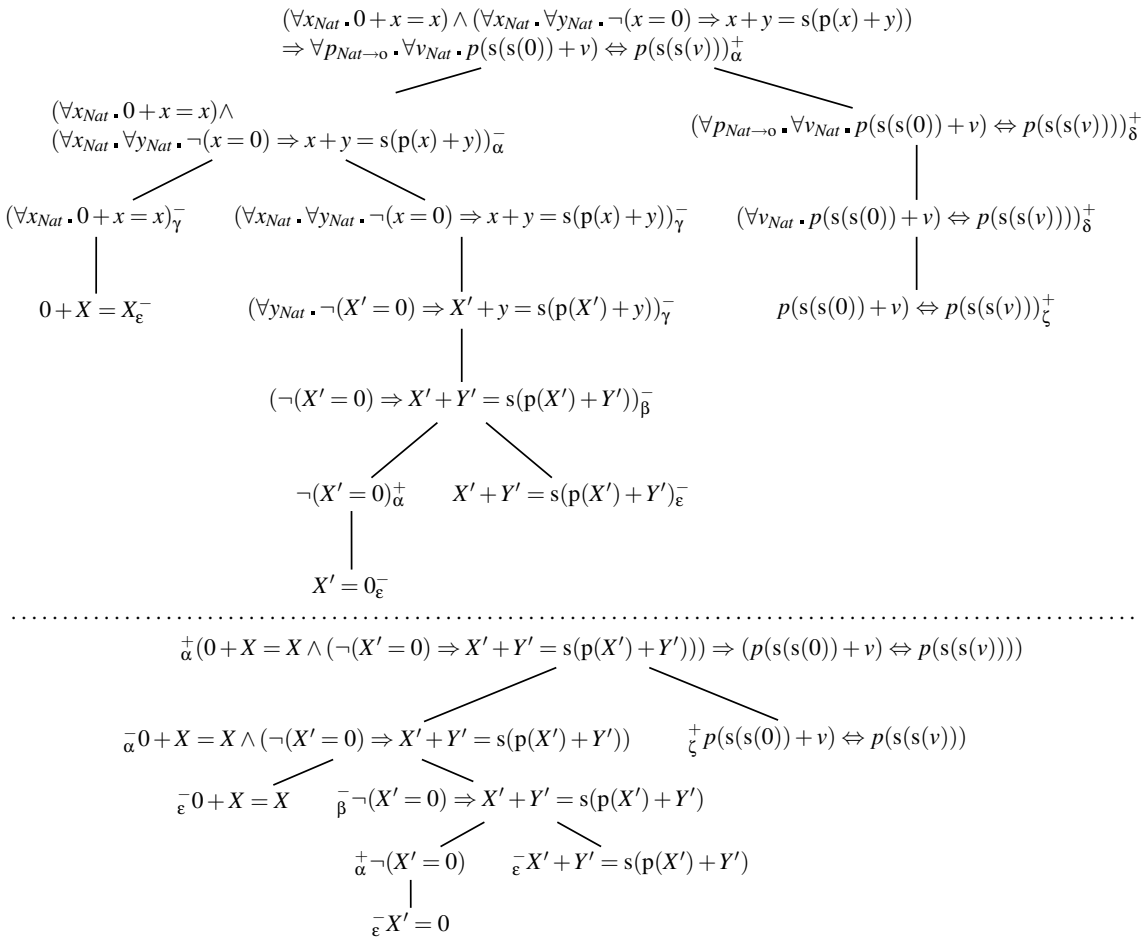


Figure 5.2: Initial proof state composed of the initial indexed formula tree (upper part) and the corresponding initial free variable indexed formula tree (lower part).

Definition 5.3.1 (Proof State, Soundness & Safeness) Let $\mathcal{L}$ be one of the logics under consideration, Q an indexed formula tree, σ an $\mathcal{L}$-substitution, and let R be a free variable indexed formula tree. Then a *proof state* is denoted by $[Q, \sigma \rhd_{\mathcal{L}} R]$. A proof step is a transformation of some proof state $[Q, \sigma \rhd_{\mathcal{L}} R]$ into another proof state $[Q', \sigma' \rhd_{\mathcal{L}} R']$, which is denoted as $[Q, \sigma \rhd_{\mathcal{L}} R] \longmapsto [Q', \sigma' \rhd_{\mathcal{L}} R']$.

Such a proof step is *sound* if, and only if, if σ is $\mathcal{L}$-admissible with respect to Q and there is an $\mathcal{L}$-satisfiable path in R then σ' is $\mathcal{L}$-admissible with respect to Q' and there is an $\mathcal{L}$-satisfiable path in R'.

A proof step is *safe* if, and only if, if σ' is $\mathcal{L}$-admissible with respect to Q' and there is an $\mathcal{L}$-satisfiable path in R' then σ is $\mathcal{L}$-admissible with respect to Q and there is an $\mathcal{L}$-satisfiable path in R. ■

Example 5.3.2 Consider as an example the formula $(\forall x_{Nat} \,.\, 0+x=x) \wedge (\forall x_{Nat} \,.\, \forall y_{Nat} \,.\, \neg(x=0) \Rightarrow x+y=\mathrm{s}(\mathrm{p}(x)+y)) \Rightarrow \forall p_{Nat\to o} \,.\, \forall v_{Nat} \,.\, p(\mathrm{s}(\mathrm{s}(0))+v) \Leftrightarrow P(\mathrm{s}(\mathrm{s}(v)))$; the initial proof state for the

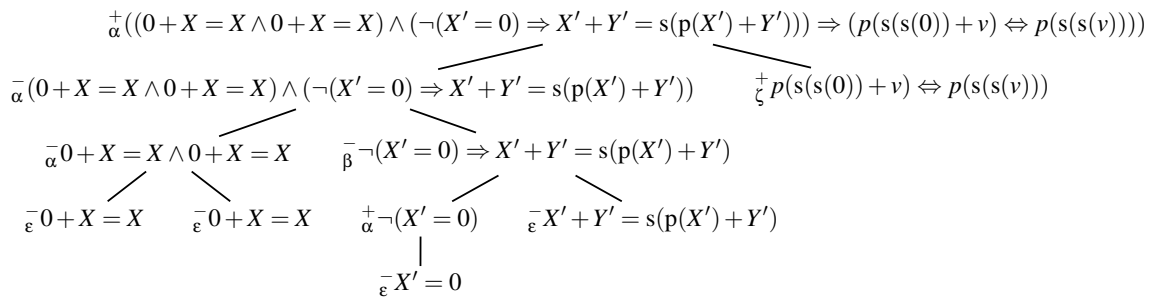


Figure 5.3: Free variable indexed formula tree resulting from contracting $\overset{-}{\varepsilon}0+X=X$.

positive formula is composed of the indexed formula tree and the free variable indexed formula tree shown in Figure 5.2. The initial substitution is the empty substitution.

The reasoning rules that manipulate a proof state $[Q,\sigma\rhd_{\mathcal{L}}R]$ are of two kinds: there are rules, like instantiation, Leibniz' equality introduction, etc., that affect Q,σ, and R and there are those that affect only R, like for instance the application of a replacement rule. The first kind of rules essentially change Q and the changes need to be propagated into R. How those changes are propagated depends on the changes in R by the second kind of rules, which we present first.

5.3.1 Contraction

Given a proof state $[Q,\sigma\rhd_{\mathcal{L}}R]$ and R_c a subtree in R, the *contraction* rule α-inserts a copy of R_c in R. Copying of a free variable indexed formula tree is the straighforward operation that preserves all references to Q.

Definition 5.3.3 (Contraction Rule) Let $[Q,\sigma\rhd_{\mathcal{L}}R]$ be a proof state, R_c a subtree of polarity p in R, and R'_c a copy of R_c. The application of the contraction rule results in the proof state $[Q,\sigma\rhd_{\mathcal{L}}R']$, where R' is obtained from R by replacing the subtree R_c by

$$\overset{p}{\alpha}\alpha(\mathbf{Label}(R_c),\mathbf{Label}(R'_c))$$
$$R_c \qquad R'_c$$

■

Example 5.3.4 Consider as an example the free variable indexed formula tree from Example 5.1.2 (p. 64). Applying the contraction rule to $\overset{-}{\varepsilon}0+X=X$ yields the free variable indexed formula tree shown in Figure 5.3. Note that the indexed formula tree of the proof state is not affected by that rule.

Lemma 5.3.5 The contraction rule is sound and safe. ■

Proof. Consider the label on R_c, i.e. the prefixed signed formula $pre(R_c).\mathbf{Label}(R_c)^p$. It is replaced by the prefixed signed formula $pre(R_c).\alpha^p(\mathbf{Label}(R_c)^p,\mathbf{Label}(R_c)^p)$. Obviously $pre(R_c).\mathbf{Label}(R_c)^p$ is $\mathcal{L}$-satisfiable if, and only if, the prefixed signed formula $pre(R_c).\alpha^p(\mathbf{Label}(R_c)^p,\mathbf{Label}(R_c)^p)$ is $\mathcal{L}$-satisfiable, since $\mathbf{Label}(R_c)=\mathbf{Label}(R'_c)$. □

5.3.2 Weakening

Given a proof state $[Q, \sigma \rhd_{\mathcal{L}} R]$ and R_w a subtree in R, the *weakening* rule replaces R_w by some $R'_w \in \mathit{Weakened}(R_w)$.

Definition 5.3.6 (Weakening Rule) Let $[Q, \sigma \rhd_{\mathcal{L}} R]$ be a proof state, R_w a subtree in R, and $R'_w \in \mathit{Weakened}(R_w)$. The application of the *weakening rule* results in a proof state $[Q, \sigma \rhd_{\mathcal{L}} R']$, where R' is obtained from R by replacing the subtree R_w by R'_w. ■

Example 5.3.7 As an example consider the free variable indexed formula tree in Figure 5.3: Applying the weakening rule to one of the $\bar{}_{\alpha} 0 + X = X$ we obtain again the initial free variable indexed formula tree.

Lemma 5.3.8 The weakening rule is sound. ■

Proof. Follows directly from Lemma 5.2.2. □

5.3.3 Structural Modal Permutation

We have retained the ν- and π-type modal connectives for intuition. However, during proof search we must be able to move these connectives over the logical connectives in order to apply the replacement rules. Take as an example a positive node R_1 in the free variable indexed formula tree that has a label $\Box(A \wedge B)$ and a further α-related negative node R_2 of label $(\Box A) \wedge (\Box B)$. In order to apply the rule from R_2 on R_1, the labels of these nodes must be equal. Thus, we need a rule to transform the former node into a positive node R_1 of the form $(\Box A) \wedge (\Box B)$. Generally we need a rule that allows us to move any modal connective over any standard logical connective, i.e. $\wedge, \vee, \neg$, and $\Rightarrow$. Writing these transformations as axioms results in either unsound or unsafe transformations in general, as for example the transformations $\Box(A \vee B) \Rightarrow \Box(A) \vee \Box(B)$ or $\Diamond(A \vee B) \Rightarrow \Diamond(A) \vee \Diamond(B)$. However, the information contained in a free variable indexed formula tree allows for sound and safe transformations: the ν- and π-type nodes in a free variable indexed formula tree have references to the nodes of secondary types ν_1 and π_1 in the indexed formula tree from which they originate. Thus, a modal node R of label $\Diamond(A \vee B)$ is indexed by that node Q in the indexed formula tree, which we write as $\Box_Q(A \wedge B)$. The prefixes of A and B are both $\mathit{pre}(R) :: \langle Q \rangle$. The transformation of R into a subtree for $\Diamond(A) \wedge \Diamond(B)$ preserves that information, i.e. we obtain an indexed version of that formula that is $\Diamond_Q(A) \wedge \Diamond_Q(B)$. Now, the prefixes of A and B are still $\mathit{pre}(R) :: \langle Q \rangle$. The preservation of the references during the transformation of the subtrees makes that transformation sound and safe.

Definition 5.3.9 (Structural Modal Permutation Rule) Let R, R_1 and R_2 be a free variable indexed formula tree. The *structural modal transformations* are then defined as

- Replacing the subtree ${}^{p}_{\nu}\nu(\alpha(R^{-p}))_Q$ by ${}^{p}_{\alpha}\alpha(\nu(R^{-p})_Q)$,
- Replacing the subtree ${}^{p}_{\nu}\nu(\alpha(R_1^{p_1}, R_2^{p_2}))_Q$ by ${}^{p}_{\alpha}\alpha(\nu(R_1^{p_1})_Q, \nu(R_2^{p_2})_Q)$,
- Replacing the subtree ${}^{p}_{\nu}\nu(\beta(R_1^{p_1}, R_2^{p_2}))_Q$ by ${}^{p}_{\alpha}\beta(\nu(R_1^{p_1})_Q, \nu(R_2^{p_2})_Q)$,

and analogously for π-type nodes. ■

Lemma 5.3.10 (Soundness & Safeness of the Structural Modal Transformations) The structural modal transformations are sound and safe. ■

Proof. We prove the statement for the two transformations of binary connectives. The proofs for the other cases are analogous.

Replacing the subtree $\mathbf{{}^{p}_{\nu}\nu(\alpha(R_1^{p_1}, R_2^{p_2}))_Q}$ by $\mathbf{{}^{p}_{\alpha}\alpha(\nu(R_1^{p_1})_Q, \nu(R_2^{p_2})_Q)}$: *the paths in the free variable indexed formula tree are transformed as follows:*

$$\ll \Gamma, \nu_Q(\alpha(R_1^{p_1}, R_2^{p_2})) \gg \quad \textit{into} \quad \ll \Gamma, \nu_Q(R_1^{p_1}), \nu_Q(R_2^{p_2}) \gg$$

The first path $\ll \Gamma, \nu_Q(\alpha(R_1^{p_1}, R_2^{p_2})) \gg$ results by the path decomposition rules in the path $\ll \Gamma, R_1^{p_1}, R_2^{p_2} \gg$, where $R_1^{p_1}$ and $R_2^{p_2}$ have the same prefix $\langle pre(R) :: Q \rangle$. The second path that results from the structural modal transformation can also be decomposed into $\ll \Gamma, R_1^{p_1}, R_2^{p_2} \gg$, and $R_1^{p_1}$ and $R_2^{p_2}$ again have the same prefix $\langle pre(R) :: Q \rangle$ as originally. Thus, the rule application is sound and safe.

Replacing the subtree $\mathbf{{}^{p}_{\nu}\nu(\beta(R_1^{p_1}, R_2^{p_2}))_Q}$ by $\mathbf{{}^{p}_{\alpha}\beta(\nu(R_1^{p_1})_Q, \nu(R_2^{p_2})_Q)}$: *the paths in the free variable indexed formula tree are transformed as follows:*

$$\begin{array}{lll} \ll \Gamma, \nu_Q(\beta(R_1^{p_1}, R_2^{p_2})) \gg & \textit{into} & \ll \Gamma, \nu_Q(R_1^{p_1}) \gg \\ & \textit{and} & \ll \Gamma, \nu_Q(R_2^{p_2}) \gg \end{array}$$

Again, the first path $\ll \Gamma, \nu_Q(\beta(R_1^{p_1}, R_2^{p_2})) \gg$ can be decomposed into the paths $\ll \Gamma, R_1^{p_1} \gg$ and $\ll \Gamma, R_2^{p_2} \gg$, where $R_1^{p_1}$ and $R_2^{p_2}$ have the same prefix $\langle pre(R) :: Q \rangle$. The second path that results from the structural modal transformation can also be decomposed into two paths $\ll \Gamma, R_1^{p_1} \gg$ and $\ll \Gamma, R_2^{p_2} \gg$, and $R_1^{p_1}$ and $R_2^{p_2}$ again have the same prefix $\langle pre(R) :: Q \rangle$ as originally. Thus, the rule application is sound and safe. □

Note that structural modal transformation rules "move" all modal connectives towards the literal nodes and group them there. This corresponds to the representation of modal prefixes in encodings of modal logic formulas into first-order logic formulas for resolution theorem proving as presented in [Nonnengart, 1995].

5.3.4 Replacement Rule Application

In this section we define the application of resolution replacement rules, whereas rewriting style replacement rules are presented below in Section 5.3.11.

The application of a resolution replacement rule to some node a consists of replacing the goal to prove a by the subgoals to prove the values of the rule. Since the values of a replacement rule may have different modal prefixes than a, a cannot be simply replaced by a conjunction of the values. Instead we replace a by some trivially proved formula and attach the values (i.e. the subgoals) at some node that governs a and complies to the modal prefix of the subgoal. In order to β-insert the subgoals at these nodes, we must be able to find a term that adequately represents the formula resulting from the insertion. Generally, given any two signed formulas $\varphi^p, \varphi'^{p'}$ we need to find a formula $\beta^p(\varphi^p, \varphi'^{p'})$. To this end we introduce the notion of β-terms and prove that those terms always exist.

Definition 5.3.11 (β-terms) Let $\mathcal{L}$ be one of the considered logics and p, p_1, p_2 defined polarities. A β-term for $\mathcal{L}$ with respect to the polarities p, p_1, and p_2 is a λ-term $\lambda\varphi_1 . \lambda\varphi_2 . \varphi$ of type $o \to o \to o$,

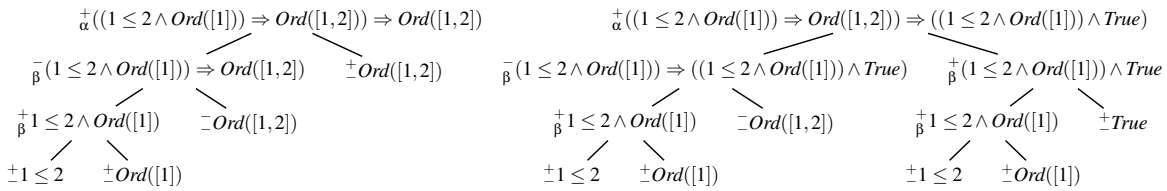


Figure 5.4: Free variable indexed formula trees before and after application of the replacement rule.

such that for any formulas ψ_1, ψ_2, if t is the $\beta\eta$ long normal form of $((\lambda\varphi_1 . \lambda\varphi_2 . \varphi)\ \psi_1\ \psi_2)$, then the signed formula t^p is of primary type β, φ_1 and φ_2 occur uniquely in t with respective polarities p_1 and p_2 and the occurrences of φ_1 and φ_2 inside t have the same modal prefix as t. ■

Lemma 5.3.12 (Existence of β-Terms) For any logic $\mathcal{L}$ from Definition 3.2.1, for any two formulas ψ_1, ψ_2, polarities p, p', and any prefix m there exists a binary β-term β for p, p, and p', such that the prefixes of ψ and ψ' inside $\beta^p(\psi^p, \psi^{p'})$ are equal to m. ■

Proof. To prove this lemma we give for each polarity constellation an example β-term which fulfills the requirements.

p	p'	**β-*term***
$+$	$+$	$\lambda\psi_1 . \lambda\psi_2 . (\wedge\ \psi_1\ \psi_2)$
$+$	$-$	$\lambda\psi_1 . \lambda\psi_2 . (\neg(\Rightarrow\ \psi_1\ \psi_2))$
$-$	$+$	$\lambda\psi_1 . \lambda\psi_2 . (\Rightarrow\ \psi_2\ \psi_1)$
$-$	$-$	$\lambda\psi_1 . \lambda\psi_2 . (\vee\ \psi_1\ \psi_2)$

□

The β-insertion of a subtree on some node consists in replacing the node with a β-type node with a respective β-term label and of subtrees the given subtree and the original node.

Definition 5.3.13 (Resolution Replacement Rule Application) Let $[Q, \sigma \rhd_{\mathcal{L}} R]$ be a proof state, a a node in R, and $u \to \langle v'_1, \ldots, v'_n \rangle$ $(n \geq 0)$ an admissible resolution replacement rule for a (i.e. $v'_i \in Weakened(v_i)$, v_i β-related to u) such that u and a are *connectable* (see Corollary 5.2.3). The application of $u \to \langle v'_1, \ldots, v'_n \rangle$ to a is defined as follows:

- For each v'_i, we determine the node p_i which (a) governs a and (b) has the same modal prefix than v'_i with respect to σ and β-insert v'_i on p_i.
- subsequently, we replace the subtree a by an initial free variable indexed formula tree for $True^+$, if a has positive polarity, or otherwise for $False^-$. ■

Example 5.3.14 Consider as an example the following free variable indexed formula tree obtained for the (positive) formula about lists of natural numbers

$$((1 \leq 2 \wedge Ord([1])) \Rightarrow Ord([1,2])) \Rightarrow Ord([1,2])$$

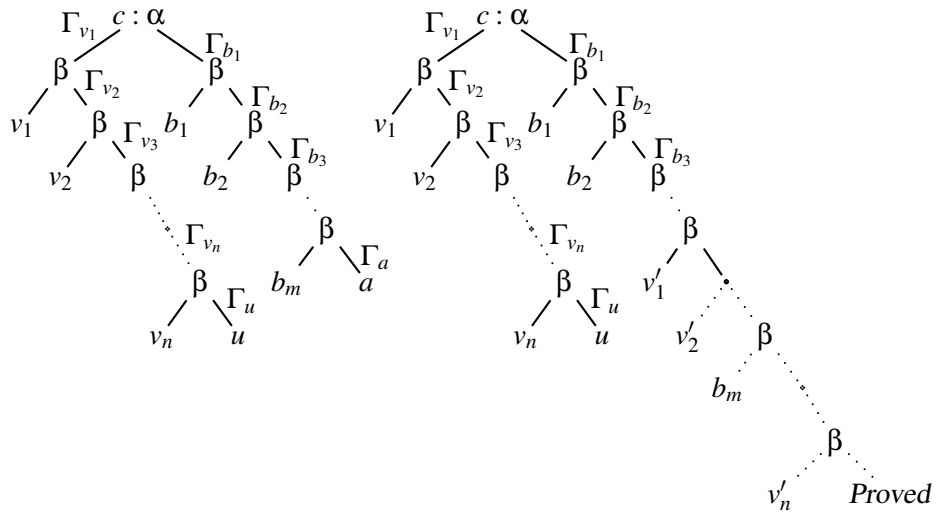


Figure 5.5: Structures of the free variable indexed formula tree before and after the rule application.

---

where $[1]$ denotes the singleton list containing 1, $[1,2]$ denotes the list with the elements 1 and 2, $\leq$ is the usual ordering over natural numbers, and *Ord* is a predicate for sorted lists. Creating a proof state for that signed formula yields the initial free variable indexed formula tree shown on the left-hand side of Figure 5.4. For the positive literal node ${}^{+}_{-}Ord([1,2])$ we obtain the replacement rule ${}^{-}_{-}Ord([1,2]) \rightarrow \left\langle {}^{+}_{\beta} 1 \leq 2 \wedge Ord([1]) \right\rangle$, which after application yields the free variable indexed formula tree shown on the right-hand side of Figure 5.4.

Lemma 5.3.15 (Soundness of Resolution Replacement Rule Application) The resolution replacement rule application is sound. ■

Proof. *Let $b_1, \ldots, b_m$ be the maximal subtrees, that are below c and β-related to a. The structure of the subtree rooted at c is shown on the left-hand side of Figure 5.5. The Γ_{v_i} denote the α-related parts to v_i, that are below v_{i-1} (or below c in case $i = 0$). Γ_u denotes the α-related parts to u, that are below the β-type node that connects the preceding subtree v_n.*

Analogously, the Γ_{b_j} denote the α-related parts to b_j, that are below b_{j-1} (or below c in case $j = 0$), and Γ_a the α-related parts to a that are below b_m (or c in case $m = 0$).

The paths through the subtree are sets of prefixed formulas and are of four kinds

1. *$\Gamma_{v_1}, \ldots, \Gamma_{v_i}, v_i, \Gamma_{b_1}, \ldots, \Gamma_{b_j}, b_j$, for all $1 \leq i \leq n$ and all $1 \leq j \leq m$.*
2. *$\Gamma_{v_1}, \ldots, \Gamma_{v_n}, \Gamma_u, u, \Gamma_{b_1}, \ldots, \Gamma_{b_j}, b_j$, for all $1 \leq j \leq m$.*
3. *$\Gamma_{v_1}, \ldots, \Gamma_{v_i}, v_i, \Gamma_{b_1}, \ldots, \Gamma_{b_m}, \Gamma_a, a$, for all $1 \leq i \leq n$*
4. *and the path $\Gamma_{v_1}, \ldots, \Gamma_{v_n}, \Gamma_u, u, \Gamma_{b_1}, \ldots, \Gamma_{b_m}, \Gamma_a, a$.*

We have to show, that if there was an $\mathcal{L}$-satisfiable path before rule application, then there is one afterwards. The structure of the new subtree is viewed on the right-hand side of Figure 5.5. We do a case analysis with respect to the above categorization:

1. *If the path was of the kind* $\Gamma_{v_1},\ldots,\Gamma_{v_i},v_i,\Gamma_{b_1},\ldots,\Gamma_{b_j},b_j$, *then it still exists in the new subtree.*

2. *If the path was of the kind* $\Gamma_{v_1},\ldots,\Gamma_{v_n},\Gamma_u,u,\Gamma_{b_1},\ldots,\Gamma_{b_j},b_j$, *then it still exists in the new subtree.*

3. *If the path was of the kind* $\Gamma_{v_1},\ldots,\Gamma_{v_i},v_i,\Gamma_{b_1},\ldots,\Gamma_{b_m},\Gamma_a,a$, *we can find in the new subtree the path* $\Gamma_{v_1},\ldots,\Gamma_{v_i},v_i,\Gamma_{v_i'},v_i'$ *where* $\Gamma_{v_i'}$ *are the* α*-related part in the new subtree, that are below* c. $\Gamma_{v_i'}$ *is a subset of* $\Gamma_{b_1},\ldots,\Gamma_{b_m},\Gamma_a$, *and from Lemma 5.2.2 we know that if* v_i *is* $\mathcal{L}$*-satisfiable , then so is* v_i'. *Thus the new path is a "subset" of the old path, and thus also* $\mathcal{L}$*-satisfiable.*

4. *If the path was* $\Gamma_{v_1},\ldots,\Gamma_{v_n},\Gamma_u,u,\Gamma_{b_1},\ldots,\Gamma_{b_m},\Gamma_a,a$, *then it is not* $\mathcal{L}$*-satisfiable, since it contains the connectable nodes* u *and* a *(cf. Corollary 5.2.3).*

Hence the resolution application of an admissible replacement rule is sound. □

The application of a resolution replacement rule is not safe, except when u has no β-related subtrees. In order to allow for a safe application of the resolution replacement rule, the contraction rule needs to be applied before applying the resolution replacement rule.

Lemma 5.3.16 The combination of the application of the contraction rule on some node b below c and that governs all β-insertion nodes for the new subgoals v_i' and the subsequent application of the resolution replacement rule is safe. ■

Proof. The proof is by induction over the structure of the direct subtree s *of* c *that contains* b.

Base Case s = b, *i.e.* b *is a direct subtree of* c. *The prefixed formula on* c *is* $pre(c).\alpha^p(\varphi,\mathbf{Label}(b))$, *where* φ *is the subformula containing the resolution replacement rule. After application of the contraction rule and the resolution replacement rule we obtain the prefixed formula* $pre(c).\alpha^p(\varphi,\alpha(\mathbf{Label}(b),\varphi'))$, *where* φ' *is the subformula obtained from* $\mathbf{Label}(b)$ *by resolution replacement rule application. Obviously it holds for any* $\mathcal{L}$*-model* M, *any world* w *and assignment* ρ*:*

$$M_w^\rho \models_{\mathcal{L}} pre(c).\alpha^p(\varphi,\alpha(\mathbf{Label}(b),\varphi')) \Longrightarrow M_w^\rho \models_{\mathcal{L}} pre(c).\alpha^p(\varphi,\mathbf{Label}(b)).$$

Induction Step: *The prefixed formula on* c *is of the form* $pre(c).\alpha^p(\varphi,\psi'(\psi(\mathbf{Label}(b))))$, *where* $\psi'(\psi(\mathbf{Label}(b)))$ *is the formula containing* $\mathbf{Label}(b)$ *and* $\psi(\mathbf{Label}(b))$ *is the formula on the parent node of* b. *By induction hypothesis we know that applying the contraction rule on the parent node* b_p *of* b *and subsequently applying the resolution replacement rule is safe. Thus we know that for any* $\mathcal{L}$*-model* M, *any world* w *and assignment* ρ*:*

$$\begin{aligned} M_w^\rho \models_{\mathcal{L}} pre(c).\alpha^p(\varphi,\psi'(\alpha(\psi(\mathbf{Label}(b)),\psi(\varphi')))) \\ \Longrightarrow M_w^\rho \models_{\mathcal{L}} pre(c).\alpha^p(\varphi,\psi'(\psi(\mathbf{Label}(b)))) \end{aligned}$$

We prove by case analysis over the uniform type of $\psi(\mathbf{Label}(b))$ *that*

$$\begin{aligned} M_w^\rho \models_{\mathcal{L}} pre(b_p).\psi^p(\alpha^p(\mathbf{Label}(b)^{p''},\varphi'^{p'})) \\ \Longrightarrow M_w^\rho \models_{\mathcal{L}} pre(b_p).\alpha^p(\psi^p(\mathbf{Label}(b)^{p''}),\psi^p(\varphi'^{p'})) \end{aligned}$$

A. $\psi^p = \lambda P\,.\,\alpha^p(\psi'', P^p)$, *where* ψ'' *is some signed formula. Then*

$$
\begin{array}{ll}
 & M^{\rho}_{w} \models_{\mathcal{L}} pre(b_p).\psi^p(\alpha^p(\mathbf{Label}(b)^{p''}, \varphi'^{p'})) \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(b_p))\ M^{\rho}_{w'} \models_{\mathcal{L}} \alpha^p(\psi'', \alpha^p(\mathbf{Label}(b)^{p''}, \varphi'^{p'})) \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(b_p))\ M^{\rho}_{w'} \models_{\mathcal{L}} \psi'' \text{ and } M^{\rho}_{w'} \models_{\mathcal{L}} \mathbf{Label}(b)^{p''} \text{ and } M^{\rho}_{w'} \models_{\mathcal{L}} \varphi'^{p'} \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(b_p))\ M^{\rho}_{w'} \models_{\mathcal{L}} \psi'' \text{ and } M^{\rho}_{w'} \models_{\mathcal{L}} \mathbf{Label}(b)^{p''} \\
 & \text{and } M^{\rho}_{w'} \models_{\mathcal{L}} \psi'' \text{ and } M^{\rho}_{w'} \models_{\mathcal{L}} \varphi'^{p'} \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(b_p))\ M^{\rho}_{w'} \models_{\mathcal{L}} \alpha^p(\psi'', \mathbf{Label}(b)^{p''}) \text{ and } M^{\rho}_{w'} \models_{\mathcal{L}} \alpha^p(\psi'', \varphi'^{p'}) \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(b_p))\ M^{\rho}_{w'} \models_{\mathcal{L}} \alpha^p(\alpha^p(\psi'', \mathbf{Label}(b)^{p''}), \alpha^p(\psi'', \varphi'^{p'})) \\
\Leftrightarrow & M^{\rho}_{w} \models_{\mathcal{L}} pre(b_p).\alpha^p(\psi^p(\mathbf{Label}(b)^{p''}), \psi^p(\varphi'^{p'}))
\end{array}
$$

B. $\psi^p = \lambda P\,.\,\beta^p(\psi'', \mathbf{Label}(b)^p)$, *where* ψ'' *is some signed formula. Then*

$$
\begin{array}{ll}
 & M^{\rho}_{w} \models_{\mathcal{L}} pre(b_p).\psi^p(\alpha^p(\mathbf{Label}(b)^{p''}, \varphi'^{p'})) \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(b_p))\ M^{\rho}_{w'} \models_{\mathcal{L}} \beta^p(\psi'', \alpha^p(\mathbf{Label}(b)^{p''}, \varphi'^{p'})) \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(b_p))\ M^{\rho}_{w'} \models_{\mathcal{L}} \psi'' \text{ or } (M^{\rho}_{w'} \models_{\mathcal{L}} \mathbf{Label}(b)^{p''} \text{ and } M^{\rho}_{w'} \models_{\mathcal{L}} \varphi'^{p'}) \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(b_p))\ (M^{\rho}_{w'} \models_{\mathcal{L}} \psi'' \text{ or } M^{\rho}_{w'} \models_{\mathcal{L}} \mathbf{Label}(b)^{p''}) \\
 & \text{and } (M^{\rho}_{w'} \models_{\mathcal{L}} \psi'' \text{ or } M^{\rho}_{w'} \models_{\mathcal{L}} \varphi'^{p'}) \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(b_p))\ M^{\rho}_{w'} \models_{\mathcal{L}} \beta^p(\psi'', \mathbf{Label}(b)^{p''}) \text{ and } M^{\rho}_{w'} \models_{\mathcal{L}} \beta^p(\psi'', \varphi'^{p'}) \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(b_p))\ M^{\rho}_{w'} \models_{\mathcal{L}} \alpha^p(\beta^p(\psi'', \mathbf{Label}(b)^{p''}), \beta^p(\psi'', \varphi'^{p'})) \\
\Leftrightarrow & M^{\rho}_{w} \models_{\mathcal{L}} pre(b_p).\alpha^p(\psi^p(\mathbf{Label}(b)^{p''}), \psi^p(\varphi'^{p'}))
\end{array}
$$

C. $\psi^p = \lambda P\,.\,\nu^p(\mathbf{Label}(b)^p)$*: analogously.*

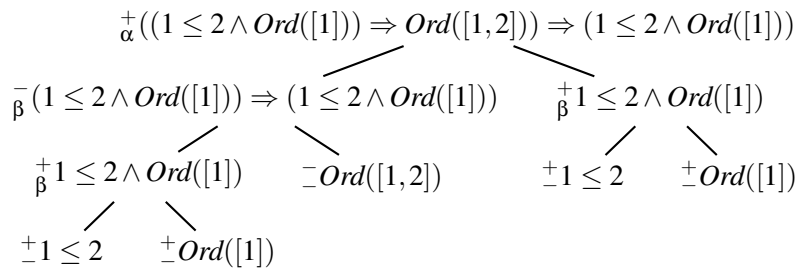
D. $\psi^p = \lambda P\,.\,\pi^p(\mathbf{Label}(b)^p)$*: analogously.* □

5.3.5 Simplification

Definition 5.3.17 (Simplification Rule) Let $[Q, \sigma \rhd_{\mathcal{L}} R]$ be a proof state, and let R' be a subtree of R. The simplification rule consists of

- if R' is proved, replace R by an initial free variable indexed formula tree for $True^+$ if the polarity of R is positive, and otherwise by an initial free variable indexed formula tree for $False^-$,
- if R' is disproved, replace R by an initial free variable indexed formula tree for $False^-$ if the polarity of R is positive, and otherwise by an initial free variable indexed formula tree for $True^+$,
- if R' a β-type node $\beta(R_1^{p_1}, R_2^{p_2})^p$ and not proved, but either R_1 or R_2 is proved, then
  - replace R' by R_i, if R_i is the non-proven subtree and $p = p_i$, or
  - replace R' by $\alpha(R_i^{p_i})^p$, if R_i is the non-proven subtree and $p \neq p_i$
- if R' an α-type node $\alpha(R_1^{p_1}, R_2^{p_2})^p$ and not disproved, but either R_1 or R_2 is disproved, then
  - replace R' by R_i, if R_i is the non-disproven subtree and $p = p_i$, or
  - replace R' by $\alpha(R_i^{p_i})^p$, if R_i is the non-disproven subtree and $p \neq p_i$
- Otherwise to leave R unchanged. ■

Example 5.3.18 Take as an example the free variable indexed formula tree from Figure 5.4 (p. 75): applying the simplification rule to the whole tree yields the simplified free variable indexed formula tree



Lemma 5.3.19 (Soundness & Safeness of Simplification) The simplification rule is sound and safe. ■

Proof. The proof is by case analysis over the cases in the definition of the simplification rule. The last case is trivially sound and safe. For the first two cases of the simplification rule the soundness and safeness follows directly from Lemma 5.1.7. For the third case we consider how the paths through R are affected by the simplification rule:

$$\ll \Gamma, \beta(\varphi^p, Proved^{p'})^p \gg \quad into \quad \ll \Gamma, \varphi^p \gg \tag{5.1}$$

$$respectively \ \ll \Gamma, \beta(\varphi^{-p}, Proved^{p'})^p \gg \quad into \quad \ll \Gamma, \alpha(\varphi^{-p})^p \gg \tag{5.2}$$

where $Proved^p$ is $True^+$, if $p = +$, and $False^-$ if $p = -$. For the path transformation (5.1) we have:

$$\begin{array}{rl} & M^{\rho}_w \models_{\mathcal{L}} pre(R').\beta(\varphi^p, Proved^{p'})^p \\ \Leftrightarrow & \text{for all } w' \in M^{\rho}_w(pre(R'))M^{\rho}_{w'} \models_{\mathcal{L}} \beta(\varphi^p, Proved^{p'})^p \\ \Leftrightarrow & \text{for all } w' \in M^{\rho}_w(pre(R'))M^{\rho}_{w'} \models_{\mathcal{L}} \varphi^p \text{ or } M^{\rho}_{w'} \models_{\mathcal{L}} Proved^{p'} \\ \overset{M^{\rho}_{w'} \not\models_{\mathcal{L}} Proved^{p'}}{\Leftrightarrow} & \text{for all } w' \in M^{\rho}_w(pre(R'))M^{\rho}_{w'} \models_{\mathcal{L}} \varphi^p \\ \Leftrightarrow & M^{\rho}_w \models_{\mathcal{L}} pre(R').\varphi^p \end{array}$$

For the path transformation (5.2) we have:

$$\begin{array}{rl} & M^{\rho}_w \models_{\mathcal{L}} pre(R').\beta(\varphi^{-p}, Proved^{p'})^p \\ \Leftrightarrow & \text{for all } w' \in M^{\rho}_w(pre(R'))M^{\rho}_{w'} \models_{\mathcal{L}} \beta(\varphi^p, Proved^{p'})^p \\ \Leftrightarrow & \text{for all } w' \in M^{\rho}_w(pre(R'))M^{\rho}_{w'} \models_{\mathcal{L}} \varphi^{-p} \text{ or } M^{\rho}_{w'} \models_{\mathcal{L}} Proved^{p'} \\ \overset{M^{\rho}_{w'} \not\models_{\mathcal{L}} Proved^{p'}}{\Leftrightarrow} & \text{for all } w' \in M^{\rho}_w(pre(R'))M^{\rho}_{w'} \models_{\mathcal{L}} \varphi^{-p} \\ \Leftrightarrow & \text{for all } w' \in M^{\rho}_w(pre(R'))M^{\rho}_{w'} \models_{\mathcal{L}} \alpha(\varphi^{-p})^p \\ \Leftrightarrow & M^{\rho}_w \models_{\mathcal{L}} pre(R').\alpha(\varphi^{-p})^p \end{array}$$

Thus, the simplification rule is sound and safe. □

To ease the definition of the second kind of rules, i.e. the rules that affect Q and whose effects need to be propagated into R, we describe the general principle for the propagation of changes in Q to R. To this end consider the respective rules and how they affect Q:

- The introduction of Leibniz' equality (cf. Definition 4.2.1) essentially inserts an α-related initial indexed formula tree L for $(\forall P\,.\,P(s) \Rightarrow P(t))^p$ to some given literal node $\overset{\xi}{\underset{\zeta}{\xi}}(s,t)$. This can be propagated to R by α-inserting on each occurrence of $\overset{\xi}{\underset{\zeta}{\xi}}(s,t)$ inside R a respective initial free variable indexed formula tree for L.

- The extensionality introduction (cf. Definition 4.3.3) behaves analogously to the previous rule: it inserts an α-related literal node for $(\lambda x\,.\,s = \lambda x\,.\,t)^p$ to some given literal node $\overset{\xi}{\underset{\zeta}{\xi}}(s,t)^p$. Again, this can be propagated to R by α-inserting on each occurrence of $\overset{\xi}{\underset{\zeta}{\xi}}(s,t)$ inside R an initial free variable indexed formula tree for $(\lambda x\,.\,s = \lambda x\,.\,t)^p$.

- The boolean ζ-expansion rule (cf. Definition 4.4.1) essentially inserts an α-related initial indexed formula tree L for $((A \Rightarrow B) \wedge (B \Rightarrow A))^+$ to some given ζ-type node $\zeta(A_o, B_o)$. This is propagated to R by α-inserting on each occurrence of $\zeta(A_o, B_o)$ inside R a respective initial free variable indexed formula tree for L.

- The instantiation rule (cf. Definition 4.5.2) instantiates γ-variables and ν-nodes. The instantiation of ν-type nodes does not affect R, except for the determination of the modal prefixes of subtrees. For the instantiation of a γ-variable X it essentially instantiates the labels of nodes, and, if X is a set variable, replaces the literal nodes of label $X(s_1,\ldots,s_n)^p$ by an initial indexed formula tree for $\sigma_Q(X(s_1,\ldots,s_n))^p_{\downarrow\beta\eta}$. This can be propagated to R by instantiating the labels of nodes in R and replacing the occurrences of $X(s_1,\ldots,s_n)^p$ by initial free variable indexed formula trees for $\sigma_Q(X(s_1,\ldots,s_n))^p_{\downarrow\beta\eta}$.

- The increase of multiplicities (cf. Definition 4.11.5) adds copies of indexed formula trees on respective γ-type and ν-type nodes and extends the overall substitution. The extension of the substitution must be handled in combination with the insertion of the new indexed formula tree. From the copying of the respective indexed formula tree we obtain a renaming ρ and an isomorphic mapping ι from nodes in the old indexed formula tree Q to nodes in the new indexed formula tree Q'. Inside R we determine the largest subtrees that contain only nodes from Q (literals as well as ν- and π-type nodes annotated by nodes from Q of secondary type ν_0 and π_0). Then we α-insert a copy of those subtrees by renaming the labels by ρ and the referenced nodes in Q in accordance with ι.

5.3.6 Leibniz' Equality

Definition 5.3.20 (Leibniz' Equality Introduction Rule) Let $[Q, \sigma \rhd_{\mathcal{L}} R]$ be a proof state, R_e an ε- or ζ-type subtree in R, Q_e its associated subtree in Q of polarity p and label $\overset{\xi}{\underset{\zeta}{\xi}}(s,t)$, and Q'_e an initial indexed formula tree for $(\forall P\,.\,P(s) \Rightarrow P(t))^p$. The application of the *Leibniz' Equality Introduction rule* on R_e results in a proof state $[Q', \sigma \rhd_{\mathcal{L}} R']$. Thereby Q' is the result of applying the Leibniz' equality introduction rule on Q_e which consisted in replacing Q_e by

$$Q_{Leibniz} = \begin{array}{c} \mathbf{Label}(Q_e)^p_\alpha \\ \diagup \quad \diagdown \\ Q_e \qquad Q'_e \end{array}$$

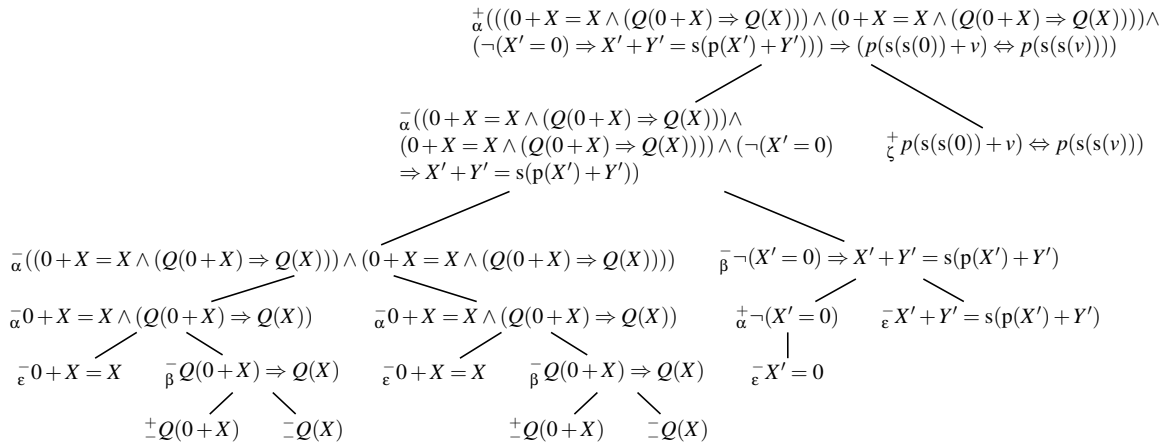


Figure 5.6: Free variable indexed formula tree resulting from introducing Leibniz' equality for the ε-type formula $\overset{-}{\varepsilon}0+X=X$.

Furthermore R' is the result of replacing all literal nodes R_L in R that are annotated by Q_e with

$$\begin{array}{c}\overset{p}{\alpha}\alpha(\mathbf{Label}(R_L),\mathbf{Label}(R'_L))\\ \diagup\quad\diagdown\\ R_L\qquad R'_L\end{array}$$

where R'_L is an initial free variable indexed formula tree for Q'_e. ∎

Example 5.3.21 Consider as an example the proof state for the formula

$$(\forall x_{Nat}\,.\,0+x=x)\wedge(\forall x_{Nat}\,.\,\forall y_{Nat}\,.\,\neg(x=0)\Rightarrow x+y=\mathrm{s}(\mathrm{p}(x)+y))$$
$$\Rightarrow\forall p_{Nat\to\mathrm{o}}\,.\,\forall v_{Nat}\,.\,p(\mathrm{s}(\mathrm{s}(0))+v)\Leftrightarrow P(\mathrm{s}(\mathrm{s}(v)))$$

after application of the contraction rule; the indexed formula tree for that proof state is shown in Figure 4.1 (p. 34) and the free variable indexed formula tree is shown in Figure 5.3 (p. 72). The introduction of the Leibniz' equality for the ε-type formula $\overset{-}{\varepsilon}0+X=X$ yields on the one hand the indexed formula tree shown in Figure 4.2 (p. 41). The adaptation of the two occurrences of that formula in the free variable indexed formula tree yields the free variable indexed formula tree shown in Figure 5.6.

Lemma 5.3.22 The Leibniz' equality introduction rule is sound and safe. ∎

Proof. The label of R_L is the prefixed formula $pre(R_L).\overset{\xi}{\zeta}{}^{p}(s,t)$ and s,t are of type τ. This prefixed formula is replaced by the formula $pre(R_L).\alpha^{p}(\overset{\xi}{\zeta}{}^{p}(s,t),\forall^{p}P\,.\,P(s)\Rightarrow P(t))$. Let M^{ρ}_{w} be an $\mathcal{L}$-model with possible world w and assignment ρ that satisfies $pre(R_L).\overset{\xi}{\zeta}{}^{p}(s,t)$. We prove the soundness and

safeness for the case where $\overset{\varepsilon}{\zeta}{}^{p}(s,t)$ *is of the form* $(s=t)^+$. *The other cases are analogous.*

$$
\begin{array}{ll}
 & M^{\rho}_{w} \models_{\mathcal{L}} pre(R_L).(s=t)^+ \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'} \models_{\mathcal{L}} (s=t)^+ \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'} \not\models_{\mathcal{L}} s=t \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'}(s) \neq M^{\rho}_{w'}(t) \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'}(s) \neq M^{\rho}_{w'}(t) \\
 & \quad \text{and there exists } p_{M_w(\tau\to o)}\ .\ (\neg p(M^{\rho}_{w'}(s)))\ \text{and}\ p(M^{\rho}_{w'}(t)) \\
\overset{P\ new}{\Leftrightarrow} & \text{for all } w' \in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'}(s) \neq M^{\rho}_{w'}(t) \\
 & \quad \text{and there exists } p \in M_w(\tau\to o)\ M^{\rho[p/P]}_{w'} \not\models_{\mathcal{L}} P(s) \Rightarrow P(t) \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'}(s) \neq M^{\rho}_{w'}(t) \\
 & \quad \text{and not for all } p \in M_w(\tau\to o)\ M^{\rho[p/P]}_{w'} \models_{\mathcal{L}} P(s) \Rightarrow P(t) \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'}(s) \neq M^{\rho}_{w'}(t) \\
 & \quad \text{and } M^{\rho}_{w'} \not\models_{\mathcal{L}} \forall P_{\tau\to o}\,.\,P(s) \Rightarrow P(t) \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'} \models_{\mathcal{L}} (s=t)^+ \text{ and } M^{\rho}_{w'} \models_{\mathcal{L}} (\forall P_{\tau\to o}\,.\,P(s) \Rightarrow P(t))^+ \\
\Leftrightarrow & \text{for all } w' \in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'} \models_{\mathcal{L}} \alpha^+(s=t^+, (\forall P_{\tau\to o}\,.\,P(s) \Rightarrow P(t))^+) \\
\Leftrightarrow & M^{\rho}_{w} \models_{\mathcal{L}} pre(R_L).\alpha^+(s=t^+, (\forall P_{\tau\to o}\,.\,P(s) \Rightarrow P(t))^+)
\end{array}
$$

□

5.3.7 Extensionality

Definition 5.3.23 (Extensionality Introduction Rule) Let $[Q, \sigma \rhd_{\mathcal{L}} R]$ be a proof state, R_e an ε- or ζ-type subtree in R, Q_e its associated subtree in Q of polarity p and label $\overset{\varepsilon}{\zeta}(s,t)$ with local variable x, and Q'_e an initial indexed formula tree for $\overset{\varepsilon}{\zeta}(\lambda x\,.\,s, \lambda x\,.\,t)$. The application of the *extensionality introduction rule* on R_e results in a proof state $[Q', \sigma \rhd_{\mathcal{L}} R']$. Thereby Q' is the result of applying the extensionality introduction rule on Q_e which consisted in replacing Q_e by

$$
Q_{Ext} = \begin{array}{c} \mathbf{Label}(Q_e)^p_\alpha \\ \diagup \quad \diagdown \\ Q_e \qquad Q'_e \end{array}
$$

Furthermore R' is the result of replacing all literal nodes R_L in R that are annotated by Q_e with

$$
\begin{array}{c} {}^{p}_{\alpha}\alpha(\mathbf{Label}(R_L), \mathbf{Label}(R'_L)) \\ \diagup \quad \diagdown \\ R_L \qquad R'_L \end{array}
$$

where R'_L is an initial free variable indexed formula tree for Q'_e. ■

Example 5.3.24 Consider the sample initial proof state shown in Figure 5.2 (p. 71). The introduction of extensionality for the ε-type node $\bar{}_{\varepsilon} X' + Y' = \mathrm{s}(\mathrm{p}(X') + Y')$ yields the indexed formula tree shown in Figure 4.3 (p. 45) and the free variable indexed formula tree shown in Figure 5.7.

Lemma 5.3.25 The extensionality introduction rule is sound and safe. ■

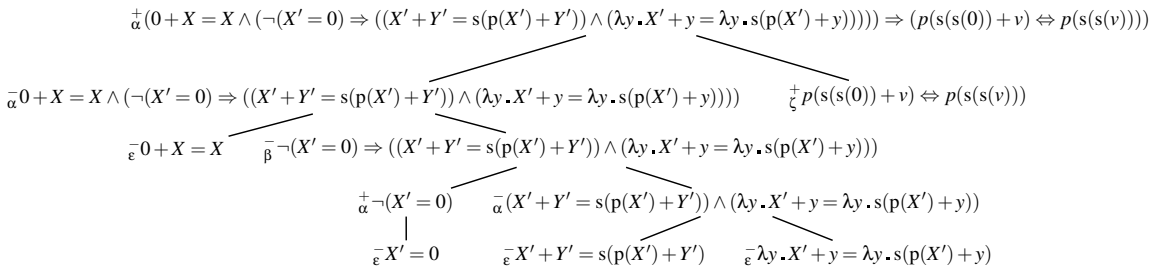


Figure 5.7: Free variable indexed formula tree after extensionality introduction for ε-type formula $\overset{-}{\varepsilon}X'+Y'=\mathrm{s}(\mathrm{p}(X')+Y')$ with respect to γ-local variable Y'.

Proof. *The label of R_L is the prefixed formula $pre(R_L).\overset{\xi}{\zeta}{}^{p}(s,t)$ where s,t are of type τ. This prefixed formula is replaced by the formula $pre(R_L).\alpha^{p}(\overset{\xi}{\zeta}{}^{p}(s,t),\overset{\xi}{\zeta}{}^{p}(\lambda x\,.\,s,\lambda x\,.\,t))$. Let M^{ρ}_{w} be an $\mathcal{L}$-model with possible world w and assignment ρ that satisfies $pre(R_L).\overset{\xi}{\zeta}{}^{p}(s,t)$. We prove the soundness and safeness for the case where $\overset{\xi}{\zeta}{}^{p}(s,t)$ is of the form $(s=t)^{+}$. The other cases are analogous.*

For the case $(s=t)^{+}$ the variable x must be δ-local with respect to $(s=t)^{+}$. Then it holds:

$$
\begin{array}{ll}
 & M^{\rho}_{w}\models_{\mathcal{L}} pre(R_L).(s=t)^{+} \\
\Leftrightarrow & \text{for all } w'\in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'}\models_{\mathcal{L}}(s=t)^{+} \\
\Leftrightarrow & \text{for all } w'\in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'}\models_{\mathcal{L}} s=t^{+} \text{ and } M^{\rho}_{w'}\models_{\mathcal{L}} s=t^{+} \\
\overset{x\ \delta-local\ \&\ Lemma\ 4.3.2}{\Leftrightarrow} & \text{for all } w'\in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'}\models_{\mathcal{L}} s=t^{+} \text{ and } M^{\rho}_{w'}\models_{\mathcal{L}}(\forall x_{\tau'}\,.\,s=t)^{+} \\
\overset{Extensionality}{\Leftrightarrow} & \text{for all } w'\in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'}\models_{\mathcal{L}} s=t^{+} \text{ and } M^{\rho}_{w'}\models_{\mathcal{L}}(\lambda x_{\tau'}\,.\,s=\lambda x_{\tau'}\,.\,t)^{+} \\
\Leftrightarrow & \text{for all } w'\in M^{\rho}_{w}(pre(R_L))\ M^{\rho}_{w'}\models_{\mathcal{L}}\alpha^{+}(s=t^{+},(\lambda x_{\tau'}\,.\,s=\lambda x_{\tau'}\,.\,t)^{+}) \\
\Leftrightarrow & M^{\rho}_{w}\models_{\mathcal{L}} pre(R_L).\alpha^{+}(s=t,\lambda x_{\tau'}\,.\,s=\lambda x_{\tau'}\,.\,t)
\end{array}
$$

□

5.3.8 Boolean ζ-Expansion

Definition 5.3.26 (Boolean ζ-Expansion Rule) Let $[Q,\sigma\rhd_{\mathcal{L}}R]$ be a proof state, R_ζ a ζ-type subtree in R, and Q_ζ its associated subtree in Q of label $\zeta(A_o,B_o)$. The application of the *boolean ζ-expansion rule* on R_ζ results in a proof state $[Q',\sigma\rhd_{\mathcal{L}}R']$, where Q' is the result of applying the boolean ζ-expansion rule on Q_ζ which introduces an intitial indexed formula tree Q_E for $((A\Rightarrow B)\wedge(B\Rightarrow A))^{+}$. Furthermore, R' is obtained by replacing all literal nodes R_L of R annotated by Q_ζ by

$$
\begin{array}{c}
\overset{p}{\alpha}\alpha(\mathbf{Label}(R_L),\mathbf{Label}(R'_L)) \\
\diagup\quad\diagdown \\
R_L\qquad R'_L
\end{array}
$$

where R'_L is an initial free variable indexed formula tree for Q_E. ■

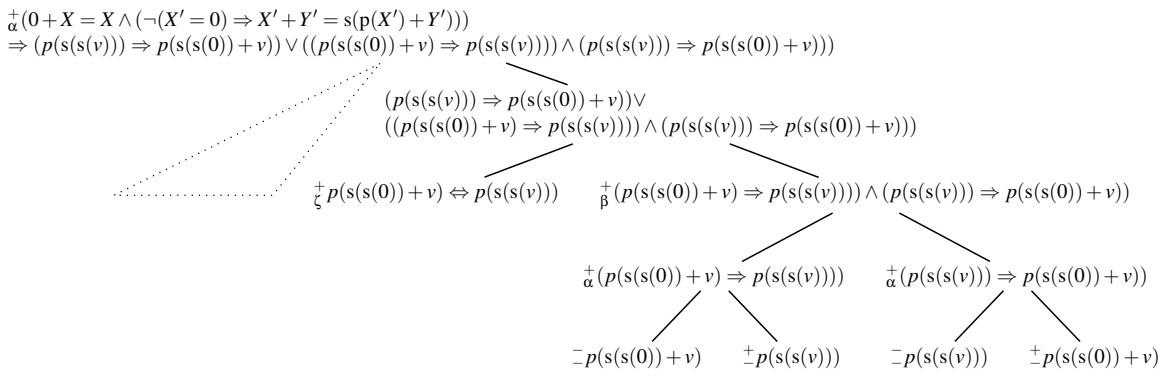


Figure 5.8: Free variable indexed formula tree after boolean ζ-expansion on $\overset{+}{\zeta}p(\mathrm{s}(\mathrm{s}(0))+v)\Leftrightarrow p(\mathrm{s}(\mathrm{s}(v)))$.

Example 5.3.27 Take as an example the proof state from Figure 5.2 (p. 71). The application of the boolean ζ-expansion rule applied to $\overset{+}{\zeta}p(\mathrm{s}(\mathrm{s}(0))+v)\Leftrightarrow p(\mathrm{s}(\mathrm{s}(v)))$ yields the indexed formula tree already shown in Figure 4.4 (p. 46) and the free variable indexed formula tree of Figure 5.8.

Lemma 5.3.28 The boolean ζ-expansion rule is sound and safe. ∎

Proof. The rule operates on literal nodes and thus does not affect the substitution which remains $\mathcal{L}$-admissible. It remains to be shown for soundness (respectively safeness) that the rule preserves the existence (respectively absence) of satisfiable paths. The rule is applied on a literal node $R_\zeta = \zeta(A,B)$, which is of positive polarity and where A and B are formulas. The new node introduced by that rule denotes the signed formula $((A\Rightarrow B)\wedge(B\Rightarrow A))^+$, which using uniform notation is $\beta(\alpha(A^-,B^+)^+,\alpha(B^-,A^+)^+)^+$. The rule transforms the paths as follows:

$$\ll\Gamma,\zeta(A,B)\gg \quad \textit{into} \quad \ll\Gamma,\zeta(A,B),\beta(\alpha(A^-,B^+)^+,\alpha(B^-,A^+)^+)^+\gg$$

We show for that transformation that the former path is satisfiable if, and only if, the latter path is satisfiable.

$$\begin{array}{ll}
 & M^{\mathfrak{p}}_w\models_{\mathcal{L}} pre(R_\zeta).\zeta(A_{\mathrm{o}},B_{\mathrm{o}})^+\\
\Leftrightarrow & \text{for all } w'\in M^{\mathfrak{p}}_w(pre(R_\zeta))\ M^{\mathfrak{p}}_{w'}\models_{\mathcal{L}}\zeta(A_{\mathrm{o}},B_{\mathrm{o}})\\
\overset{\text{Definition 3.4.2}}{\Leftrightarrow} & \text{for all } w'\in M^{\mathfrak{p}}_w(pre(R_\zeta))\ M^{\mathfrak{p}}_{w'}\not\models_{\mathcal{L}}A=B\\
\Leftrightarrow & \text{for all } w'\in M^{\mathfrak{p}}_w(pre(R_\zeta))\ M^{\mathfrak{p}}_{w'}(A)\neq M^{\mathfrak{p}}_w(B)\\
\overset{M^{\mathfrak{p}}_w(\mathrm{o})=\{\top,\bot\}}{\Leftrightarrow} & \text{for all } w'\in M^{\mathfrak{p}}_w(pre(R_\zeta))\ (M^{\mathfrak{p}}_{w'}(A)=\top \text{ and } M^{\mathfrak{p}}_{w'}(B)=\bot) \text{ or}\\
 & (M^{\mathfrak{p}}_{w'}(A)=\bot \text{ and } M^{\mathfrak{p}}_{w'}(B)=\top)\\
\Leftrightarrow & \text{for all } w'\in M^{\mathfrak{p}}_w(pre(R_\zeta))\ (M^{\mathfrak{p}}_{w'}\models_{\mathcal{L}}A \text{ and } M^{\mathfrak{p}}_{w'}\not\models_{\mathcal{L}}B) \text{ or}\\
 & (M^{\mathfrak{p}}_{w'}\not\models_{\mathcal{L}}A \text{ and } M^{\mathfrak{p}}_{w'}\models_{\mathcal{L}}B)\\
\overset{\text{Definition 3.4.2}}{\Leftrightarrow} & \text{for all } w'\in M^{\mathfrak{p}}_w(pre(R_\zeta))\ (M^{\mathfrak{p}}_{w'}\models_{\mathcal{L}}A^- \text{ and } M^{\mathfrak{p}}_{w'}\models_{\mathcal{L}}B^+) \text{ or}\\
 & (M^{\mathfrak{p}}_{w'}\models_{\mathcal{L}}A^+ \text{ and } M^{\mathfrak{p}}_{w'}\models_{\mathcal{L}}B^-)\\
\overset{\text{Lemma 3.4.3}}{\Leftrightarrow} & \text{for all } w'\in M^{\mathfrak{p}}_w(pre(R_\zeta))\ M^{\mathfrak{p}}_{w'}\models_{\mathcal{L}}\alpha(A^-,B^+)^+ \text{ or } M^{\mathfrak{p}}_{w'}\models_{\mathcal{L}}\alpha(B^-,A^+)^+\\
\overset{\text{Lemma 3.4.3}}{\Leftrightarrow} & \text{for all } w'\in M^{\mathfrak{p}}_w(pre(R_\zeta))\ M^{\mathfrak{p}}_{w'}\models_{\mathcal{L}}\beta(\alpha(A^-,B^+)^+,\alpha(B^-,A^+)^+)^+\\
\Leftrightarrow & M^{\mathfrak{p}}_w\models_{\mathcal{L}} pre(R_\zeta).\beta(\alpha(A^-,B^+)^+,\alpha(B^-,A^+)^+)^+
\end{array}$$

□

5.3.9 Instantiation

Let $[Q, \sigma \rhd_{\mathcal{L}} R]$ be a proof state. The instantiation rule extends the actual substitution σ by applying an $\mathcal{L}$-substitution σ' such that the resulting overall $\mathcal{L}$-substitution $\sigma' \circ \sigma$ is $\mathcal{L}$-admissible. Generally, an $\mathcal{L}$-substitution consists of an object variable substitution σ'_Q and a modal substitution σ'_M. The modal substitution has no direct effect on Q, except for the determination of the equality of prefixes and for the overall ordering. The object variable substitution affects Q by applying the substitution to the labels of the non-literal nodes and by replacing literal nodes Q_L in Q by an initial indexed formula tree $Q_L^{\sigma'}$ for $\sigma'_Q(\mathbf{Label}(Q_L))$. Each such Q_L may be associated to one or more literal nodes in R. Thus, while replacing Q_L by $Q_L^{\sigma'}$ we need to replace the corresponding literal nodes in R by initial free variable indexed formula trees for $Q_L^{\sigma'}$.

Definition 5.3.29 (Instantiation Rule) Let $[Q, \sigma \rhd_{\mathcal{L}} R]$ be a proof state, and σ' an $\mathcal{L}$-substitution such that $\sigma' \circ \sigma$ is $\mathcal{L}$-admissible. The instantiation rule results in a proof state $[Q', \sigma' \circ \sigma \rhd_{\mathcal{L}} R']$ where Q' results from Q as defined in Definition 4.5.2 and R' results from R by applying the substitution to the non-literal nodes and by replacing all literal nodes in R with associated literal node Q_L in Q that have been replaced in Q' by some $Q_L^{\sigma'}$ with an initial free variable indexed formula tree for $Q_L^{\sigma'}$. ■

Lemma 5.3.30 The instantiation rule is sound. ■

Proof. The instantiation rule is sound, since the admissibility of the substitution is checked on Q (resp. Q'). It also preserves the existence of $\mathcal{L}$-satisfiable paths, as is proved as follows: let M be an $\mathcal{L}$-model of the whole signed formula φ in Q before application of the substitution σ'. If M satisfies φ^p then for all possible worlds w and all assignments ρ' it holds: $M_w^{\rho'} \models_{\mathcal{L}} \varphi^p$. From Lemma 4.1.14 it follows that for any assignment ρ and substitution σ' there is a assignment ρ' such that

$$M_w^{\rho} \models_{\mathcal{L}} \sigma'(\varphi)^p \Longrightarrow M_w^{\rho'} \models_{\mathcal{L}} \varphi^p$$

Hence, M $\mathcal{L}$-satisfies $\sigma'(\varphi)^p$. □

New Variables. Higher-order unification may generate new variables. In Section 4.6 we showed how these additional variables are handled inside Q. Thus the integration of new variables changes a proof state $[Q, \sigma \rhd_{\mathcal{L}} R]$ to $[Q', \sigma \rhd_{\mathcal{L}} R]$ where Q' contains the additional bindings for the new variables.

5.3.10 Increase of Multiplicities

The increase of multiplicities is necessary for a *safe* instantiation. This is achieved by increasing the multiplicities of the parent nodes of the binding nodes of instantiated object variables and variable nodes, before applying a substitution. In Section 4.11 we showed how the multiplicities are increased in the indexed formula tree Q. From there we obtain the subtrees of primary type $\mathbf{V}_0$ or Π_0 that have been obtained by copying, as well as a mapping ι from nodes in the old subtrees to nodes in the new subtree and a renaming θ of object level variables. Assume the overall modal substitution is σ_M: then propagating the new multiplicities into the free variable indexed formula tree R is achieved by considering the subtrees in R that have an associated node Q' in $dom(\iota)$ or where $\sigma_M(Q') \cap dom(\iota) \neq \emptyset$. Those subtrees are either literal nodes or subtrees of type ν with associated node Q' such that $\sigma_M(Q') \cap dom(\iota) \neq \emptyset$. Thus, we need to copy those subtrees of type ν by renaming the references into Q by ι and the object level variables by θ. For the respective literal nodes in R that do not occur in one of these subtrees we consider the maximal subtrees R that contain only those kind of literal nodes. The maximal subtrees are also copied by applying the renamings θ and ι.

Definition 5.3.31 (Increase of Multiplicities) Let $[Q, \sigma \rhd_{\mathcal{L}} R]$ be a proof state, and Q a set of subtrees from Q of which to increase the multiplicities. The new proof state $[Q', \sigma' \rhd_{\mathcal{L}} R']$ is obtained by

- increasing the multiplicities in Q according to Definition 4.11.5 which results in Q', a variable renaming θ and a mapping ι on subtrees of the indexed formula tree Q'.
- Let R_M be the maximal subtrees that have an associated node of type ν_0 in $dom(\iota)$ and R_L the maximal subtrees that contain only literal nodes in $dom(\iota)$ and that do not occur in R_M. For each subtree $R_0 \in R_M \cup R_L$ we α-insert a copy of R_0 that has been renamed with respect to θ and ι. ■

Lemma 5.3.32 The increase of multiplicities is sound and safe. ■

Proof. *By Lemma 4.11.7 the new overall substitution that results from instantiation required for the increase of multiplicities is still $\mathcal{L}$-admissible. Now consider the maximal subtrees $R_0 \in R_M \cup R_L$ in R that have been copied. Each such subtree has a label $pre(R_0).\varphi^p$ and has been replaced by a subtree of the same prefix and label $pre(R_0).\alpha^p(\varphi^p, \theta(\varphi)^p)$. Since θ is only a renaming of variables it trivially holds for every $\mathcal{L}$-model M, possible world w and assignment ρ there exists a ρ' such that*

$$M_w^{\rho} \models_{\mathcal{L}} pre(R_0).\varphi^p \Longrightarrow M_w^{\rho'} \models_{\mathcal{L}} pre(R_0).\alpha^p(\varphi^p, \theta(\varphi)^p)$$

Conversely, for every $\mathcal{L}$-model M, possible world w and assignment ρ' it follows from Lemmata 3.4.3 and 4.1.8 that

$$M_w^{\rho'} \models_{\mathcal{L}} pre(R_0).\alpha^p(\varphi^p, \theta(\varphi)^p) \Longrightarrow M_w^{\rho'} \models_{\mathcal{L}} pre(R_0).\varphi^p$$

□

5.3.11 Application of Rewriting Replacement Rules

A resolution replacement rule is always applied to a node of the free variable indexed formula tree, if the left-hand side of the rule is connectable to that node. A rewriting replacement rule can also be applied to some node, but may also be virtually applied to some subterm of the label of a literal node. For the application of a rewrite replacement rule to some node the rewriting step is encoded internally by a combination of the Leibniz' equality introduction and the instantiation rule. We call this kind of rewriting on nodes *boolean rewriting*, since it consists of the application of an equivalence $A \Leftrightarrow B$.

As already presented in Section 3.5, rewriting inside labels of literal nodes also relies on this combination of Leibniz' equality and instantiation, but due to the presence of quantifiers or λ-binders inside labels, it may require an additionally application of the extensionality rule.

Since the rewriting "inside" literal nodes relies on extensionality introduction and rewriting on nodes we first define the rewriting on nodes.

Definition 5.3.33 (Rewriting Replacement Rule Application On Nodes) Let $[Q, \sigma \rhd_{\mathcal{L}} R]$ be a proof state, a a node in R of polarity p and let $u \to \langle v, v'_1, \ldots, v'_n \rangle$ $(n \geq 0)$ be an admissible rewriting replacement rule for a, where u and v are the left- and right-hand sides of an ε-type position v_0. The application of $u \to \langle v, v'_1, \ldots, v'_n \rangle$ to a is defined as follows:

1. Apply the Leibniz' equality introduction rule to v_0 to obtain

$$\beta^-(P(\mathbf{Label}(v))^p, P(\mathbf{Label}(u))^{-p}),$$

2. Instantiate P by $\lambda x \,.\, x$ to obtain $\beta^-(\mathbf{Label}(v)^p, \mathbf{Label}(u)^{-p})$, which results in the resolution replacement rule
$$\mathbf{Label}(u)^{-p} \to \langle \mathbf{Label}(v)^p, v_1, \ldots, v_n \rangle .$$
3. Apply $\mathbf{Label}(u)^{-p} \to \langle \mathbf{Label}(v)^p, v_1, \ldots, v_n \rangle$ to a. ■

Lemma 5.3.34 The rewriting style replacement rule application *on nodes* is sound. ■

Proof. The Leibniz' equality introduction rule is sound and safe. The resolution style rule application is sound. □

As a motivating example for rewriting inside literal nodes we refer to the example from Section 3.5. We first state the definition for rewriting inside literal nodes and afterwards give an example that illustrates the different steps of the definition.

Definition 5.3.35 (Rewriting Replacement Rule Application Inside Literal Nodes) Let $[Q, \sigma \rhd_{\mathcal{L}} R]$ be a proof state, a a literal node of label φ in R and of polarity p, π a valid subterm occurrence inside $\mathbf{Label}(a)$, and let $u \to \langle v, v'_1, \ldots, v'_n \rangle$ $(n \geq 0)$ be an admissible rewriting replacement rule for a, where u and v are the left- and right-hand sides of an ε-type position v_0 of label $\varepsilon^-(s,t)$. The application of $u \to \langle v, v'_1, \ldots, v'_n \rangle$ on a at π is defined as follows:

- Let $x_1, \ldots, x_n$ be the variables that are free in $\varphi_{|\pi}$, but not in φ. Let further be σ' a substitution such that $\sigma'(s) = \sigma'(\varphi_{|\pi})$ and $x_i \notin dom(\sigma'), 1 \leq i \leq n$.
- Let $D := \{X \in dom(\sigma') \mid \exists x_i \,.\, x_i \in \sigma'(X)\}$ be the variables that are instantiated with a term in which occurs one of the x_i. Then σ' is partitioned into two disjunct substitutions defined by
$$\sigma'_1 := \sigma'_{|D} \text{ and } \sigma'_2 := \sigma'_{|dom(\sigma') \setminus D}$$
- Apply the extensionality introduction rule on v_0 for the variables in D to obtain v'_0 of label $\lambda y_1 \,.\, \ldots \lambda y_n \,.\, s = \lambda y_1 \,.\, \ldots \lambda y_n \,.\, t$.

  If this fails the rule application fails.
- Otherwise apply the Leibniz' equality introduction on v'_0 to obtain the formula
$$\gamma^p P \,.\, \beta^p (P(\lambda y_1 \,.\, \ldots \lambda y_n \,.\, s)^{-p}, P(\lambda y_1 \,.\, \ldots \lambda y_n \,.\, t)^p).$$
  This results in the resolution replacement rule
$$P(\lambda y_1 \,.\, \ldots \lambda y_n \,.\, s)^{-p} \to \langle P(\lambda y_1 \,.\, \ldots \lambda y_n \,.\, t)^p, v_1, \ldots, v_n \rangle .$$
- Apply the substitution $\{\lambda f \,.\, \varphi_{|\pi \leftarrow f(\sigma'_1(y_n), \ldots, \sigma'_1(y_1))} / P\} \circ \sigma'_2$.
- Apply the (instantiated) resolution replacement rule. ■

Lemma 5.3.36 The application of a rewriting replacement rule *inside nodes* is sound. ■

Proof. Since the application of rewriting replacement rules is a combination of other CORE *calculus rules, it is sound.* ■

As an example to illustrate the rewriting inside literal nodes consider the following higher-order literal $Q(\lambda x_{Nat} \,.\, s(s(x)) + y)^-$, where Q is a predicate, s is the successor function on natural numbers and $+$ addition on natural numbers. For the rewriting consider the formula $\forall u, v \,.\, s(u) + v = s(u + v)$, which gives rise to the rewrite replacement rule $s(u) + v \rightarrow \langle s(u + v) \rangle$. The application of the rule to the literal at the subterm $s(s(x) + y)$ is achieved as follows:

1. The bound variable in $Q(\lambda x \,.\, s(s(x)) + y)$ is x, and the substitution σ' is $\{s(x)/u, v/y\}$.
2. Then $D := \{u\}$, $\sigma'_1 := \{s(x)/u\}$, and $\sigma'_2 := \{v/y\}$.
3. The application of the extensionality introduction rule over $u \in D$ on $s(u) + v = s(u + v)$ results in $\lambda u \,.\, s(u) + v = \lambda u \,.\, s(u + v)$.
4. The Leibniz' equality introduction on $\lambda u \,.\, s(u) + v = \lambda u \,.\, s(u + v)$ results in $\forall P \,.\, P(\lambda u \,.\, s(u) + v) \Rightarrow P(\lambda u \,.\, s(u + v))$.
5. The γ-variable P is instantiated with $\lambda f \,.\, Q(\lambda x \,.\, f(s(x)))$ which results in $Q(\lambda x \,.\, s(s(x)) + v) \Rightarrow Q(\lambda x \,.\, s(s(x) + v))$.
6. Finally σ'_2 is applied which results in $Q(\lambda x \,.\, s(s(x)) + y) \Rightarrow Q(\lambda x \,.\, s(s(x) + y))$, which yields the required resolution replacement rule $Q(\lambda x \,.\, s(s(x)) + y) \rightarrow \langle Q(\lambda x \,.\, s(s(x) + y)) \rangle$. The rule is applicable on $Q(\lambda x \,.\, s(s(x)) + y)$ and results in $Q(\lambda x \,.\, s(s(x) + y))$.

5.3.11.1 Limitations of Replacement Rule Rewriting

Although the previous rewriting with replacement rules supports rewriting below bindings of variables, it still is limited. As an example consider the conditional variant $\forall u, v \,.\, u \neq 0 \Rightarrow u + v = s(p(u) + v)$, where p denotes the predecessor function on natural numbers, instead of the equation $\forall u, v \,.\, s(u) + v = s(u + v)$. This axiom results in the conditional rewriting replacement rule $[u \neq 0]\ u + v \rightarrow \langle s(p(u) + v) \rangle$. Applying this rule results in the following steps:

1. The bound variable in $Q(\lambda x \,.\, s(s(x)) + y)$ is x, and the substitution σ' is $\{s(s(x))/u, v/y\}$.
2. Then $D := \{u\}$, $\sigma'_1 := \{s(s(x))/u\}$, and $\sigma'_2 := \{v/y\}$.

The next step that consists of the extensionality introduction over u fails, since u occurs in the condition $u \neq 0$ and thus is not γ-local to $u + v = s(p(u) + v)$. Thus, the application of the replacement rule fails. A way to look at this problem is that it fails because there is no way of moving the condition of the replacement rule below the binder of x.

One possibility to remedy this problem is to use for each type τ an "if-then-else" function $C_{o \times \tau \times \tau \rightarrow \tau}$ (cf. [Andrews, 2002], p. 235). Having this function would integrate the condition of the rule below the binder of x and in the above example the result of the replacement rule application would be:

$$Q(\lambda x \,.\, C(s(s(x)) \neq 0, s(s(x)) \neq 0, s(p(s(s(x))) + y), s(s(x)) + y))$$

Instead of using C we could also use the description operator $\iota : (\tau \rightarrow o) \rightarrow \tau$, for each type τ. The function C can be defined by the description operator as follows:

$$C(A, s, t) = \iota(\lambda y \,.\, (A \wedge y = s) \vee (\neg A \wedge y = t))$$

The description operator itself can be defined by the following axiom schemas:

$$\forall y_\tau . \iota(\lambda x_\tau . x = y) = y$$

for every type τ. Using the description operator, the result of the replacement rule application would be:

$$Q(\lambda x . \iota(\lambda z . (s(s(x)) \neq 0 \wedge z = s(p(s(s(x))) + y)) \vee (s(s(x)) = 0 \wedge z = s(s(x)) + y)))$$

Another solution that avoids the use of the description operator consists of removing the conditions of the equations by decomposing the problem

$$(u \neq 0 \Rightarrow u + v = s(p(u) + v)) \Rightarrow Q(\lambda x . s(s(x)) + y)$$

into

$$(u + v = s(p(u) + v) \Rightarrow Q(\lambda x . s(s(x)) + y)) \wedge (u \neq 0 \wedge Q(\lambda x . s(s(x)) + y))$$

and subsequently applying the unconditional equation $u + v = s(p(u) + v)$ on $Q(\lambda x . s(s(x)) + y)$, which yields

$$(u + y = s(p(u) + y) \Rightarrow Q(\lambda x . s(p(s(s(x))) + y))) \wedge (u \neq 0 \wedge Q(\lambda x . s(s(x)) + y))$$

The CORE calculus rules do not support that kind of decomposition directly. However, in Chapter 10 we present such a β-decomposition rule which is used in [Schütte, 1977] and prove that it is an admissible rule in the CORE calculus. Thus, it can be used to perform this kind of β-decomposition.

5.3.12 Cut

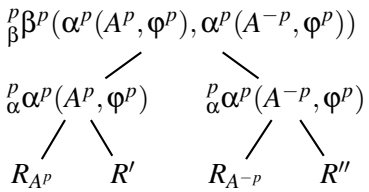
The cut rule is the basis for different kinds of reasoning steps like speculative proof steps, lemma introduction, proof by contradiction or case analysis. It consists of replacing a prefixed formula $w.\varphi$ by

$$w.\beta^p(\alpha^p(A^p, \varphi^p), \alpha^p(A^{-p}, \varphi^p)).$$

The new occurrences of φ^p are simple copies of the old subtree. The subtrees for A^p and A^{-p} are initial free variable indexed formula trees for the respective subtrees in the indexed formula tree for $\beta^p(A^p, A^{-p})$ that has been α-inserted in the corresponding indexed formula tree Q to represent the cut. The problem here is to determine where the cut must actually be performed in Q. The problem arises since the free variable indexed formula tree φ^p usually does not correspond to a single subtree in Q, but has been constructed by replacement rule applications. The exact position of the cut formula however cannot be determined yet, as it depends on how the parts of that cut formula are used later-on in the proof, i.e. which parts of A are "connected" to parts in φ^p. Thus, we follow a defensive approach and assume any part of A can in principle be connected to any part in φ^p. Technically speaking, this implies that we determine all subtrees of Q that are referenced in φ^p – either by literal nodes or ν- or π-type nodes – and determine the smallest subtree in Q that contains all these subtrees. That subtree is then used to actually perform the cut over A.

Definition 5.3.37 (Cut Rule) Let $[Q, \sigma \rhd_{\mathcal{L}} R]$ be a proof state, and let R' be a subtree of R with polarity p and label φ, and A a formula. Assume further, that Q' is the smallest subtree of Q that contains all subtrees referenced in R'. The cut over A on R' results in a new proof state $[Q^*, \sigma^* \rhd_{\mathcal{L}} R^*]$,

where Q^* and σ^* result from the cut over A on Q' in Q (cf. Definition 4.7.1). From there two free variable indexed formula trees R_{A^p} and $R_{A^{-p}}$ of respective signed labels A^p and A^{-p} are constructed from the initial indexed formula trees for A^p and A^{-p}. Finally R^* is obtained from R by replacing the subtree R' with the subtree



where R'' is a copy of R'. ∎

Lemma 5.3.38 The cut rule is sound and safe. ∎

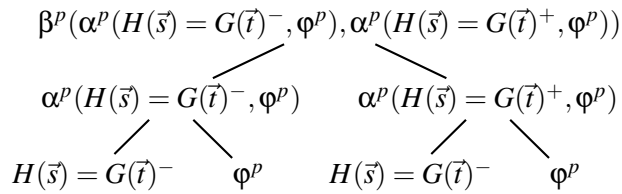
Proof. The $\mathcal{L}$-admissibility of σ^* with respect to Q^* has been shown in Lemma 4.7.3. It remains to prove that the prefixed formula $pre(R').\varphi^p$ is $\mathcal{L}$-satisfiable, if, and only if, the prefixed formula $pre(R').\beta^p(\alpha^p(A^p,\varphi^p),\alpha^p(A^{-p},\varphi^p))$ is $\mathcal{L}$-satisfiable. Let M be an $\mathcal{L}$-model, w a possible world and ρ an assignment. Then it holds:

$$
\begin{array}{ll}
 & M^\rho_w \models_{\mathcal{L}} pre(R').\beta^p(\alpha^p(A^p,\varphi^p),\alpha^p(A^{-p},\varphi^p)) \\
\Leftrightarrow & \text{for all } w' \in M^\rho_w(pre(R'))\ M^\rho_{w'} \models_{\mathcal{L}} \beta^p(\alpha^p(A^p,\varphi^p),\alpha^p(A^{-p},\varphi^p)) \\
\Leftrightarrow & \text{for all } w' \in M^\rho_w(pre(R'))\ M^\rho_{w'} \models_{\mathcal{L}} \alpha^p(A^p,\varphi^p) \text{ or } M^\rho_{w'} \models_{\mathcal{L}} \alpha^p(A^{-p},\varphi^p) \\
\Leftrightarrow & \text{for all } w' \in M^\rho_w(pre(R'))\ (M^\rho_{w'} \models_{\mathcal{L}} A^p \text{ and } M^\rho_{w'} \models_{\mathcal{L}} \varphi^p) \\
 & \text{or } (M^\rho_{w'} \models_{\mathcal{L}} A^{-p} \text{ and } M^\rho_{w'} \models_{\mathcal{L}} \varphi^p) \\
\Leftrightarrow & \text{for all } w' \in M^\rho_w(pre(R'))\ (M^\rho_{w'} \models_{\mathcal{L}} A^p \text{ and } M^\rho_{w'} \models_{\mathcal{L}} \varphi^p) \\
 & \text{or } (M^\rho_{w'} \not\models_{\mathcal{L}} A^p \text{ and } M^\rho_{w'} \models_{\mathcal{L}} \varphi^p) \\
\Leftrightarrow & \text{for all } w' \in M^\rho_w(pre(R'))\ (M^\rho_{w'} \models_{\mathcal{L}} A^p \text{ or } M^\rho_{w'} \not\models_{\mathcal{L}} A^p) \text{ and } M^\rho_{w'} \models_{\mathcal{L}} \varphi^p) \\
\Leftrightarrow & \text{for all } w' \in M^\rho_w(pre(R'))\ M^\rho_{w'} \models_{\mathcal{L}} \varphi^p \\
\Leftrightarrow & M^\rho_w \models_{\mathcal{L}} pre(R').\varphi^p
\end{array}
$$

□

Flex-Flex Constraints. Higher-order unification may generate flex-flex constraints. In Section 4.9 we presented how those are integrated via the cut-rule into the indexed formula tree. These constraints arise during higher-order unification, and are directly related to the application of a replacement rule in order to enable its application. The exact subtree on which the constraints need to be inserted is thus determined by the subtree a of the free variable indexed formula tree on which the replacement rule is applied.

Assume the flex-flex constraint is $H(\vec{s}) = G(\vec{t})$ and the signed label of the free variable indexed formula tree a is φ^p. To introduce that constraint we perform a cut over that constraint formula which replaces the free variable indexed formula tree a by



This allows to apply the rewriting replacement rule that results from $H(\vec{s}) = G(\vec{t})^-$ on the left occurrence of φ^p, in order enable the application of the actual replacement rule on that formula[2]. The

[2]If there are more than one flex-flex constraint, this process needs to be iterated to integrate all flex-flex constraints before applying the actual replacement rule.

occurrence of $H(\vec{s}) = G(\vec{t})^+$ represents the actual new goal that consists of proving the flex-flex constraints.

5.4 Completeness

In this section we prove the completeness of the calculus consisting of contraction, weakening, structural modal permutation, replacement rule applications, simplification, increase of multiplicity, instantiation, Leibniz' equality introduction, extensionality introduction, boolean ζ-expansion, and cut.

Theorem 5.4.1 (Completeness) Let φ be an $\mathcal{L}$-formula, Q an initial indexed formula tree for φ^+, R an initial free variable indexed formula tree for Q, and Id the empty substitution. If φ is $\mathcal{L}$-valid then there is a CoRE derivation

$$[Q, Id \rhd_{\mathcal{L}} R] \longmapsto^* [Q', \sigma \rhd_{\mathcal{L}} {}^{+}_{-} True]$$

■

Proof. *The completeness proof relies on the soundness and completeness results of Theorem 4.12.1 which is due to [Wallen, 1990, Andrews, 1989, Pfenning, 1987]. The proof sketch is as follows: from Theorem 4.12.1 we assume that we have guessed the right multiplicities for γ- and ν-type nodes, the right combined substitution σ, the necessary introductions of Leibniz' equality, extensionality introductions, boolean ζ-expansions, cut, and have moved any $\Box$ and $\Diamond$-quantifier in front of literal nodes using the structural modal permutation rule. All paths in the resulting free variable indexed formula tree R_P are (propositionally) $\mathcal{L}$-unsatisfiable. That is from $[Q, Id \rhd_{\mathcal{L}} R]$ we can derive a proof state $[Q_P, \sigma \rhd_{\mathcal{L}} R_P]$. In a second phase we have to prove that from $[Q_P, \sigma \rhd_{\mathcal{L}} R_P]$ we can derive $[Q_P, \sigma \rhd_{\mathcal{L}} {}^{+}_{-} True]$. The problem $[Q_P, \sigma \rhd_{\mathcal{L}} R_P]$ is essentially propositional, since all necessary substitutions have already been applied. In this second phase we prove that the combination of the contraction rule, resolution replacement rule application, and simplification allows us to simulate the* path resolution *rule from [Murray & Rosenthal, 1987a]. Since* path resolution *is complete the* CoRE *calculus is also complete. However, while path resolution derives an empty subgraph, we show that in* CoRE *we obtain the final proof state $[Q_P, \sigma \rhd_{\mathcal{L}} {}^{+}_{-} True]$. The technical proof is presented in Appendix A.* □

Note that cut is only necessary for completeness for the case of higher-order logic with Henkin semantics. For any other logic the simulation of the extensionality rule from [Pfenning, 1987] is not necessary to establish completeness, which is the only case that actually requires the cut rule. We discuss the issue of cut elimination in that case in the following Section 5.5.

5.5 A Note on Cut Elimination

We have already discussed that cut is not necessary for all logics but higher-order logic with Henkin semantics. We briefly discuss the fact that cut elimination is probably impossible in the CoRE calculus without loosing completeness of the calculus with respect to that logic. In [Benzmüller *et al*, 2002b] it is shown that for higher-order logics with Henkin semantics together with the rules ξ, *b* (cf. Figure 5.9) and the $\beta\eta$-normalisation rule, cut elimination is impossible[3] (cf. Example 2.2.17 in [Benzmüller *et al*, 2002b]). In the CoRE calculus we assume all terms to be in $\beta\eta$ normal form, so $\beta\eta$-normalisation is built-in. Furthermore, the ξ-rule corresponds to the CoRE extensionality introduction rule and the

[3] Actually [Benzmüller *et al*, 2002b] shows the result for a rule *f* instead of ξ and $\beta\eta$-normalisation. However, *f* is admissible in the presence of ξ and $\beta\eta$-normalisation rules.

$$\frac{\Gamma \vdash \forall X \,.\, M = N, \Delta}{\Gamma \vdash \lambda X \,.\, M = \lambda X \,.\, N, \Delta}\ \xi \qquad\qquad \frac{\Gamma, A \vdash B, \Delta \quad \Gamma, B \vdash A, \Delta}{\Gamma \vdash A_o = B_o, \Delta}\ b$$

Figure 5.9: Sequent calculus ξ- and b-rules from [Benzmüller *et al*, 2002b].

b-rule corresponds to the ζ-expansion rule on booleans. The alternative to the cut-rule presented in [Benzmüller *et al*, 2002b] is to use the following two rules:

$$\frac{\Gamma, A \vdash A = B, B, \Delta}{\Gamma, A \vdash B, \Delta}\ \text{Init}^{=} \qquad \frac{\Gamma \vdash A_1 = B_1, \Delta \quad \ldots \quad \Gamma \vdash A_n = B_n, \Delta}{\Gamma \vdash (hA_1, \ldots A_n) = (hB_1, \ldots B_n), \Delta}\ \text{dec}^{h}$$

We conjecture that the respective CoRe counterparts of both rules are not admissible in the cut free CoRe calculus, and thus cut elimination is not possible in CoRe. For future work we propose to investigate this question in more detail. In particular, we propose to check whether employing counterparts of $\text{Init}^{=}$ and dec^{h} instead of the cut rule is sufficient for a Henkin complete and cut-free variant of the CoRe calculus.

5.6 Summary

The proof theory of the CoRe system, as presented in this chapter, is actually a meta proof theory as it encompasses a variety of logics. The central notion is that of a proof state which contains all the relevant information about the status of the proof. It consists of an indexed formula tree Q, the actual substitution σ, and a free variable indexed formula tree R. Q is an indexed formula tree (respectively an expansion tree proof) and this is used to represent the dependencies among variable and modal quantifiers. It forms the backbone of the proof theory with respect to soundness and ensures the admissibility of substitutions. The actual interface to the user and reasoning engines is R which is a free variable formula, the variables being bound in Q. The free variable formula is annotated with polarities and uniform types and this proof theoretic information is the basis for a uniform notion of the logical context of subformulas as well as replacement rules. The replacement rules can on the one hand be viewed as the operationalisation of assertion level proof steps and therefore support the proof development at the assertion level. On the other hand, from a logical point of view, they are generalised resolution and paramodulation rules, which is a suitable representation for automatic reasoning procedures.

The working copy is manipulated by the CoRe reasoning rules and the soundness proof ensures that if R can be transformed to $True^{+}$ (respectively $False^{-}$), then the initial conjecture holds. The CoRe calculus consists of 12 rules and its completeness has been proved for the class of logics under considerations. The set of calculus rules is minimal for the whole class of logics and in Section 5.5 we presented some evidence for the fact that cut elimination is not possible without loosing completeness for higher-order logic with Henkin semantics.

Although in principle the calculus enforces the reduction of the initial free variable indexed formula tree to $True^{+}$ (respectively $False^{-}$), we refrain from enforcing this in practice. Indeed, as pointed out in Section 1.2 of the introduction, the free variable indexed formula tree represents possible case splits by means of logical connectives. Thus, in practice, a proof state is proved, if the free variable indexed formula tree is trivially provable by simplification, which corresponds to the fact that all cases represented in the free variable indexed formula tree are proved.

Hierarchical Reasoning

–

Part III

Chapter 6

Window Inferencing

The proof theory introduced in Chapter 5 provides all necessary features to support sound and complete contextual reasoning. The proof state is always a formula, which meets the requirements sketched in the introduction (Section 1.1.3). However, so far, the open goals are a list with exactly one element, namely the formula representing the proof state. This signed formula, or rather the free variable indexed formula tree (FVIF-tree) of the proof state, contains all possible conjunctive subgoals and all alternatives. Indeed, consider a subtree of the whole FVIF-tree: then each subtree that is β-related to a given subtree is a conjunctively related side-goal. Analogously, each α-related subtree is an alternative goal. Thus, instead of dealing with one single formula, it should be possible to focus on β-related subtrees without actually decomposing the whole formula. Additionally, a focus mechanism would allow us to mimic the style of proof search enforced by standard sequent calculi. Moreover, in contrast to these calculi, keeping the whole FVIF-tree would allow us to undo decompositions by simply retracting the focus to top-level[1]. A detailed analysis of two presentationally different versions of a "same" proof is presented in Section 6.1, which further motivates the benefit of a focus mechanism.

This proof search technique of focusing and un-focusing is known as *window inferencing* (or *window inference* as it was originally called) [Robinson & Staples, 1993,Grundy, 1991,Staples, 1995]. In this chapter we define window inferencing on top of the CoRe reasoning rules and extend it to support the variety of reasoning rules provided by the underlying framework. However, while in [Robinson & Staples, 1993, Grundy, 1991, Staples, 1995] the reasoning on subparts of the formula gives rise to proof obligations, in our case the CoRe framework provides all the contextual reasoning capabilities required by window inference and no further proof obligations arise.

In Section 6.2 we define the notion of windows for subtrees of FVIF-trees and window trees to capture the hierarchical structure of windows. Based on these notions we define window proof states as an extension to CoRe proof states. In Section 6.3.1 we present the reasoning rules on windows that allow us to focus and un-focus on subparts of the FVIF-tree. Those rules only affect the hierarchy of windows represented in the window tree, but not the FVIF-tree. Finally, the actual window inference rules that affect the structure of the FVIF-tree are presented in Section 6.3.2. Thereby we define for each of the CoRe reasoning rules a corresponding window inference rule.

[1]Note, that this would be even possible after having performed some changes on the subtrees, i.e. after having transformed the focused subformula.

6.1 Motivation

As a motivation for focusing we present two proofs of the following theorem about sums of natural numbers: $\forall n \,.\, \sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2$. The two proofs are essentially the same, although their presentations differ. The first proof is presented in an intuitive, structured handwritting style.

Example Proof 6.1.1 of $\forall n \,.\, \sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2$**.** *The proof is by induction over n:*

Base Case $n = 0$**:**

1. *By* $n = 0$ *we obtain* $\sum_{i=1}^{0} i^3 = (\sum_{i=1}^{0} i)^2$.
2. *By definition of* $\sum$ *and square* (x^2) *we obtain* $0 = 0$

Induction Step $n = n' + 1$**:** *The induction hypothesis is* $\sum_{i=1}^{n'} i^3 = (\sum_{i=1}^{n'} i)^2$.

1. *By* $n = n' + 1$ *we obtain* $\sum_{i=1}^{n'+1} i^3 = (\sum_{i=1}^{n'+1} i)^2$.
2. *By definition of* $\sum$ *we obtain* $(n'+1)^3 + \sum_{i=1}^{n'} i^3 = ((n'+1) + \sum_{i=1}^{n'} i)^2$.
3. *By* $(a+b)^2 = a^2 + 2ab + b^2$ *we obtain* $(n'+1)^3 + \sum_{i=1}^{n'} i^3 = (n'+1)^2 + 2(n'+1)(\sum_{i=1}^{n'} i) + (\sum_{i=1}^{n'} i)^2$.
4. *By Ind. Hyp. it reduces to* $(n'+1)^3 = (n'+1)^2 + 2(n'+1)(\sum_{i=1}^{n'} i)$.
5. *By* $\sum_{i=1}^{n'} = \frac{n'(n'+1)}{2}$ *we obtain* $(n'+1)^3 = (n'+1)^2 + 2(n'+1)\frac{n'(n'+1)}{2}$
6. *And finally* $(n'+1)^3 = (n'+1)^3$ □

The same proof without explicit proof structure, i.e. where the proof state is contained in a single formula, is as follows:

Example Proof 6.1.2 of $\forall n \,.\, \sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2$**.** *To the goal we apply the (higher-order) induction axiom for natural numbers* $\forall P \,.\, ((\forall n \,.\, n = 0 \Rightarrow P(n)) \wedge \forall n, n' \,.\, (n = n' + 1 \wedge P(n'))) \Rightarrow \forall n \,.\, P(n))$ *and obtain*

$$\forall n \,.\, n = 0 \Rightarrow \sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2 \wedge \forall n, n' \,.\, n = n' + 1 \wedge \sum_{i=1}^{n'} i^3 = (\sum_{i=1}^{n'} i)^2 \Rightarrow \sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2 \tag{6.1}$$

We apply the condition $n = 0$ *to the subformula* $\sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2$ *which results in*

$$\forall n \,.\, n = 0 \Rightarrow \sum_{i=1}^{0} i^3 = (\sum_{i=1}^{0} i)^2 \wedge \forall n, n' \,.\, n = n' + 1 \wedge \sum_{i=1}^{n'} i^3 = (\sum_{i=1}^{n'} i)^2 \Rightarrow \sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2 \tag{6.2}$$

To that modified subformula we apply twice the definition of $\sum$ *which results in*

$$\forall n \,.\, n = 0 \Rightarrow 0 = 0 \wedge \forall n, n' \,.\, n = n' + 1 \wedge \sum_{i=1}^{n'} i^3 = (\sum_{i=1}^{n'} i)^2 \Rightarrow \sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2 \tag{6.3}$$

On the other part of the formula we apply $n = n' + 1$ *twice to the subformula* $\sum_{i=1}^{n'} i^3 = (\sum_{i=1}^{n'} i)^2$ *which reduces to*

$$\forall n \,.\, n = 0 \Rightarrow 0 = 0 \wedge \forall n, n' \,.\, n = n' + 1 \wedge \sum_{i=1}^{n'} i^3 = (\sum_{i=1}^{n'} i)^2 \Rightarrow \sum_{i=1}^{n'+1} i^3 = (\sum_{i=1}^{n'+1} i)^2 \tag{6.4}$$

Applying the definition of $\sum$ twice to that formula leaves us with

$$\forall n \,.\, n = 0 \Rightarrow 0 = 0 \wedge \forall n, n' \,.\, n = n' + 1 \wedge \sum_{i=1}^{n'} i^3 = (\sum_{i=1}^{n'} i)^2 \Rightarrow (n'+1)^3 + \sum_{i=1}^{n'} i^3 = ((n'+1) + \sum_{i=1}^{n'} i)^2 \quad (6.5)$$

By $(a+b)^2 = a^2 + 2ab + b^2$ we obtain

$$\begin{array}{l} \forall n \,.\, n = 0 \Rightarrow 0 = 0 \wedge \\ \forall n, n' \,.\, n = n' + 1 \wedge \sum_{i=1}^{n'} i^3 = (\sum_{i=1}^{n'} i)^2 \\ \quad \Rightarrow (n'+1)^3 + \sum_{i=1}^{n'} i^3 = (n'+1)^2 + 2(n'+1)(\sum_{i=1}^{n'} i) + (\sum_{i=1}^{n'} i)^2 \end{array} \quad (6.6)$$

Applying the induction hypothesis $\sum_{i=1}^{n'} i^3 = (\sum_{i=1}^{n'} i)^2$ and subsequent simplification by $(a+b=c+b) \Leftrightarrow (a=c)$ we obtain

$$\begin{array}{l} \forall n \,.\, n = 0 \Rightarrow 0 = 0 \wedge \\ \forall n, n' \,.\, n = n' + 1 \wedge \sum_{i=1}^{n'} i^3 = (\sum_{i=1}^{n'} i)^2 \\ \quad \Rightarrow (n'+1)^3 = (n'+1)^2 + 2(n'+1)(\sum_{i=1}^{n'} i) \end{array} \quad (6.7)$$

Applying $\sum_{i=1}^{n'} = \frac{n'(n'+1)}{2}$ to $\sum_{i=1}^{n'} i$ results in

$$\forall n \,.\, n = 0 \Rightarrow 0 = 0 \wedge \forall n, n' \,.\, n = n' + 1 \wedge \sum_{i=1}^{n'} i^3 = (\sum_{i=1}^{n'} i)^2 \Rightarrow (n'+1)^3 = (n'+1)^2 + 2(n'+1)(\frac{n'(n'+1)}{2}) \quad (6.8)$$

which after some further simple rearrangements results in

$$\forall n \,.\, n = 0 \Rightarrow 0 = 0 \wedge \forall n, n' \,.\, n = n' + 1 \wedge \sum_{i=1}^{n'} i^3 = (\sum_{i=1}^{n'} i)^2 \Rightarrow (n'+1)^3 = (n'+1)^3 \quad (6.9)$$

A subsequent simple simplification shows that now the proof is completed. □

The two proofs are essentially the same, except that the first has a rich proof structure, where the different cases and assumptions are explicit, while the second proof has a poor proof structure and everything is contained in a single formula. However, consider the state (6.1) in the sample proof 6.1.2: the structure of the sample proof 6.1.1 at the same stage is contained in the formula and we can obtain it by *focusing* on the respective goal formulas $\sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2$ and interpreting the logical connectives according their (uniform) type:

Example Proof 6.1.3 *The proof is by induction over n:*

Base Case $n = 0$**:** *The goal is to prove* $\sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2$.

Induction Step $n = n' + 1$**:** *The induction hypothesis is* $\sum_{i=1}^{n'} i^3 = (\sum_{i=1}^{n'} i)^2$. *The goal is to prove* $\sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2$.

Similarly, we can obtain any stage of the sample proof 6.1.1 by focusing on the respective subformulas in the corresponding step of the sample proof 6.1.2. The reason for that is that the logical connectives in the complete formula correlate with the proof structuring concepts.

The CORE proof theory supports the development of proofs in the style presented in the sample proof 6.1.2. We now introduce window structures that allow us to focus on subparts of the complete

formula. The content of the windows, i.e. the subformulas the focus of attention is on, are the visible parts of the formula to, for instance, the user. The structure of the proof is dictated by the logical connectives above the windows and the proof can be presented accordingly. Since the proof state is still a complete FVIF-tree, the formulas in the logical context of a window can still be determined by the underlying CoRe context mechanism. Furthermore, the context formulas can be applied onto the subformulas, which changes the content of a window. However, we not only allow to focus on formulas, but also to focus on any subterm below the literal level. This has already been introduced in the original window inference from [Robinson & Staples, 1993], and proved to be an adequate mechanism for theorem proving heuristics as *Rippling* [Bundy *et al*, 2003, Bundy *et al*, 1990b, Hutter, 1990] for inductive theorem proving, or its generalized variant for equational theorem proving [Hutter, 1997a, Hutter, 1997b].

6.2 Windows, Window Structures and Window Proof States

In this section we first define the notion of a *window* and then the hierarchy of windows by introducing *window structures*. Intuitively, windows show us a small but currently important part of the FVIF-tree, and technically they are pointers to subtrees and subterms of the label of a leaf-node of the FVIF-tree in a proof state. Introducing a window for a subtree allows the reasoning process to focus on that subtree hiding the surrounding parts of the whole FVIF-tree. The contextual information in the surrounding parts is still visible in the window and the rules obtained from the logical context can be used to manipulate the subtree contained in the window.

Clearly it is possible to recursively add windows to subtrees. The subtree relationship in a FVIF-tree entails a partial ordering among windows, which captures the hierarchical structure of windows. The maximal windows with respect to that induced ordering are so-called *active windows*, which are those that are actually visible to the reasoning engines and the user.

Before defining the notion of windows and window structures we define the target domain of windows in some FVIF-tree R. As motivated above, a window may denote any subtree of R as well as any proper subterm of a label of some leaf node of R. Thus, the target domain of some window for R is defined as follows:

Definition 6.2.1 (Substructures of FVIF-trees) Let R be a FVIF-tree. The *substructures* of R are all subtrees of R and all R'_π where R' is a leaf-node of R and π a non-empty valid subterm occurrence of $\mathbf{Label}(R')$. We denote that set by $\mathcal{S}(R)$. Those $S \in \mathcal{S}(R)$ that are leaf nodes annotated by some π are called *inner substructures*.

For convenience we also define $\mathcal{S}(S)$ as the set that contains all substructures of S, i.e. $\mathcal{S}(R)$ if S is a FVIF-tree R, and otherwise, if $S := R_\pi$, then $\mathcal{S}(S)$ are all $R_{\pi'}$, where π' is a valid subterm occurrence for $\mathbf{Label}(R)$ and π is a prefix of π'. ∎

In the following sections we will have to replace substructures by other substructures, which we define as follows:

Definition 6.2.2 (Replacement of Substructures) Let S, S' be substructures of some FVIF-tree, $S' \in \mathcal{S}(S)$, and S'' a substructure of another FVIF-tree. Then we denote by $(S_{|S' \leftarrow S''}, \iota)$ the replacement of S' with S'' in S together with a partial mapping $\iota : \mathcal{S}(S) \setminus \mathcal{S}(S') \hookrightarrow \mathcal{S}(S_{|S' \leftarrow S''})$ which are defined by

- If S' is a subtree and S'' is a subtree then it denotes the standard replacement of S' with S''; ι is the mapping of the substructures of S not in S' to their corresponding substructures in $S_{|S' \leftarrow S''}$.

- If $S' := R'_\pi$ and $S'' := R''_\pi$, and if the labels of R' and R'' are equal up to the subterms denoted by π, then $S_{|S' \leftarrow S''}$ denotes the replacement of R' by R''; ι is the mapping of the substructures of S that are not in S' to their corresponding substructures in $S_{|S' \leftarrow S''}$.

- Otherwise the replacement is undefined. ■

Note that in the above definition, if the whole subtree S is replaced, i.e. $S' = S$, then $dom(\iota) = \emptyset$.
Finally, windows are defined as follows:

Definition 6.2.3 (Windows) Let R be a FVIF-tree, $\mathcal{W}$ an enumerable set, and $f : \mathcal{W} \hookrightarrow \mathcal{S}(R)$ a partial function. We say that f is a *window structure for R* and each $n \in dom(f)$ is a *window* that denotes the subtree $f(n)$. The polarity, uniform type and label of n are those of $f(n)$, if $f(n)$ is a subtree. Otherwise $f(n) := R_\pi$ and n has undefined polarity ($\circ$) and uniform type ($-$), and its label is $\mathbf{Label}(R)_{|\pi}$.

We denote by (S, f) the combination of a substructure with a window structure f for S, and say that S *is annotated by* f. We say further that f is *complete* for S if, and only if, there is an $n \in dom(f)$ such that $f(n) = S$. ■

The *windows in some substructure S with respect to* f, denoted by $\mathrm{Win}(S, f)$ are all $n \in dom(f)$ such that $f(n) \in \mathcal{S}(S)$. The *restriction of a window structure f to S* is the function $f_{\downarrow S}$ defined by

$$f_{\downarrow S}(n) := \begin{cases} f(n) & \text{if } f(n) \in \mathcal{S}(S) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Note that $f_{\downarrow S}$ is always a window structure for S. The hierarchy among windows $n \in dom(f)$ is induced by the subtree and subterm relationships of the denoted substructures.

Definition 6.2.4 (Window Hierarchy and Active Windows) Let S be a substructure, f a window structure for S, and $n, n' \in dom(f)$. Then *n is smaller than n'* (we write $n \prec_f n'$ or $n' \succ_f n$) if, and only if,

- $f(n)$ is a subtree and $f(n')$ is either a proper subtree of $f(n)$ or $f(n')$ is a leaf-node R' annotated by a subterm occurrence and R' is a subtree of $f(n)$, or

- $f(n)$ is a leaf node R annotated by π and $f(n')$ is that same leaf node R annotated by π' and π is a proper prefix of π'.

The *parent window with respect to f* of some window n is that n', if it exists, such that $n' \prec_f n$ holds, and there is no n'', such that $n' \prec_f n''$ and $n'' \prec_f n$ hold. Conversely, the *child windows of n with respect to f* are all windows of which n is the parent window.

A window *n governs a substructure S* if, and only if, $S \in \mathcal{S}(f(n))$ and there is no further window n' such that $n \prec_f n'$ and for which $S \in \mathcal{S}(f(n'))$ holds.

The *active windows of f* are the windows in $dom(f)$ that are maximal with respect to $\prec_f$. ■

Intuitively the active windows denote those substructures of S that are visible to the reasoning engines and the user. As an example consider the initial FVIF-tree (cf. Figure 5.1, p. 64) for the formula

$$(\forall x_{Nat} \,.\, 0 + x = x) \wedge (\forall x_{Nat} \,.\, \forall y_{Nat} \,.\, \neg(x = 0) \Rightarrow x + y = \mathrm{s}(\mathrm{p}(x) + y))$$
$$\Rightarrow \forall p_{Nat \to \mathrm{o}} \,.\, \forall v_{Nat} \,.\, p(\mathrm{s}(\mathrm{s}(0)) + v) \Leftrightarrow P(\mathrm{s}(\mathrm{s}(v)))$$

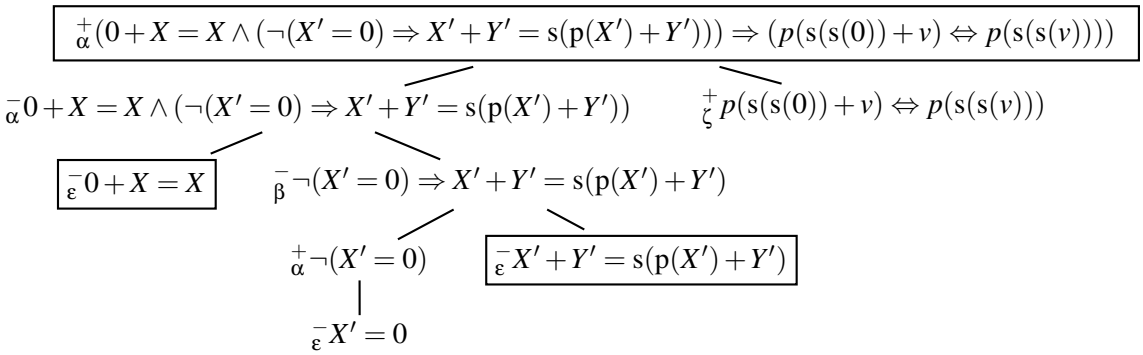


Figure 6.1: Example window structure for the initial FVIF-tree of the running example.

We introduce windows for the top-level node as well as for the subtrees $\overset{-}{\varepsilon}0+X=X$ and $\overset{-}{\varepsilon}X'+Y'=\mathrm{s}(\mathrm{p}(X')+Y')$. In Figure 6.1 (p. 100) we show the FVIF-tree together with the window structure, where we display graphically the windows on subtrees by putting a box around the root node of that subtree. In that example, the active windows are those on $\overset{-}{\varepsilon}0+X=X$ and $\overset{-}{\varepsilon}X'+Y'=\mathrm{s}(\mathrm{p}(X')+Y')$, while the root node is their parent window.

We now define the notion of a *window proof state* as an extension of a CoRe proof state by a window structure f for the FVIF-tree in the proof state.

Definition 6.2.5 (Window Proof State) Let $[Q,\sigma\rhd_{\mathcal{L}}R]$ be a CoRe proof state and f a window structure for R that is complete for R. Then $[Q,\sigma\rhd_{\mathcal{L}}(R,f)]$ is a *window proof state*. Its active windows are those of f. ■

Given a window proof state $[Q,\sigma\rhd_{\mathcal{L}}(R,f)]$, the visible goals in this proof state are the *active windows of* f. Whether those are conjunctive subgoals or alternatives depends on whether those are β-related or α-related.

For the specification of how the CoRe reasoning rules affect the window structure we define the replacement of a substructure S'' of (S,f) by (S',f'), where f is a complete window structure for S and f' is a window structure for S'.

Definition 6.2.6 (Replacement of Annotated Substructures) Let $(S,f),(S',f')$ be annotated substructures, f complete for S, and S'' a substructure of S. The *replacement* $(S,f)_{|S''\leftarrow(S',f')}$ *of* S'' *by* (S',f') *in* (S,f) is defined if, and only if, $(S_{|S''\leftarrow S'},\iota)$ is defined and $(dom(f)\setminus\mathrm{Win}(S'',f))\cap dom(f')=\emptyset$ holds. If it is defined, the replacement results in (S^*,f^*) where $S^*:=S_{|S''\leftarrow S'}$ and f^* is defined by

- If either $S''\neq S$, or $S'':=S$ and f' is complete for S', then

$$f^*(n):=\begin{cases}\iota(f(n)) & \text{if } n\in dom(f)\setminus\mathrm{Win}(S'',f)\\ f'(n) & \text{if } n\in dom(f')\\ \text{undefined} & \text{otherwise}\end{cases}$$

- Otherwise, if $S''=S$ and f' is not complete for S', then assume $n_0\notin dom(f')$ in

$$f^*(n):=\begin{cases}f'(n) & \text{if } n\in dom(f')\\ S' & \text{if } n=n_0\\ \text{undefined} & \text{otherwise}\end{cases}$$

■

The above replacement of annotated substructures is the only rule required to define the effects of CoRe reasoning rules on the window structure. A common effect pattern is also the insertion of an annotated substructure, as for example α-insertion for contraction, Leibniz' equality introduction, extensionality introduction, or β-insertion for replacement rule application. However, the insertion of annotated substructures relies on the previous replacement of annotated substructures. Nevertheless we present its formal definition, since it is widely used in the subsequent sections. In that definition we need to combine two partial functions: to this end we introduce the operator $\oplus$, which is defined on partial functions g, g' for which holds $\forall n \in dom(g) \cap dom(g') \,.\, g(n) = g'(n)$, by:

$$(g \oplus g')(n) := \begin{cases} g(n) & \text{if } n \in dom(g) \\ g'(n) & \text{if } n \in dom(g') \setminus dom(g) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Definition 6.2.7 (Insertion of Annotated Substructures) Let $(S,f),(S',f')$ be annotated substructures such that f is complete for S and $dom(f) \cap dom(f') = \emptyset$, let S'' be a substructure of S, and both S' and S'' are not inner substructures. Then we define the α-insertion (respectively β-insertion) of (S',f') on S'' in (S,f) by the replacement of S'' with the annotated α-type substructure $(\alpha(S'',S'), f_{\downarrow S''} \oplus f')$ (respectively the β-type substructure $(\beta(S'',S'), f_{\downarrow S''} \oplus f')$. We denote the resulting annotated substructures respectively by $(S,f)_{|S'' \leftarrow (\alpha(S'',S'), f_{\downarrow S''} \oplus f')}$ and $(S,f)_{|S'' \leftarrow (\beta(S'',S'), f_{\downarrow S''} \oplus f')}$ ■

6.3 CoRe Window Inference Rules

The reasoning rules for window proof states are twofold: firstly, there are rules to manipulate the window tree, i.e. to open new windows, or to close active windows. More specifically, these rules support opening of subwindows for active windows, which corresponds to the window opening rule from [Robinson & Staples, 1993]. Additionally they support opening of windows for windows that are not active, which allows us to focus on formulas in the logical context of a given window. Also we may consider alternatives to the actual goal or side-goals to an actual goal, depending on whether the new window is α- or β-related to existing active windows. Finally, active windows can be closed, which corresponds roughly to the window closing rule from [Robinson & Staples, 1993]. The first kind of rules are presented in Section 6.3.1. The second kind of rules are the window inference rules for all CoRe calculus rules, which are presented in Section 6.3.2.

6.3.1 Window Inference Rules for Window Structures

In order to prove a formula, the initial window proof state is $[Q, \sigma \rhd_{\mathcal{L}} (R, \{n \mapsto R\})]$, where $[Q, \sigma \rhd_{\mathcal{L}} R]$ is the initial proof state for the formula and n is the initial window denoting the whole subtree R. The window inference rules to manipulate the window tree are (1) opening subwindows for an active window, (2) opening further subwindows for non-active windows, and (3) closing active windows.

Definition 6.3.1 (Subwindows for Active Windows) Let $[Q, \sigma \rhd_{\mathcal{L}} (R,f)]$ be a window proof state, n an active window with respect to f, and $S_1,\ldots,S_k$ proper and independent[2] substructures of $f(n)$. The opening of subwindows for n on S is defined by

$$\frac{[Q, \sigma \rhd_{\mathcal{L}} (R,f)]}{[Q, \sigma \rhd_{\mathcal{L}} (R, \{n_{S_1} \mapsto S_1, \ldots, n_{S_k} \mapsto S_k\} \oplus f])} \text{ Subwindows for } n \text{ on } S$$

[2] I.e. for each $1 \leq i \leq n$ it holds $f(n) \notin \mathcal{S}(S_i)$ and for each $1 \leq i \neq j \leq n$ it holds $S_i \notin \mathcal{S}(S_j)$.

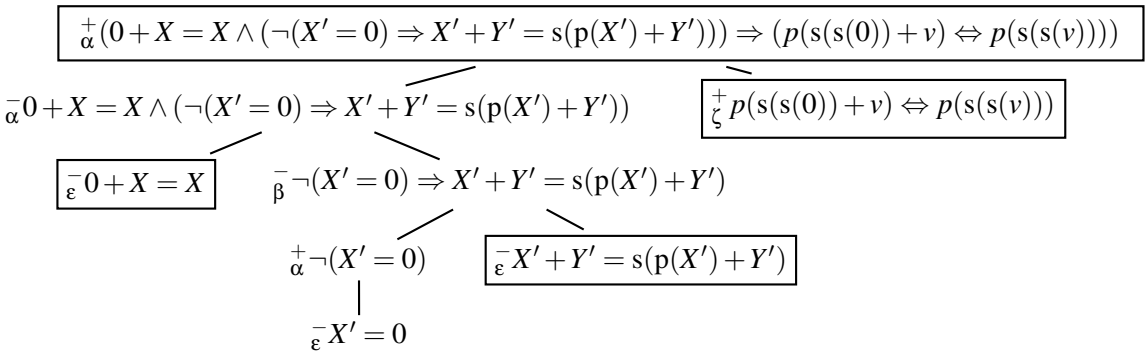


Figure 6.2: Example for opening a subwindow for a non-active window.

---

where the n_{S_i} are new with respect to $dom(f)$, and $\{n_{S_1} \mapsto S_1, \ldots, n_{S_k} \mapsto S_k\}$ denotes the partial function of domain $\{n_{S_1}, \ldots, n_{S_k}\}$ that maps each n_{S_i} to S_i. ■

While the subwindow opening rule introduces subwindows for *active* windows, it is convenient to also allow for further subwindows for non-active windows. This is supported by the subwindow addition rule.

Definition 6.3.2 (Subwindows for Non-Active Windows) Let $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ be a window proof state, n a non-active window with respect to f, S a proper substructure of $f(n)$. If $dom(f_{\downarrow S}) = \emptyset$ and n governs S, then the opening of a subwindow for n on S is defined by

$$\frac{[Q, \sigma \rhd_{\mathcal{L}} (R, f)]}{[Q, \sigma \rhd_{\mathcal{L}} (R, \{n_S \mapsto S\} \oplus f)]} \text{ Subwindow for } n \text{ on } S$$

where n_S is new with respect to $dom(f)$. ■

Consider as an example the window structure for the FVIF-tree shown in Figure 6.1 (p. 100): Opening a further subwindow for the non-active top-level window results in the window structure viewed in Figure 6.2. The active windows of that window structure are then $\overset{-}{\varepsilon}0+X=X$, $\overset{-}{\varepsilon}X'+Y'=\mathrm{s}(\mathrm{p}(X')+Y')$, and $\overset{+}{\zeta}p(\mathrm{s}(\mathrm{s}(0))+v)\Leftrightarrow p(\mathrm{s}(\mathrm{s}(v)))$.

Finally, we introduce a rule to remove active windows, which allows to un-focus.

Definition 6.3.3 (Window Closing Rule) Let $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ be a window proof state, n a window with respect to f with child windows $n_1, \ldots, n_k$. If all n_i are active windows with respect to f, then the subwindow closing rule is defined as

$$\frac{[Q, \sigma \rhd_{\mathcal{L}} (R, f)]}{[Q, \sigma \rhd_{\mathcal{L}} (R, f')]} \text{ Close Subwindows of } n$$

where $f'(n) := \begin{cases} f(n) & \text{if } n \notin \{n_1, \ldots, n_k\} \\ \text{undefined} & \text{otherwise} \end{cases}$ ■

Example 6.3.4 For example consider the window structure in Figure 6.2. Closing the window on $\overset{+}{\zeta}p(\mathrm{s}(\mathrm{s}(0))+v)\Leftrightarrow p(\mathrm{s}(\mathrm{s}(v)))$ allows us to come back to the window structure from Figure 6.1 (p. 100).

This completes the reasoning rules that affect the window structure only. In the next section we show how the standard CORE calculus rules are used as window reasoning rules and how they affect the window structure.

6.3.2 CORE Calculus Window Inference Rules

We shall now give the window versions of the actual CORE calculus rules by defining an appropriate window inference rule for each calculus rule.

6.3.2.1 Axiom

The axiom rule closes a proof when the FVIF-tree R of a proof state $[Q, \sigma \rhd_{\mathcal{L}} R]$ is proved, i.e. it is a single node for $True^+$ or $False^-$. We denote this single node tree by $Proved^p$, where p is R's polarity. For the window version of that rule we require that the window structure f of the window proof state $[Q, \sigma \rhd_{\mathcal{L}} (Proved^p, \{n \mapsto Proved^p\})]$ has a single window on $Proved^p$.

Definition 6.3.5 (Window Axiom Rule) Let $[Q, \sigma \rhd_{\mathcal{L}} (R, \{n \mapsto R\})]$ be a window proof state. If R is a FVIF-tree for $True^+$ or $False^-$, then that window proof state is proved. ∎

6.3.2.2 Contraction

The contraction rule is applied to some window proof state $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ and replaces a subtree R' of R by some new subtree that contains R' and its copy R''. From the copying we obtain an isomorphic mapping $\iota : \mathcal{S}(R') \to \mathcal{S}(R'')$ and together with $f_{\downarrow R'}$ we define new window structure $f_{R''}$ for R'' such that there is a mapping $\iota' : dom(f_{\downarrow R'}) \to dom(f_{R''})$ and $\iota \circ f_{\downarrow R'} = f_{R''} \circ \iota'$ holds, i.e. the following diagram must commute:

$$\begin{array}{ccc} \mathcal{S}(R') & \xrightarrow{\quad\iota\quad} & \mathcal{S}(R'') \\ {\scriptstyle f_{\downarrow R'}}\uparrow & /\!/ & \uparrow{\scriptstyle f_{R''}} \\ dom(f_{\downarrow R'}) & \xrightarrow[\quad\iota'\quad]{} & dom(f_{R''}) \end{array}$$

Then the window contraction rule consists of the α-insertion of $(R'', f_{R''})$ on R' in (R, f).

Definition 6.3.6 (Window Contraction Rule) Let $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ be a window proof state, R' a subtree of R, R'' a copy of R' with mapping ι, a window structure $f_{R''}$ obtained from $f_{\downarrow R'}$ and ι, and a mapping $\iota' : \mathcal{S}(R') \to \mathcal{S}(R'')$. Then the *window contraction rule* is defined by

$$\frac{[Q, \sigma \rhd_{\mathcal{L}} (R, f)]}{[Q, \sigma \rhd_{\mathcal{L}} (R, f)_{|R' \leftarrow (\alpha(R', R''), f_{\downarrow R'} \oplus f_{R''})}]} \text{ Window contraction of } R'$$

if, and only if, $\iota \circ f_{\downarrow R'} = f_{R''} \circ \iota'$ holds. ∎

Example 6.3.7 Consider as an example the window structure of Figure 6.2 (p. 102): The application of the window contraction rule on $\bar{}_{\varepsilon} 0 + X = X$ duplicates that subtree and inserts a new subwindow for the copied subtree since there was a window on the original subtree. The resulting window structure is shown in Figure 6.3 (p. 104).

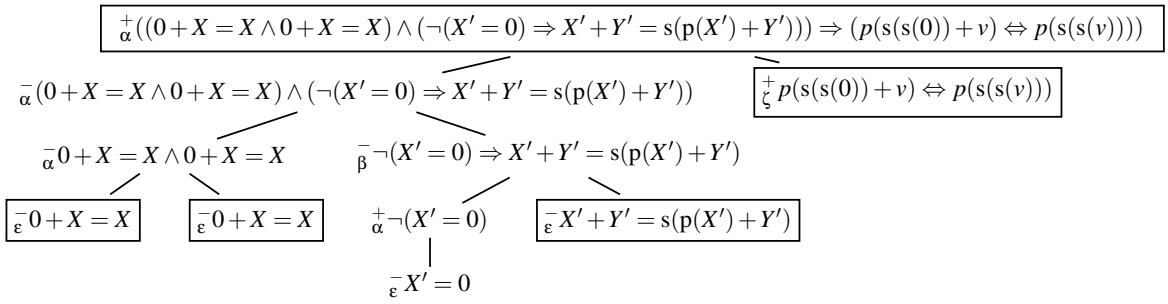


Figure 6.3: Window structure resulting from contracting $\overset{-}{\varepsilon}0+X=X$.

6.3.2.3 Weakening

The weakening rule replaces a subtree R' of R by a subtree $R'' \in \mathit{Weakened}(R')$. Some of the subtrees of R' are no longer present in the weakened subtree and we need to construct a window structure for R'' from R'. To this end we extend the definition of weakened subtrees to annotated subtrees that return a weakened subtree together with an adequate window structure. In that definition we need the converse operation $\ominus$ to $\oplus$, which is defined on arbitrary partial functions g, g' by

$$(g \ominus g')(n) := \begin{cases} g(n) & \text{if } n \in \mathit{dom}(g) \setminus \mathit{dom}(g') \\ \text{undefined} & \text{otherwise} \end{cases}$$

Definition 6.3.8 (Weakening of Annotated FVIF-trees) Let (R, f) be an annotated FVIF-tree. The set $\mathit{Weakened}(R, f)$ of weakened annotated FVIF-trees for (R, f) is defined recursively over the structure of R:

- If R is a leaf node, then
$$\mathit{Weakened}(R, f) := \{(R, f_{\downarrow R})\}$$
- If $R := \alpha^p(R_1, R_2)$, then
  - if $\exists n \in \mathit{dom}(f)\,.\, f(n) = \alpha^p(R_1, R_2)$, then
$$\begin{aligned} \mathit{Weakened}(\alpha^p(R_1,R_2), f) := \quad & \{(\alpha^p(R_1^w, R_2^w), f_1 \oplus f_2 \oplus \{n \mapsto \alpha^p(R_1^w, R_2^w)\}) \\ & \qquad \mid (R_i^w, f_i) \in \mathit{Weakened}(R_i, f), i = 1,2\} \\ & \cup \mathit{Weakened}(R_1, (f \ominus \{n' \mapsto R_1 \mid \forall n'\}) \oplus \{n \mapsto R_1\}) \\ & \cup \mathit{Weakened}(R_2, (f \ominus \{n' \mapsto R_2 \mid \forall n'\}) \oplus \{n \mapsto R_2\}) \end{aligned}$$
  - otherwise
$$\begin{aligned} \mathit{Weakened}(\alpha^p(R_1,R_2), f) := \quad & \{(\alpha^p(R_1^w, R_2^w), f_1 \oplus f_2) \\ & \qquad \mid (R_i^w, f_i) \in \mathit{Weakened}(R_i, f), i = 1,2\} \\ & \cup \mathit{Weakened}(R_1, f) \cup \mathit{Weakened}(R_2, f) \end{aligned}$$
- If $R := \beta^p(R_1, R_2)$, then

– If $\exists n \in dom(f)\,.\,f(n) = \beta^p(R_1,R_2)$ then

$$Weakened(\beta^p(R_1,R_2),f) := \{(\beta^p(R_1^w,R_2^w), f_1 \oplus f_2 \oplus \{n \mapsto \beta^p(R_1^w,R_2^w)) \mid (R_i^w,f_i) \in Weakened(R_i,f), i=1,2\}$$

– otherwise

$$Weakened(\beta^p(R_1,R_2),f) := \{(\beta^p(R_1^w,R_2^w), f_1 \oplus f_2) \mid (R_i^w,f_i) \in Weakened(R_i,f), i=1,2\}$$

– If $R := \nu^p(R')$, then

– If $\exists n \in dom(f)\,.\,f(n) = \nu^p(R')$ then

$$Weakened(\nu^p(R'),f) := \{(\nu^p(R'^w), f' \oplus \{n \mapsto \nu^p(R'^w)\}) \mid (R'^w,f') \in Weakened(R',f)\}$$

– Otherwise

$$Weakened(\nu^p(R'),f) := \{(\nu^p(R'^w), f') \mid (R'^w,f') \in Weakened(R',f)\}$$

– If $R := \pi^p(R')$, then

– If $\exists n \in dom(f)\,.\,f(n) = \pi^p(R')$, then

$$Weakened(\pi^p(R'),f) := \{(\pi^p(R'^w), f' \oplus \{n \mapsto \pi^p(R'^w)\}) \mid (R'^w,f') \in Weakened(R')\}$$

– Otherwise

$$Weakened(\pi^p(R'),f) := \{(\pi^p(R'^w), f') \mid (R'^w,f') \in Weakened(R')\}$$

■

Using the weakening of annotated FVIF-trees the window weakening rule is defined over the replacement of annotated substructures as follows:

Definition 6.3.9 (Window Weakening Rule) Let $[Q, \sigma \rhd_{\mathcal{L}} (R,f)]$ be a window proof state, R'' a subtree of R. Then the *window weakening rule is defined by*

$$\frac{[Q, \sigma \rhd_{\mathcal{L}} (R,f)]}{[Q, \sigma \rhd_{\mathcal{L}} (R,f)_{|R'' \leftarrow (R',f')}]}\ \text{Window weakening of } R''$$

where $(R',f') \in Weakened(R'', f_{\downarrow R''})$. ■

Example 6.3.10 Consider as an example the window structure of Figure 6.3 (p. 104): Weakening the $\bar{}_{\varepsilon} 0 + X = X$ introduced by contraction removes that subtree and its window and we obtain again the window structure from Figure 6.2 (p. 102).

6.3.2.4 Structural Modal Permutations

The structural modal rule replaces for example subtrees of R that have the form $\nu_Q(\alpha(R_1,R_2))$ by $\alpha(\nu_Q(R_1),\nu_Q(R_2))$. The window version of that rule replaces the substructure $\nu_Q(\alpha(R_1,R_2))$ in (R,f) with the annotated substructure $(\alpha(\nu_Q(R_1),\nu_Q(R_2)),f')$ for some f'. For the specification of f' we proceed as follows: any window structure inside the R_i is preserved. If there is a window on $\nu_Q(\alpha(R_1,R_2))$ or $\alpha(R_1,R_2)$, then those are set to $\alpha(\nu_Q(R_1),\nu_Q(R_2))$, possibly by merging them.

Definition 6.3.11 (Window Structural Modal Permutation Rule) Let $[Q,\sigma \rhd_{\mathcal{L}} (R,f)]$ be a window proof state, R' a subtree of R on which the CORE structural modal permutation rule is applicable. The *window structural modal permutation rule* is then defined as follows:

1. If $R' = \nu^p(\alpha^p(R''^{-p}))$, then

$$\frac{[Q,\sigma \rhd_{\mathcal{L}} (R,f)]}{[Q,\sigma \rhd_{\mathcal{L}} (R,f)_{|\nu^p(\alpha^p(R''^{-p}))\leftarrow(\alpha^p(\nu^{-p}(R''^{-p})),f')}]}\ \text{Unary window modal permutation on } R'$$

   where f' is defined by

   - if there is an n such that $f(n) = \nu^p(\alpha^p(R''^{-p}))$ and an n' with $f(n') = \alpha^p(R''^{-p})$, then $f' := f_{\downarrow R''} \oplus \{n \mapsto \alpha^p(\nu^{-p}(R''^{-p}))\}$.
   - if there is an n' with $f(n') = \alpha^p(R''^{-p})$, but no n with $f(n) = \nu^p(\alpha^p(R''^{-p}))$, then $f' := f_{\downarrow R''} \oplus \{n' \mapsto \alpha^p(\nu^{-p}(R''^{-p}))\}$.
   - otherwise $f' := f_{\downarrow R''}$.

2. If $R' = \nu^p(\alpha^p(R_1^{p_1},R_2^{p_2}))$, then

$$\frac{[Q,\sigma \rhd_{\mathcal{L}} (R,f)]}{[Q,\sigma \rhd_{\mathcal{L}} (R,f)_{|\nu^p(\alpha^p(R_1^{p_1},R_2^{p_2}))\leftarrow(\alpha^p(\nu^{p_1}(R1^{p_1}),\nu^{p_2}(R_2^{p_2})),f')}]}\ \text{Binary window modal permutation on } R'$$

   where f' is defined by

   - if there is an n such that $f(n) = \nu^p(\alpha^p(R_1^{p_1},R_2^{p_2}))$ and an n' with $f(n') = \alpha^p(R_1^{p_1},R_2^{p_2})$, then $f' := f_{\downarrow R_1} \oplus f_{\downarrow R_2} \oplus \{n \mapsto \alpha^p(\nu^{p_1}(R1^{p_1}),\nu^{p_2}(R_2^{p_2}))\}$.
   - if there is an n' with $f(n') = \alpha^p(R_1^{p_1},R_2^{p_2})$, but no n with $f(n) = \nu^p(\alpha^p(R_1^{p_1},R_2^{p_2}))$, then $f' := f_{\downarrow R_1} \oplus f_{\downarrow R_2} \oplus \{n' \mapsto \alpha^p(\nu^{p_1}(R_1^{p_1}),\nu^{p_2}(R_2^{p_2}))\}$.
   - otherwise $f' := f_{\downarrow R_1} \oplus f_{\downarrow R_2}$.

   The other cases of the rule for the different forms of R' are analogously. ∎

Example 6.3.12 Consider as an example the formula $\Box(A \wedge B) \Rightarrow \Box A$ and its window structure on the left-hand side of Figure 6.4, where the subscripts Q and Q' are the references to the variable nodes in the corresponding indexed formula tree. The application of the window structural modal permutation to the subtree $\bar{\nu}\Box_Q(A \wedge B)$ moves the modal quantifier inwards and inherits the window on that node to the new node $\bar{\alpha}(\Box_Q(A) \wedge \Box_Q(B))$.

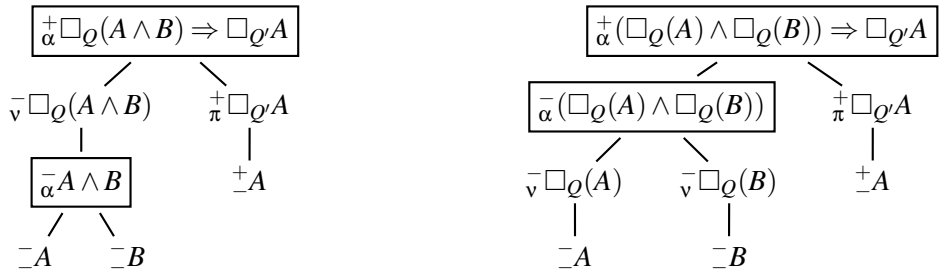


Figure 6.4: Window structures before and after structural modal permutation on $\overset{-}{\alpha}A \wedge B$.

6.3.2.5 Resolution Style Replacement Rule Application

The resolution style replacement rule application of $u \rightarrow \langle v'_1, \ldots, v'_m \rangle$ on some subtree a replaces the subtree a by a subtree for *Proved* and β-inserts the (weakened) subtrees v'_i. However, for the window version of that rule we must accommodate a possible window structure that is inside a and inherit it in an adequate manner. This is for instance the case if we have a resolution replacement rule that stems from a rewrite replacement rule. If there is a window structure, say f_a, inside a, we can easily construct a window structure f_u for u since a and u have the same label and thus u and a have isomorphic substructures. Assume now that the label of u can be expressed by $\mathbf{Label}(u) := ((\lambda x_1, \ldots, x_n \,.\, s(x_1, \ldots, x_n))u_1 \ldots u_n)$, where the u_i correspond to the active windows in u and the x_i denote the positions of these windows and hence occur exactly once in $s(x_1, \ldots, x_n)$. In order to adequately inherit that structure during rule application, we need to find a v'_i which label is of the form $((\lambda x_1, \ldots, x_n \,.\, s(x_1, \ldots, x_n))v_1 \ldots v_n)$. If so, we can inherit the window structure to that v_i. Note that this requirements also ensure that all non-active windows in u can be uniquely assigned to v_i as the context of the $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ are equal. In order to formalise that requirement we introduce the notion of isomorphic substructures *up to some substructures and a substitution*. We use that notion afterwards to strengthen the application condition of replacement rules for the window version of the resolution replacement rule application rule.

Definition 6.3.13 (Isomorphic Substructures up to some Substructures and Substitution) Let S, S' be two substructures, σ an $\mathcal{L}$-substitution, and $S_1, \ldots, S_n \in \mathcal{S}(S)$ and $S'_1, \ldots, S'_n \in \mathcal{S}(S')$. We say that *$S$ and S' are isomorphic up to $S_1, \ldots, S_n$ and $S'_1, \ldots, S'_n$ and* σ if, and only if, $\sigma(\mathbf{Label}(S)) = \mathbf{Label}(S'_{|S'_i \leftarrow S_i, i=1\ldots n})$. If so, then there exists an injective morphism $\iota : \mathcal{S}(S) \rightarrow \mathcal{S}(S'_{|S'_i \leftarrow S_i, i=1\ldots n})$. That function is the identity function on the S_i.

Furthermore, we denote by $\iota_{S,S'}$ the mapping $\{S_1 \mapsto S'_1, \ldots, S_1 \mapsto S'_1\} \circ \iota$, which is an injective morphism

$$\iota_{S,S'} : (\mathcal{S}(S) \setminus \bigcup_{j=1}^{n} \mathcal{S}(S_j)) \cup \{S_1, \ldots S_n\} \rightarrow (\mathcal{S}(S') \setminus \bigcup_{j=1}^{n} \mathcal{S}(S'_j)) \cup \{S'_1, \ldots S'_n\}$$

■

The restriction of the window application of a resolution replacement rule $u \rightarrow \langle v'_1, \ldots, v'_m \rangle$ on some annotated FVIF-tree (a, f) is then that we must find a v'_i that has the same modal prefix than a and is isomorphic to a up to some $v_1, \ldots v_n \in \mathcal{S}(v'_i)$ and the substructures $a_1, \ldots, a_n$ of a which are denoted

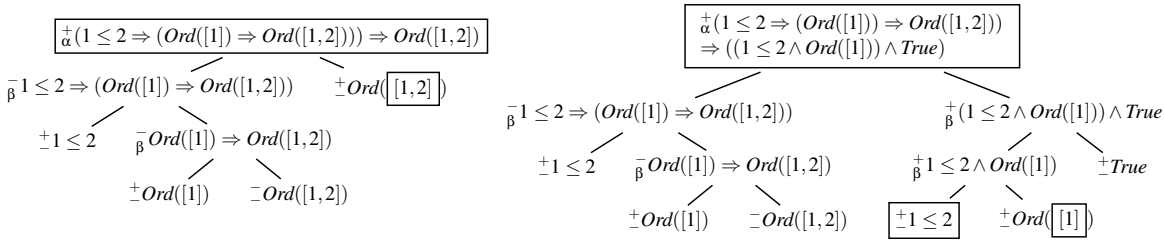


Figure 6.5: Window structures before and after application of the replacement rule.

---

by the active windows of f. From the respective mapping ι_{a,v'_i} we can construct a window structure f' for v'_i and an isomorphism $\iota' : dom(f) \to dom(f')$ such that the following diagram commutes:

$$\begin{array}{ccc} (\mathcal{S}(a) \setminus \bigcup_{j=1}^{n} \mathcal{S}(a_j)) \cup \{a_1, \ldots a_n\} & \xrightarrow{\quad \iota \quad} & (\mathcal{S}(v'_i) \setminus \bigcup_{j=1}^{n} \mathcal{S}(v'_{ij})) \cup \{v'_{i1}, \ldots v'_{in}\} \\ f \uparrow & // & \uparrow f' \\ dom(f) & \xrightarrow[\quad \iota' \quad]{} & dom(f') \end{array}$$

For the definition of the window resolution replacement rule application note that we also introduce new windows that denote the additional β-inserted subtrees $v'_j, j = 1 \ldots n, j \neq i$ that are inserted by the resolution replacement rule on subtrees R^* for which $f_{\downarrow R^*}$ is non-empty.

Definition 6.3.14 (Window Resolution Replacement Rule Application) Let $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ be a window proof state, a a subtree of R, and $u \to \langle v'_1, \ldots, v'_m \rangle$ an admissible resolution replacement rule for a. The rule is window applicable on a in (R, f) if, and only if, there is a v'_i that has the same modal prefix than a, and has substructures $v_i^{'1}, \ldots, v_i^{'n}$, such that v'_i is isomorphic to a with respect to $v_i^{'1}, \ldots, v_i^{'n}$ and the active windows in a with respect to $f_{\downarrow a}$ and results in the mapping ι_{a,v'_i}. Then the window resolution replacement rule application is defined by

$$\frac{[Q, \sigma \rhd_{\mathcal{L}} (R, f)]}{[Q, \sigma \rhd_{\mathcal{L}} (R^*, f^*)]} \text{ Apply } u \to \langle v'_1, \ldots, v'_m \rangle \text{ on } a$$

where (R^*, f^*) results from

- the replacement of a with $(Proved^p, \emptyset)$,
- the β-insertion of (v'_i, f') on some adequate subtree, and where f' is such that there is an isomorphism ι' and it holds $\iota \circ f = f' \circ \iota'$.
- the β-insertion of (v'_j, f_j) on some adequate subtree R_j for $j = 1 \ldots n, j \neq i$, where $f_j := \{n_j \mapsto v'_j\}$, n_j is new, if $f_{\downarrow R_j} \neq \{\}$; otherwise $f_j := \{\}$. ∎

Example 6.3.15 Consider as an example the window structure view on the left-hand side of Figure 6.5, where there is an window on the inner substructure $[1,2]$ of the subtree $\overline{}^{+}Ord([1,2])$. The

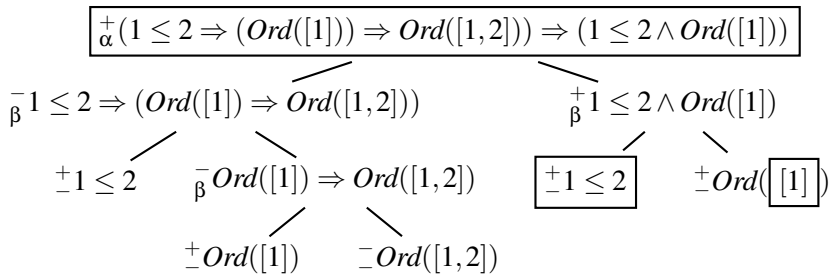


Figure 6.6: Window structure after simplification of the window structure from Figure 6.5 (p. 108).

---

application of the resolution replacement rule $\overset{-}{-}Ord([1,2]) \rightarrow \left\langle \overset{+}{-}1 \leq 2, \overset{+}{-}Ord([1]) \right\rangle$ to that subtree inherits that window to the inner substructure $[1]$ of the inserted subgoal $\overset{+}{-}Ord([1])$ and adds a new window for the further subgoal $\overset{+}{-}1 \leq 2$. The resulting window structure is shown on the right-hand side of Figure 6.5.

6.3.2.6 Simplification

The simplification rule replaces a subtree R' of R by (1) either a subtree R'' for either $True^+$ or $False^-$ if R' is proved, or by a subtree R'' for $True^-$ or $False^+$ if it is disproved. Or (2) if R' is of the form $\beta(R_1,R_2)$ where exactly one of the R_i is proved, by $R_j, j \neq i$, if R_j and R' have the same polarity, and otherwise by $\alpha(R_j)$. Or, finally, (3) if R' is of the form $\alpha(R_1,R_2)$ where exactly one of the R_i is disproved, by $R_j, j \neq i$, if R_j and R' have the same polarity, and otherwise by $\alpha(R_j)$.

In the first case, the window structure for R' is removed and if there was a window on R', then it shall denote R'' afterwards. In the second case any window structure for R_i is deleted, while the window structure for R_j, $j \neq i$, is preserved. If there are windows n and n' that respectively denote $\beta(R_1,R_2)$ and R_j and the replacing subtree is $\alpha(R_j)$, then n' denotes $\alpha(R_j)$ afterwards. Otherwise, if the replacing subtree is R_j, then n is removed. The third case finally is analogously to the last case.

Definition 6.3.16 (Window Simplification Rule) Let $[Q,\sigma \rhd_{\mathcal{L}} (R,f)]$ be a window proof state, R' a subtree of R.

1. If R' of polarity p is proved (respectively disproved), then let R'' be a FVIF-tree for $True^+$ (respectively $False^+$), if $p = +$, and for $False^-$ (respectively $True^-$), if $p = -$. Then the simplification rule is

$$\frac{[Q,\sigma \rhd_{\mathcal{L}} (R,f)]}{[Q,\sigma \rhd_{\mathcal{L}} (R,f)_{|R' \leftarrow (R'',f'')}]} \text{ Window simplify } R'$$

   where $f''(n) := \begin{cases} R'' & \text{if f(n) = R',} \\ \text{undefined} & \text{otherwise.} \end{cases}$

2. If $R' := \beta(R_1,R_2)$ and not proved (respectively $\alpha(R_1,R_2)$ and not disproved) and R_i is proved (respectively disproved), then:

   - If R' and R_j, $j \neq i$, have the same polarity, then the simplification rule is:

$$\frac{[Q,\sigma \rhd_{\mathcal{L}} (R,f)]}{[Q,\sigma \rhd_{\mathcal{L}} (R,f)_{|R' \leftarrow (R_j,f_j)}]} \text{ Window simplify } R'$$

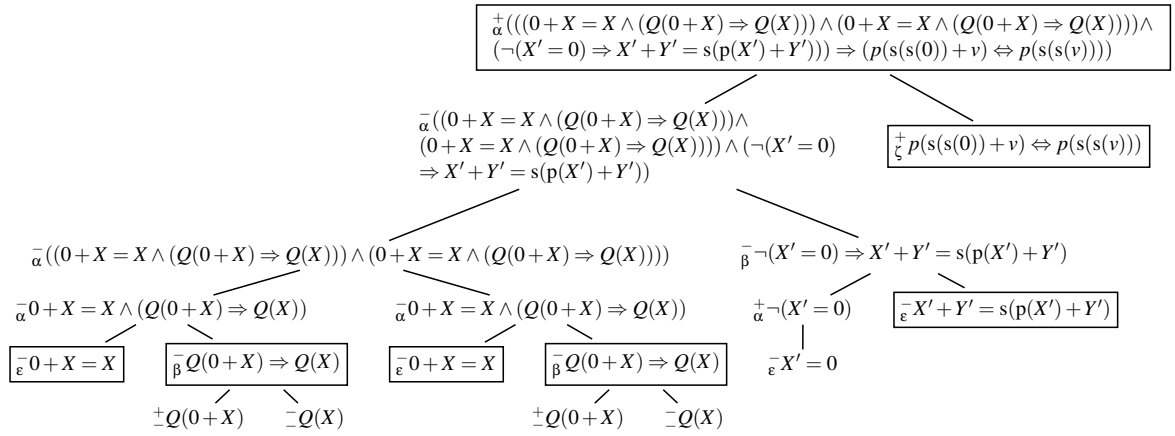


Figure 6.7: Window structure resulting from introducing Leibniz' equality for the ε-type formula $\overset{-}{\varepsilon}0+X=X$.

---

where $f_j := f_{\downarrow R_j}$.

– If R' and R_j, $j \neq i$, have opposite polarities, then the simplification rule is:

$$\frac{[Q, \sigma \rhd_{\mathcal{L}} (R, f)]}{[Q, \sigma \rhd_{\mathcal{L}} (R, f)_{|R' \leftarrow (\alpha(R_j), f_j)}]}\ \text{Window simplify } R'$$

$$\text{where } f_j(n) := \begin{cases} f_{\downarrow R_j}(n) & \text{if } n \in dom(f_{\downarrow R_j}) \\ \alpha(R_j) & \text{if } f(n) := R' \\ \text{undefined} & \text{otherwise.} \end{cases}$$

■

Example 6.3.17 Consider as an example the window structure on the left-hand side of Figure 6.5 (p. 108). The window simplification of it results in the window structure shown in Figure 6.6 (p. 109).

6.3.2.7 Leibniz' Equality

Given a window proof state $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$, let $Q' := \overset{\xi}{\zeta}(s,t)^p$ be an ε- or ζ-type leaf node in Q of polarity p, and $R_1, \ldots, R_n$ the leaf nodes in R that belong to Q'. Then the Leibniz' equality introduction rule for Q' α-inserts on each R_i a FVIF-tree R_i' for $(P(s) \Rightarrow P(t))^p$. For the window version of that rule the window structure for the R_i remain in place and new windows are added for each R_i'.

Definition 6.3.18 (Window Leibniz' Equality Introduction Rule) Let $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ be a window proof state, $Q' := \overset{\xi}{\zeta}(s,t)$ a leaf node of polarity p in Q, and $R_1, \ldots, R_n$ the leaf nodes in R that belong to Q'. Let further Q'' be an indexed formula tree for $(\forall P \,.\, P(s) \Rightarrow P(t))^p$ and $R_1', \ldots, R_n'$ FVIF-trees for Q''. Then the *window Leibniz' equality introduction rule* is

$$\frac{[Q, \sigma \rhd_{\mathcal{L}} R]}{[Q_{|Q' \leftarrow \alpha(Q', Q'')}, \sigma \rhd_{\mathcal{L}} (R, f)_{|R_i \leftarrow (\alpha(R_i, R_i'), f_i), i=1 \ldots n}]}\ \text{Window Leibniz' equality introduction on } Q'$$

where for all $1 \leq i \leq n$, $f_i := f_{\downarrow R_i} \oplus f_i'$, where $f_i' := \{\}$ if $f_{\downarrow R_i} = \{\}$, and otherwise $f_i' := \{n_i \mapsto R_i'\}$ and n_i is new. ■

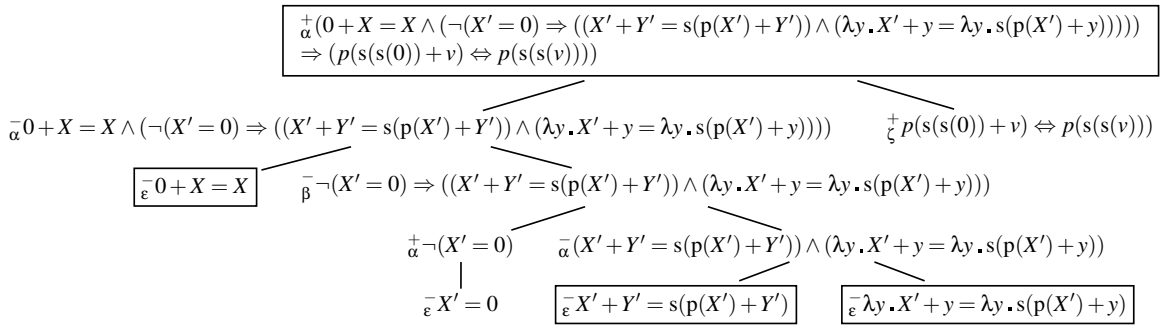


Figure 6.8: Window structure after extensionality introduction for ε-type formula $\overset{-}{\varepsilon}X'+Y'=\mathrm{s}(\mathrm{p}(X')+Y')$ with respect to γ-local variable Y'.

Example 6.3.19 Consider as an example the window structure obtained by window contraction shown in Figure 6.3 (p. 104). The application of the window Leibniz' equality introduction rule for $\overset{-}{\varepsilon}0+X=X$ inserts for both occurrences the respective Leibniz' equality subtrees and adds subwindows for the introduced subtrees, since there were subwindows on the occurrences of $\overset{-}{\varepsilon}0+X=X$. The resulting window structure is shown in Figure 6.7.

6.3.2.8 Extensionality

Given a window proof state $[Q,\sigma\rhd_{\mathcal{L}}(R,f)]$, let $Q':=\overset{\xi}{\zeta}(s,t)^p$ be an ε- or ζ-type leaf node in Q of polarity p, x a variable that is local for Q' and $R_1,\ldots,R_n$ the leaf nodes in R that belong to Q'. Then the extensionality introduction rule for Q' with x α-inserts on each R_i a FVIF-tree R_i' for $(\lambda x.s=\lambda x.t)^p$. For the window version of that rule the window structure for the R_i remain in place and new windows are added for each R_i'.

Definition 6.3.20 (Window Extensionality Introduction Rule) Let $[Q,\sigma\rhd_{\mathcal{L}}(R,f)]$ be a window proof state, $Q':=\overset{\xi}{\zeta}(s,t)$ a leaf node of polarity p in Q, x local for Q', and $R_1,\ldots,R_n$ the leaf nodes in R that belong to Q'. Let further Q'' be an indexed formula tree for $(\lambda x.s=\lambda x.t)^p$ and $R_1',\ldots,R_n'$ FVIF-trees for Q''. Then the *window extensionality introduction rule* is

$$\frac{[Q,\sigma\rhd_{\mathcal{L}}R]}{[Q_{|Q'\leftarrow\alpha(Q',Q'')},\sigma\rhd_{\mathcal{L}}(R,f)_{|R_i\leftarrow(\alpha(R_i,R_i'),f_i),i=1\ldots n}]}\ \text{Window extensionality introduction for } Q' \text{ with } x$$

where for all $1\leq i\leq n$, $f_i:=f_{\downarrow R_i}\oplus f_i'$, where $f_i':=\{\}$ if $f_{\downarrow R_i}=\{\}$, and otherwise $f_i':=\{n_i\mapsto R_i'\}$ and n_i is new. ■

Example 6.3.21 Consider as an example the window structure from Figure 6.1 (p. 100). The window extensionality introduction on $\overset{-}{\varepsilon}X'+Y'=\mathrm{s}(\mathrm{p}(X')+Y')$ with respect to the γ-local variable Y' α-inserts the subtree $\overset{-}{\varepsilon}\lambda y.X'+y=\lambda y.\mathrm{s}(\mathrm{p}(X')+y)$ and adds a window for that subtree since there was a window on $\overset{-}{\varepsilon}X'+Y'=\mathrm{s}(\mathrm{p}(X')+Y')$. The resulting window structure is shown in Figure 6.8.

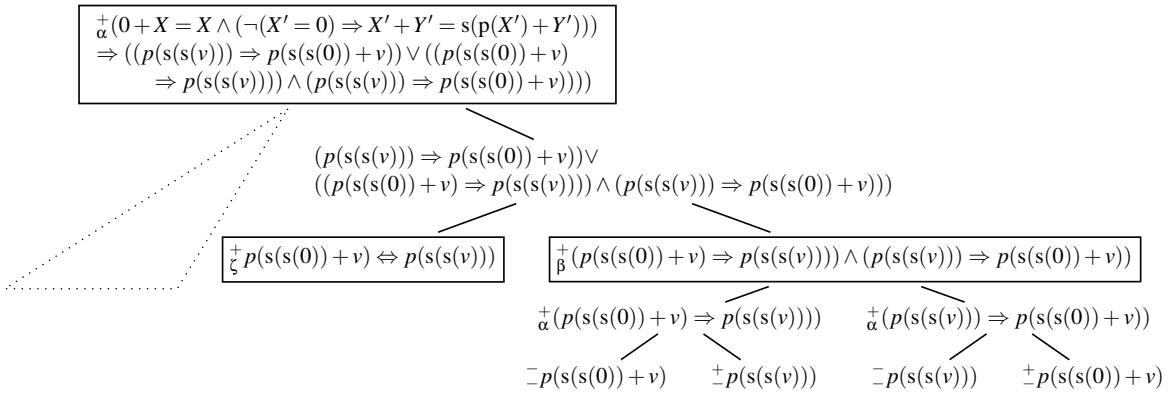


Figure 6.9: Window structure after boolean ζ-expansion on $\overset{+}{\zeta}p(\mathrm{s}(\mathrm{s}(0))+v)\Leftrightarrow p(\mathrm{s}(\mathrm{s}(v)))$.

6.3.2.9 Boolean ζ-Expansion

Given a window proof state $[Q,\sigma\rhd_{\mathcal{L}}(R,f)]$, let $Q':=\zeta(A_{\mathrm{o}},B_{\mathrm{o}})^{+}$ be a ζ-type leaf node in Q of positive polarity, and $R_1,\ldots,R_n$ the leaf nodes in R that belong to Q'. Then the boolean ζ-expansion rule for Q' α-inserts on each R_i a FVIF-tree R_i' for $((A\Rightarrow B)\wedge(B\Rightarrow A))^{+}$. For the window version of that rule the window structure for the R_i remain in place and new windows are added for each R_i'.

Definition 6.3.22 (Window Boolean ζ-Expansion Rule) Let $[Q,\sigma\rhd_{\mathcal{L}}(R,f)]$ be a window proof state, $Q':=\zeta(A_{\mathrm{o}},B_{\mathrm{o}})$ a positive leaf node in Q, and $R_1,\ldots,R_n$ the leaf nodes in R that belong to Q'. Let further Q'' be an indexed formula tree for $((A\Rightarrow B)\wedge(B\Rightarrow A))^{+}$ and $R_1',\ldots,R_n'$ FVIF-trees for Q''. Then the *window boolean ζ-expansion rule* is

$$\frac{[Q,\sigma\rhd_{\mathcal{L}}R]}{[Q_{|Q'\leftarrow\alpha(Q',Q'')},\sigma\rhd_{\mathcal{L}}(R,f)_{|R_i\leftarrow(\alpha(R_i,R_i'),f_i),i=1\ldots n}]}\ \text{Window boolean }\zeta\text{-expansion of }Q'$$

where for all $1\leq i\leq n$, $f_i:=f_{\downarrow R_i}\oplus f_i'$, where $f_i':=\{\}$ if $f_{\downarrow R_i}=\{\}$, and otherwise $f_i':=\{n_i\mapsto R_i'\}$ and n_i is new. ■

Example 6.3.23 Consider as an example the window structure after opening an additional subwindow for the non-active top-level window from Figure 6.2 (p. 102). The application of the window boolean ζ-expansion for $\overset{+}{\zeta}p(\mathrm{s}(\mathrm{s}(0))+v)\Leftrightarrow p(\mathrm{s}(\mathrm{s}(v)))$ α-inserts the new subtree for $\overset{+}{\beta}(p(\mathrm{s}(\mathrm{s}(0))+v)\Rightarrow p(\mathrm{s}(\mathrm{s}(v))))\wedge(p(\mathrm{s}(\mathrm{s}(v)))\Rightarrow p(\mathrm{s}(\mathrm{s}(0))+v))$ and adds a window for it since there was a window on $\overset{+}{\zeta}p(\mathrm{s}(\mathrm{s}(0))+v)\Leftrightarrow p(\mathrm{s}(\mathrm{s}(v)))$. The resulting window structure is shown in Figure 6.9.

6.3.2.10 Instantiation

The instantiation rule replaces each literal in which occurs an instantiated variable by a new FVIF-tree for the instantiated label. If a higher-order variable is instantiated and somewhere there is a window structure inside the substructure of an occurrence of that variable, then that window structure is affected by that instantiation. In these cases we simply remove the window structure that is inside such a substructure. Formally, we define the window version of the instantiation rule by:

Definition 6.3.24 (Window Instantiation) Let $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ be a window proof state, and σ' an $\mathcal{L}$-substitution, such that $\sigma' \circ \sigma$ is $\mathcal{L}$-admissible. Let further be $Q_1, \ldots, Q_n$ the leaf nodes in Q that are affected by σ', and for each Q_i let $R_i^1, \ldots, R_i^{n_i}$ be the leaf nodes in R that belong to Q_i. Finally, for each Q_i let Q_i' be the indexed formula tree for $\sigma'(\mathbf{Label}(Q_i))$ that replaces Q_i and $R_i'^k$ be the FVIF-tree for Q_i' that replaces R_i^k. Then the *window instantiation rule* is

$$\frac{[Q, \sigma \rhd_{\mathcal{L}} R]}{[Q_{|Q_i \leftarrow Q_i'}, \sigma' \circ \sigma \rhd_{\mathcal{L}} (R, f)_{|R_i^k \leftarrow (R_i'^k, f_i^k), 1 \leq i \leq n, 1 \leq k \leq n_i}]} \text{ Window instantiation } \sigma'$$

where the f_i^k are defined by: let $S_1, \ldots, S_j$ be the maximal substructures in R_i^k that contain a higher-order variable from $dom(\sigma')$. They correspond to substructures $S_1', \ldots, S_j'$ of $R_i'^k$ such that R_i^k and $R_i'^k$ are isomorphic up to $S_1, \ldots, S_j$ and $S_1', \ldots, S_j'$ and σ'. From there we obtain the injective morphism $\iota_{R_i^k, R_i'^k}$ and together with the restriction of f to the domain of $\iota_{R_i^k, R_i'^k}$ can obtain f_i^k and ι_i^k such that $\iota_{R_i^k, R_i'^k} \circ f_{\downarrow dom(\iota_{R_i^k, R_i'^k})} = f_i^k \circ \iota_i^k$ holds. ∎

6.3.2.11 Increase of Multiplicities

The increase of multiplicities increases the multiplicities of specific subtrees in Q from which we obtain a variable renaming ρ. Furthermore, the rule α-inserts new subtrees $R_1'', \ldots, R_n''$ respectively on specific subtrees $R_1', \ldots R_n'$ of R (cf. Definition 5.3.31), and it holds $\rho(\mathbf{Label}(R_i')) = \mathbf{Label}(R_i'')$, for all $1 \leq i \leq n$.

For the window version of that rule we copy any window structure inside the R_i' to the respective R_i''. To this end we use the property that R_i' and R_i'' are isomorphic up to the substitution ρ. From there we can obtain an adequate f_i' for R_i'' from $f_{\downarrow R_i'}$.

Definition 6.3.25 (Window Increase of Multiplicities) Let $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ be a window proof state, Q' the indexed formula tree that results from the increase of multiplicities in Q, ρ the respective renaming, ι the respective automorphism on subnodes of Q', and σ' the new overall $\mathcal{L}$-substitution. Furthermore, let $R_1', \ldots R_n'$ be the subtrees of R that are copied modulo ρ and ι to obtain $R_1'', \ldots, R_n''$. Then the *window rule to increase multiplicities* is

$$\frac{[Q, \sigma \rhd_{\mathcal{L}} (R, f)]}{[Q', \sigma' \rhd_{\mathcal{L}} (R, f)_{|R_i' \leftarrow (\alpha(R_i', R_i''), f_{\downarrow R_i'} \oplus f_i'), i=1 \ldots n}]} \text{ Window increase multiplicities}$$

where f_i' is obtained from the isomorphic relationship up to ρ of R_i' and R_i'' which entails the morphisms $\iota_{R_i', R_i''}$ and ι_i such that $\iota_{R_i', R_i''} \circ f_{\downarrow R_i'} = f_i' \circ \iota_i$ holds. ∎

6.3.2.12 Rewriting Style Replacement Rule Application

This rule is a specific combination of Leibniz' equality introduction, instantiation, and resolution style replacement rule application. During the definition of the window resolution style replacement rule application we have already taken care that the rewriting replacement rule application is handled in the appropriate manner.

6.3.2.13 Cut

The Cut over some formula φ on some subtree R' of R and of defined polarity p is achieved by

1. α-inserting an initial indexed formula tree Q' for the closed quantified cut formula $(\exists\vec{x}.(\varphi \Rightarrow \varphi))^+$ (respectively $(\Diamond\exists\vec{x}.(\varphi \Rightarrow \varphi))^+$ for modal logics) together with an $\mathcal{L}$-substitution σ' and FVIF-trees R^-_φ and R^+_φ respectively for the negative and positive subtrees of label φ in Q',
2. copies R' to R'' and replaces R' in R with $\beta(\alpha(R^-_\varphi, R'), \alpha(R^+_\varphi, R''))$.

For the window version of that rule we inherit any window structure inside R' to R'' and add windows denoting R^-_φ and R^+_φ only if there was a window on or inside R' before rule application, i.e. if $f_{\downarrow R'} \neq \{\}$ where f is the window structure before rule application.

Definition 6.3.26 (Window Cut Rule) Let $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ be a window proof state, R' a subtree of R with defined polarity, Q' the initial indexed formula tree for $(\exists\vec{x}.(\varphi \Rightarrow \varphi))^+$[3], σ' the substitution to adequately integrate Q', R^-_φ and R^+_φ FVIF-trees respectively for the negative and positive subtrees of label φ in Q', and R'' a copy of R'. Then the *window cut rule* is

$$\frac{[Q, \sigma \rhd_{\mathcal{L}} (R, f)]}{[\alpha(Q, Q'), \sigma' \circ \sigma \rhd_{\mathcal{L}} (R, f)_{|R' \leftarrow (\beta(\alpha(R^-_\varphi, R'), \alpha(R^+_\varphi, R'')), f')}]} \text{ Window cut over } \varphi$$

where f' is defined by: let f'' be the window structure for R'' obtained from $f_{\downarrow R'}$ and its isomorphic relationship to R'. Furthermore, let $f_\varphi := \{\}$ if $f_{\downarrow R'} = \{\}$ and otherwise $f_\varphi := \{n^- \mapsto R^-_\varphi, n^+ \mapsto R^+_\varphi\}$, n^-, n^+ new. Then $f' := f_{\downarrow R'} \oplus f'' \oplus f_\varphi$. ∎

6.4 Summary

In this chapter we added further intuitive reasoning capabilities onto the CoRE rules to support the focusing and manipulation of arbitrary subparts of the FVIF-tree. It turned out that window inferencing is only a technical add-on to support a hierarchical reasoning style which does not provide additional contextual reasoning capabilities, since the underlying framework already supports all necessary contextual reasoning capabilites.

Focusing on subparts of the FVIF-tree supports the user and the reasoning engines to arbitrarily choose a list of open goals, consider alternatives to chosen subgoals, and come back on those decisions, without having to backtrack.

Finally, the different possibilities to adapt the window tree structure during the application of the CoRE cut rule allows to represent various reasoning methods like case analysis, proof by contradiction and speculative proof steps. This is the basis to support both classical tactical or fully automatic proof search procedures and proof planning procedures within the same framework.

[3] respectively $(\Diamond\exists\vec{x}.(\varphi \Rightarrow \varphi))^+$ for modal logics.

Chapter 7

Change of Representation

Representational change is an important feature not only in mathematical problem solving. It is often used in order to simplify a given problem posed in some representation language by translating it into a more adequate representation of the problem. The translation can either by an adequate reformulation of the problem and hence a proof obtained for the problem in the new representation proves the original problem. Or the translation is a strict abstraction of the source representation in which case there is a priori no formal relationship between a proof with respect to the new representation and a possible proof with respect to the original representation. However, the proof with respect to the new representation can for instance be used as a plan to guide the search for a proof with respect to the original representation.

In Section 7.1 we present some changes of representation known from the literature that have been successfully used in theorem proving, especially proof planning. Based on the requirement specification we present in Section 7.2 the concepts that underly the infrastructure for representational change implemented in CoRe.

7.1 Examples for Representational Changes

Labelled Fragments in Inductive Theorem Proving. The major task in inductive theorem proving is to apply the induction hypothesis to the inductive conclusion, i.e. to change the representation such that this application becomes possible. The key observation to support a goal-directed guidance of the manipulation of the induction conclusion in order to eventually make the induction hypothesis applicable is that the latter is contained in the induction conclusion, i.e. it is a *skeleton*. The differences between the induction hypothesis and the induction conclusion are then occurrences of function symbols around the parts that belong to the skeleton. Take as an example the following formula over x as induction conclusion $\varphi(x+\mathrm{s}(y))$ and let the induction hypothesis be $\varphi(x)$. The differences between these formulas are the occurrences of $+$, y, and s. In order to apply $\varphi(x)$ the induction conclusion must be transformed into a formula of the form $\Psi(\varphi(x))$. The differences are made explicit by annotating the function symbols by colours, e.g. *white* if they belong to the skeleton and *gray* if they belong to the differences: $\varphi($ x $+\mathrm{s}(y)$ $)$.

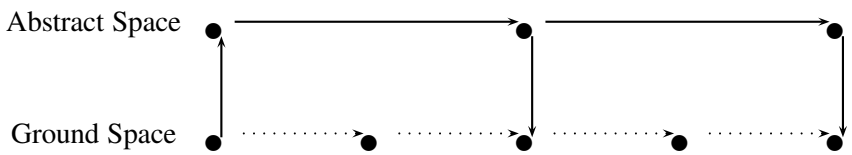
The guidance information that enables the application of the induction hypothesis consists in applying rules that – from an operational point of view – move the differences towards the top-level of the formula, while preserving the skeleton parts. An example for this kind of rule is

$$X + (Y + Z) = (X + Y) + Z \tag{7.1}$$

The *labelled fragments* representation of [Hutter, 1994] is an abstraction from the concrete function symbols contained in the differences. Thereby all occurrences of function symbols that form the differences are abstracted to a new uninterpreted function symbol $\bullet$. In the above example we obtain: $\varphi(\bullet(x))$. Similarly, the axioms are abstracted with respect to the skeleton/context annotations, which results for example in

$$X + \bullet(Y) = \bullet(X + Y) \tag{7.2}$$

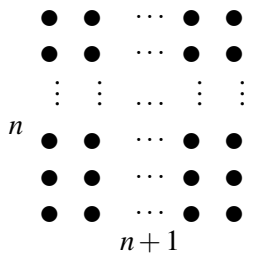
This way the abstract space is constructed from the ground space. A proof in the abstract space does not entail that there is a proof in the ground space. However, it can serve as a proof plan for the proof in the ground space, and each intermediate step in the abstract space corresponds to some of the intermediate formulas in the ground space proof. That is, we have



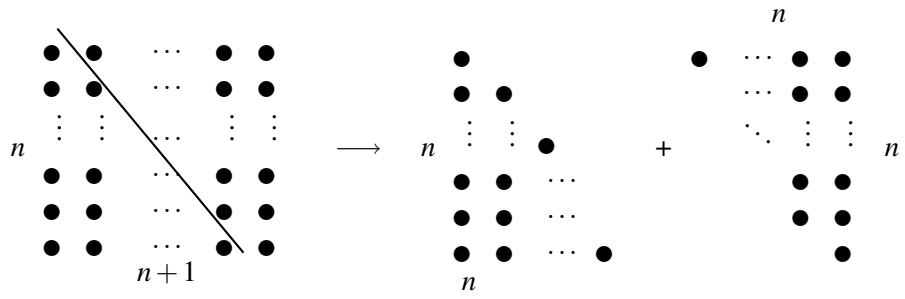
Although the existence of a proof in the abstract space does not imply that there is a proof in the ground space, the converse holds: if there is no proof in the abstract space, then there ist no proof in the ground space.

From a logical point of view the abstract space consists, like the ground space, of a set of axioms and an abstracted conjecture. In order to support the reuse of generic reasoning procedures, the abstract space should be represented in the same formalism than the ground space.

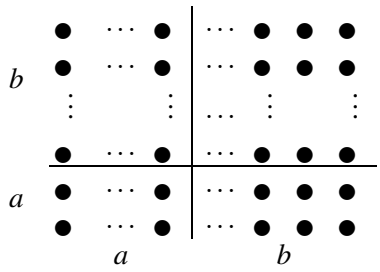
Diagrammatic Reasoning. Diagrams and geometric operations over diagrams are a formalism widely used in mathematical problem solving. [Jamnik *et al*, 1997, Jamnik *et al*, 1999] investigate how to support this style of reasoning, especially for reasoning about sums and products over natural numbers. For instance the product of n and $n+1$ can be represented by the following diagram:



Geometric operations over these diagrams correspond to specific mathematical operations over natural numbers. For example half of the above product diagram is obtained by virtually drawing a diagonal line in the square that results in two isosceles triangles of height and width n:



Similarly, the product can be decomposed into sub-diagrams which corresponds to the decomposition of the natural number represented by the product into a^2, b^2, ab, and ba:



In order to support this style of reasoning within a theorem prover a language describing the geometric objects is defined: square(a,b) for squares of height a and width b, and triangle(a,b) for triangles of height a and width b. The geometric operations are described by further functions like half(square$(n,n+1)$) that divides a square into two triangles, "flip" to transform an $n \times m$-square (resp. triangle) into an $m \times n$-square (resp. triangle), or "+ "for the horizontal and vertical composition of squares. The semantics of these operations is then defined by the following axioms:

$$\text{half}(\text{square}(n,n+1)) = \text{triangle}(n,n) \tag{7.3}$$

$$\text{flip}(\text{square}(n,m)) = \text{square}(m,n) \tag{7.4}$$

$$\text{square}(a,b)+\text{square}(c,b) = \text{square}(a+c,b) \tag{7.5}$$

$$\text{square}(a,b)+\text{square}(a,c) = \text{square}(a,b+c) \tag{7.6}$$

$$\begin{aligned}\text{square}(a+b,a+b) &= (\text{square}(a,a)+\text{square}(a,b)) \\ &\quad +(\text{square}(b,a)+\text{square}(b,b))\end{aligned} \tag{7.7}$$

To use that representation for an actual problem solving task, we first have to determine whether it is a problem in the domain of natural numbers. If so, the abstract space is constructed from the ground space by mapping the axioms and conjectures into the diagram representation and adding the above axioms describing the geometric operations. The problem on the abstract space is then again composed of a set of axioms and a conjecture. If the mapping is adequate then a proof with respect to the abstract space entails that there is a proof in the ground space. If the mapping is not adequate, for instance if the conjecture in the ground space contained properties that could not be represented by diagrams, then the abstract proof can still be used as a proof plan to guide (parts of) the proof in the ground space.

Abstracting to Simpler Representations. More classical abstractions than the above are mappings of problems given with respect to some logic $\mathcal{L}$ to a problem with respect to a weaker logic $\mathcal{L}'$. Examples are:

- The mapping of higher-order logic problems to first-order logic problems: either a higher-order logic problem is reformulated as a first-order logic problem as done in [Kerber, 1992], or the higher-order logic problem is essentially a first-order order problem, and thus can be reformulated as such in a canonical way. The advantage of these reformulations is that the proof search procedures for first-order logic are more efficient respectively more well developed than those for higher-order logic. In both cases the first-order proof entails the existence of a higher-order logic proof, if the initial transformation was adequate. Otherwise, it can serve as a proof plan for those parts of the initial higher-order problem that have been adequately transformed.
- The mapping of first-order logic problems to propositional logic problems. This is essentially similar to the previous case. Note that in this case the transformation allows to move from a semi-decidable domain to a decidable domain.
- The mapping of (parts of) a problem into a representation suitable to use a decision procedure. Take as an example a decision procedure for linear arithmetic: if the original problem contains subproblems that are essentially linear arithmetic problems, then an explicit transformation into the representation for pure linear arithmetic problems is required in order to enable the application of the decision procedure. Again, the proof with respect to the abstract domain implies the existence of a proof in the original domain only if the mapping is adequate. Otherwise, the abstract proof can be used as a proof plan to guide (parts of) the proof with respect to the original domain.

These examples demonstrate the following informal requirement specification for an infrastructure that efficiently supports the use of abstractions during proof search:

- Proof search with respect to different representations should be supported in parallel. Although both proof states should be separated from each other for soundness reasons, the relations between the proofs with respect to different levels of abstractions should be explicit in order to communicate this information to all partners involved in the problem solving process.
- Abstractions should be treated as first-class citizens like other (pure) calculus rules. They should be provided as such to the different partners analogously to the calculus rules and contextual information. This requires a mechanism to define the application domains of representational abstractions in order to allow to check their applicability in some proof state.

7.2 Concepts and Rules for Representational Change

In this section we introduce the basic concepts that underlie the definition and use of change of representations, i.e., abstractions, in CoRe. Regarding the informal requirement specification in the previous section, CoRe supports multiple simultaneous proof states. Establishing the connection between proofs that belong to different proof states is a matter of proof representation and it will be addressed in Chapter 8. It remains to provide the concepts of abstractions, the explicit representation of the applicability of abstractions, and the treatment of abstractions and refinements as first-class citizens during proof search.

In order to describe the application domain of an abstraction we introduce the notion of a *reasoning domain* in Section 7.2.1. Intuitively a reasoning domain is a signature containing the types and constants that are required by an abstraction function. Based on this we define the concept of a *representational abstraction* in Section 7.2.2.

7.2.1 Reasoning Domains

For the description of representational abstractions, we do not impose a declarative description of the representational abstractions but rather allow for any kind of description of those transformations, explicitly including any programming language. In order to support an explicit representation of the application domain for specific representational abstractions, we introduce the notion of *reasoning domains*. The intuition that underlies their definition is that a representational abstraction usually has built-in knowledge about specific types and constants and exploits that built-in knowledge in order to compute a new representation. Take as an example the representational abstraction to diagram representations: the abstraction exploits the knowledge about functions over the natural numbers. Since its implementation needs to recognise that type and the respective functions, it must rely on the syntax, i.e. a base type *Nat* and function symbols $+, \times, -, \Sigma$ of types $Nat \times Nat \rightarrow Nat$ as well as the logical connectives $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$. Furthermore, the application domain of a representational abstraction may be restricted to a specific logic, like the change from a higher-order logic representation to some first-order domain. Reasoning domains are a mean to make that syntactical and logical requirements explicit.

Definition 7.2.1 (Reasoning Domains) Let $\mathcal{L}$ be a logic and Σ a valid $\mathcal{L}$-signature. Then $\mathbb{D} := (\mathcal{L}, \Sigma)$ is a *reasoning domain*. A reasoning domain $\mathbb{D} := (\mathcal{L}, \Sigma)$ is *more specific* than a reasoning domain $\mathbb{D}' := (\mathcal{L}', \Sigma')$ if, and only if, $\mathcal{L}'$ is a sub-logic of $\mathcal{L}$ and Σ' is contained in Σ. ■

A representational abstraction is then applicable to some proof state if its reasoning domain is less specific than the reasoning domain of the proof state. Thereby reasoning domain of a (window) proof state consists of the logic of that proof state together with all types and constants that occur in that proof state, which is defined as follows:

Definition 7.2.2 (Reasoning Domain of Window Proof States) Let $\mathcal{L}$ be logic, *WPS* a window proof state with respect to $\mathcal{L}$, and Σ the $\mathcal{L}$-signature that consists of all types and constants in *WPS*. Then the *reasoning domain of WPS* is $(\mathcal{L}, \Sigma)$. ■

Given a reasoning domain $\mathbb{D}$ we denote by $\mathcal{WPS}_{\mathbb{D}}$ the set of window proof states that have a reasoning domain that is equal to or more specific than $\mathbb{D}$.

7.2.2 Representational Abstractions

A representational abstraction is a mapping $\aleph$ that maps window proof states to window proof states. In order to describe the type of the mapping, the representational abstraction consists of a reasoning domain describing its application domain and a target reasoning domain. Then the mapping $\aleph$ is a function from the set of window proof states of the source reasoning domain into the set of window proof states for the target reasoning domain. The source reasoning domain is thereby a description that typically only *approximates* the actual domain of the mapping $\aleph$. Thus, $\aleph$ is typically only a *partial* function. In order to represent the relations between the ground and abstract proof states, the representational abstraction function must provide the information which active windows of the

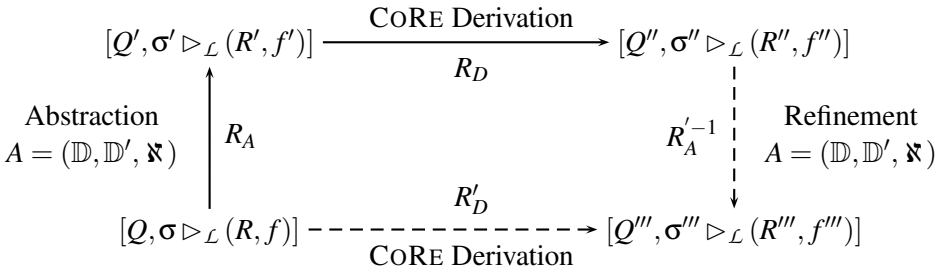


Figure 7.1: Abstraction & refinement.

---

ground proof state correspond to which active windows of the abstract proof state. This is used to establish the corresponding proof links in the proof representation in Chapter 8. Thus, the representational abstraction function $\aleph$ must provide both the new proof state and the relationship between old and new active windows in the respective window trees. Note that this active window relationship is indeed a relation and not a mapping, since an active window with respect to the ground proof state may correspond to more than one active window in the abstract proof state.

Definition 7.2.3 (Representational Abstractions) A *representational abstraction* is a 3-tuple $A = (\mathbb{D}, \mathbb{D}', \aleph)$ where $\mathbb{D}, \mathbb{D}'$ are reasoning domains and $\aleph$ a partial mapping from $\mathcal{WPS}_{\mathbb{D}}$ into $\mathcal{WPS}_{\mathbb{D}'} \times \mathcal{R}$, where $\mathcal{R}$ is the set of binary relations between window structures, such that:

- If $\aleph([Q, \sigma \rhd_{\mathcal{L}} (R, f)]) = ([Q', \sigma' \rhd_{\mathcal{L}} (R', f')], \mathcal{R})$ then for all (n, n') in $\mathcal{R}$, n is an active window of f with respect to R and n' is an active window of f' with respect to R'.

We say that $\mathbb{D}$ is the *source* reasoning domain of A and $\mathbb{D}'$ is the *target* reasoning domain of A. ■

The representational abstractions can be used at any stage of the proof process to change the representation. The relation $\mathcal{R}$ indicates which active window of the ground proof state is abstracted to which active windows in the abstract proof state. The actual application of the representational abstraction consists of first checking the applicability of the abstraction by comparing the source reasoning domain of the abstraction with the reasoning domain of the actual proof state $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$. Secondly, the abstraction function is applied on the proof state which yields the new proof state $[Q', \sigma' \rhd_{\mathcal{L}} (R', f')]$. Thus, instead of having a single proof state at the time, the use of representational abstractions requires to support the representation and management of multiple proof states.

7.2.3 Representational Refinements

In this section we define the converse of the representational abstraction in order to support the refinement of an abstract proof state to the original proof state. The situation is as follows: we started with a proof state $WPS = [Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ that has been abstracted to the proof state $WPS' = [Q', \sigma' \rhd_{\mathcal{L}} (R', f')]$. From that proof state we obtained the proof state $WPS'' = [Q'', \sigma'' \rhd_{\mathcal{L}} (R'', f'')]$ by applying CORE reasoning rules. We now have to map back this new proof state to obtain a successor proof state for *WPS*.

We have sketched the situation in Figure 7.1: in the figure, the solid lines describe the actual situation and the dashed lines describe the additional steps that need to be introduced in order to refine the abstract proof. Assume R_A is the relationship between active windows of *WPS* and *WPS'*, and R_D is a relationship between active windows in *WPS'* and the derived abstract proof state *WPS''*. For

the moment we assume such a relationship R_D exists and we define it in Chapter 8. The refinement problem consists in determining a proof state $WPS''' = [Q''', \sigma''' \rhd_{\mathcal{L}} (R''', f''')]$, such that:

1. $\aleph(WPS''') = (WPS'', R'_A)$, i.e. WPS'' is the proof state obtained by representational abstraction from WPS''' and R'_A is the relationship between active windows from W''' to W'',
2. there is a CoRe derivation from WPS to WPS''' and R'_D is the relationship between active windows in WPS and WPS''',
3. and it holds $R_D \circ R_A = R'_A \circ R'_D$, where $R \circ R' := \{(w, w'') \mid \exists w' \,.\, (w, w') \in R \text{ and } (w', w'') \in R'\}$.

Note that in the figure the representational refinement relation from WPS'' to WPS''' is annotated with the active window relationship $R_A^{'-1}$ that is the inverse of the relationship obtained by abstraction; i.e. $R_A^{'-1} := \{(w', w) \mid (w, w') \in R'_A\}$.

Definition 7.2.4 (Representational Refinement) Let $A = (\mathbb{D}, \mathbb{D}', \aleph)$ be a representational abstraction, WPS, WPS', WPS'', and WPS''' be window proof states, such that

1. $\aleph(WPS) = (WPS', R_A)$, $\aleph(WPS''') = (WPS'', R'_A)$
2. $WPS' \longmapsto^*_W WPS''$ and $WPS \longmapsto^*_W WPS'''$ are valid CoRe proofs with respective active window relationships R_D and R'_D.

Then WPS''' is a representational refinement of WPS'' by A if, and only if, it holds $R_D \circ R_A = R'_A \circ R'_D$. ■

In the definition of representational refinement we explicitly refrained to impose a constructive way to determine the refining proof state WPS'''. In general there are different ways to find that proof state: for example if the abstract representation contains enough details in order to compute the proof state WPS''' from WPS'', then the derivation of WPS to WPS''' can be performed by, for instance, using the cut rule. Or the abstract derivation from WPS' to WPS'' can be used to guide the derivation of WPS''' from WPS and subsequently checking whether $\aleph(WPS''') = (WPS'', R'_A)$ holds.

7.3 Summary

Reasoning domains have been defined as a declarative approximation for the application domain of the actual representational abstractions. This information is used in order to check the applicability of representational abstractions and provide the user and the reasoning engines with the applicable abstractions, similar to any other contextual information about possible continuations of the proof. Furthermore, we defined the effect of using a representational abstraction at any stage of the proof search and the relations between active windows of the original proof state and the abstract proof state. Finally we defined the invariants for the refinement of (parts of) an abstract proof.

Chapter 8

Hierarchical Proof Datastructure

The framework defined in this thesis aims at a communication infrastructure which mediates between the user and the reasoning engines. In the design of the framework we distinguished two major parts: first, the status of the proof as well as the possible next steps must be presented in an intelligible manner to both the user and the reasoning engines. To this end we introduced in Chapter 6 the notion of a window proof state that is based on the CoRe calculus for intuitive contextual reasoning. The second major part of the framework is concerned with information about the history of a proof as a complementary information to the status of a proof. We also envisioned a proof representation with different types of proofs, namely proofs based on calculus rules, proof plans, or, more generally, any kind of proof step annotated with a natural language description.

This chapter is concerned with the definition of such a proof representation for CoRe. In Section 8.1 we motivate the datastructure used for proofs. In the first part of that section we sketch the proof representation required to adequately encode the CoRe window inference rules. The analysis of these rules leads to the informal definition of the methodological *roles* of subgoals introduced by the window inference rules. Finally, we motivate the hierarchical structure of proofs that allows to abbreviate portions of proofs, to expand speculative proof steps, or to use representational abstractions and refinements. The formal definition of the hierarchical proof datastructure is then presented in Section 8.2 and we show how the informal categories of *intra-level* and *inter-level* proof construction steps from Section 1.1.2 are naturally represented in the hierarchical proof datastructure.

8.1 Motivation of the Hierarchical Proof Datastructure

The goal is to provide a proof representation that deals with all the aspects of proof development considered so far in this thesis. Ideally, the proof representation should on the one hand adequately represent the proof history and efficiently support proof continuations and on the other hand be in a format that is easy to proof check. However, because of the complexity of the CoRe window calculus the latter is not possible, without encoding the complexity in the proof checking rules. The causes for the complexity are the global effects of some reasoning rules, as for instance replacement rule applications or the increase of multiplicities, which hampers simple proof checking of CoRe window proofs. Therefore, we concentrate on the first aspects, i.e. to develop a proof representation that adequately represent the proof history and efficiently supports proof continuations and leave the proof checking aspects for future work.

8.1.1 CoRe Window Inference Rules

The open goals of a window proof state $WPS = [Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ are the active windows with respect to f. Therefore to design the proof datastructure we define a proof node for each active window. Note that the actual window proof state *WPS* is shared among these proof nodes for the active windows. In order to capture the sharing of the window proof state among the proof nodes, we add the window proof state *WPS* to each proof node. Thus, a proof node consists of a window proof state *WPS* and one of its active windows $w \in dom(f)$. For notational convenience we also add labels to proof node. Thus a (window) proof node is denoted by

Label	WPState		Window	Justification/Abstraction/Refinement
L_0	WPS	$\vdash$	w	

where L_0 is its label, *WPS* its window proof state with window structure f and $w \in dom(f)$ an active window from *WPS*.

We now consider the CoRe window inference rules. From an operational point of view they either replace active windows by new windows, close active windows, or add further active subwindows to non-active windows. We consider the rules in more detail in order to motivate the effect of a window rule application on the proof nodes.

Active Subwindow Opening Rule. The rule that opens subwindows $w_1, \ldots, w_n$ for an active window w relates w's proof node to the proof nodes for the new windows.

Label	WPState		Window	Justification/Abstraction/Refinement
L_0	WPS	$\vdash$	w	*Subwindow-Open* : $L_1, \ldots, L_n$
L_1	WPS_1	$\vdash$	w_1	
$\vdots$	$\vdots$	$\vdots$		
L_n	WPS_1	$\vdash$	w_n	

We say that w's proof node *is justified by the proof nodes for the w_i's via the subwindow opening rule* and add that justification to w's proof node. Note that the new windows belong to a new window proof state WPS_1. We say that a node is *open* if, and only if, the node has no justification. The invariant for the proof representation is that the open nodes all have the same window proof state.

Non-Active Subwindow Opening Rule. The rule that opens a subwindow w' for a *non-active window* w is used to focus on subtrees that are surrounding some given window. The rule adds a new active subwindow for this subtree to w's list of subwindows. Since w is a non-active window, there is a proof node for w with justification *Subwindow-Open*$(L_1, \ldots, L_n)$ in the proof representation. Adding a further subwindow transforms the proof to

Label	WPState		Window	Justification/Abstraction/Refinement
L_0	WPS	$\vdash$	w	*Subwindow-Open* : $L_1, \ldots, L_n, L_{n+1}$
L_1	WPS_1	$\vdash$	w_1	
$\vdots$	$\vdots$	$\vdots$		
L_n	WPS_1	$\vdash$	w_n	
L_{n+1}	WPS_2	$\vdash$	w'	

The window proof state by the time the additional subwindow is introduced differs from the window proof states introduced for the other subwindows w_i since the window structures differ. In order to ensure the invariant of the proof representation, we must adapt the window proof states WPS_1 for the open goals to WPS_2. To this end we introduce a justification *Adapt-Window-Proof-State* for each open node and obtain

Label	WPState		Window	Justification/Abstraction/Refinement
L_0	WPS	$\vdash$	w	*Subwindow-Open* : $L_1, \ldots, L_n, L_{n+1}$
L_1	WPS_1	$\vdash$	w_1	*Adapt-Window-Proof-State* : L'_1
L'_1	WPS_2	$\vdash$	w_1	
$\vdots$	$\vdots$	$\vdots$		
L_n	WPS_1	$\vdash$	w_n	*Adapt-Window-Proof-State* : L'_n
L'_n	WPS_2	$\vdash$	w_n	
L_{n+1}	WPS_2	$\vdash$	w'	

Subwindow Closing Rule. The final window reasoning rule that only affects the window structure is the subwindow closing rule. It closes all active subwindows of some window w and the parent window gets active again. It introduces a new window proof node for the parent window and justifies all window proof nodes of its subwindows by the justification *Window-Close*.

Label	WPState		Window	Justification/Abstraction/Refinement
L_0	WPS_0	$\vdash$	w	*Subwindow-Open* : $L_1, \ldots, L_n, L_{n+1}$
L_1	WPS_1	$\vdash$	w_1	*Adapt-Window-Proof-State* : L'_1
L'_1	WPS_2	$\vdash$	w_1	*Window-Close* : L_{n+2}
$\vdots$	$\vdots$	$\vdots$		
L_n	WPS_1	$\vdash$	w_n	*Adapt-Window-Proof-State* : L'_n
L'_n	WPS_2	$\vdash$	w_n	*Window-Close* : L_{n+2}
L_{n+1}	WPS_2	$\vdash$	w'	*Window-Close* : L_{n+2}
L_{n+2}	WPS_3	$\vdash$	w	

Window Axiom Rule. The next CORE window inference rule is the *Axiom* rule. It justifies a window proof node, whose window proof state is proved and the window is the top-level window. Thus, it transforms the proof

Label	WPState		Window	Justification/Abstraction/Refinement
$\vdots$	$\vdots$	$\vdots$		
L	$[Q, \sigma \rhd_{\mathcal{L}} (\mathit{Proved}, \{w \mapsto \mathit{Proved}\})]$	$\vdash$	w	

where $[Q, \sigma \rhd_{\mathcal{L}} (\mathit{Proved}, \{w \mapsto \mathit{Proved}\})]$ is a proved window proof state and its window structure is the single window w into

Label	WPState		Window	Justification/Abstraction/Refinement
$\vdots$	$\vdots$	$\vdots$		
L	$[Q, \sigma \rhd_{\mathcal{L}} (\mathit{Proved}, \{w \mapsto \mathit{Proved}\})]$	$\vdash$	w	*Axiom*

Window Contraction and Multiplicity-Increase. Both the window contraction rule and the window multiplicity-increase rule α-insert copies R'' of some subtrees R' and, if there is a window structure inside R', creates an isomorphic window structure for R''. Otherwise, the domain of the window structure is not affected. We present the change of the proof representation for the contraction rule. The change of the proof representation for the multiplicity increasing rule is analogous.

If the window structure in R' is empty, i.e. $f_{\downarrow R'} = \emptyset$, we can assume that w is the active window such that $f(w)$ contains the copied subtree as a proper substructure. Then the window contraction proof step is represented by

Label	WPState		Window	Justification/Abstraction/Refinement
⋮	⋮	⋮		
L	WPS	$\vdash$	w	$Contraction(R') : L'$
L'	WPS'	$\vdash$	w	
⋮	⋮	⋮		

Otherwise, if there was a window structure f' in R' then we obtain f'' as window structure for R'' which is isomorphic to f'. Then for each active window $w_1, \ldots, w_n$ of f' there is a one-to-one correspondence to the active windows $w'_1, \ldots, w'_n$ of f''. Thus, each proof node for w_i is justified by the window contraction rule to a new proof node for w_i and a proof node for w'_i, both with respect to the new window proof state. Thus, the resulting proof representation is

Label	WPState		Window	Justification/Abstraction/Refinement
⋮	⋮	⋮		
L_1	WPS	$\vdash$	w_1	$Contraction(R') : L'_1, L''_1$
⋮	⋮	⋮		
L_n	WPS	$\vdash$	w_n	$Contraction(R') : L'_n, L''_n$
L'_1	WPS'	$\vdash$	w_1	
⋮	⋮	⋮		
L'_n	WPS'	$\vdash$	w_n	
L''_1	WPS'	$\vdash$	w'_1	
⋮	⋮	⋮		
L''_n	WPS'	$\vdash$	w'_n	
⋮	⋮	⋮		

In either case in order to ensure the invariant of the proof representation we must adapt the window proof state of the remaining open goals by using the justification *Adapt-Window-Proof-State*.

The effect of the window multiplicity increasing rule on the proof representation is in principle like a multiple application of the contraction rule. Thus, the proof representation is changed accordingly.

Window Weakening, Simplification, and Modal Structural Permutation. The window weakening rule, the window simplification rule, and the window modal permutation rule replace a subtree R' by some subtree R'', that may contain less structure. If there was a non-empty window structure f' inside R', then in all three cases a weakened window structure f'' for R'' is derived from R' and f. If there was no window structure inside R', then there is no window structure in R''.

In the first case, assume the active windows in R' with respect to f' are $w_1,\ldots,w_n$ and the active windows in R'' with respect to f'' are $w'_1,\ldots,w'_k$. Due to the definition of f'' it holds for each w'_i that either it is in $w_1,\ldots,w_n$, or it is a parent window of some w_j with respect to f' in R'. We denote by $\text{Children}(w'_i,f',\{w_1,\ldots,w_n\})$ all children of w'_i with respect to f' in $\{w_1,\ldots,w_n\}$. By the structure of the FVIF-tree and the definition of window structures the sets $\text{Children}(w'_i,f',\{w_1,\ldots,w_n\})$ are disjoint sets.

- *Invariant windows*: for each w_i it holds that it is either in $w'_1,\ldots,w'_k$ and we denote the set of those w_i by I;
- *Closed windows:* or there is a w'_j such that $w_i \in \text{Children}(w'_j,f',\{w_1,\ldots,w_n\})$ and we denote the set of these w_i by C;
- *Deleted windows:* or, they are in none of the above categories and we denote that set by D.

For each w_i there is before rule application a proof node

Label	WPState		Window	Justification/Abstraction/Refinement
L	WPS	$\vdash$	w_i	

If $w_i \in I$, then that proof node is justified by a window weakening (or simplification, or modal structural permutation) justification to obtain

Label	WPState		Window	Justification/Abstraction/Refinement
L	WPS	$\vdash$	w_i	$\text{Weakening}(R',R'') : L'$
L'	WPS'	$\vdash$	w_i	

where WPS' is the new window proof state. For each w'_j where $\text{Children}(w'_j,f,\{w_1,\ldots,w_n\}) \neq \emptyset$ we introduce a new proof node with respect to the new proof state WPS'.

Label	WPState		Window	Justification/Abstraction/Refinement
$L_{w'_j}$	WPS	$\vdash$	w'_j	

For each $w_i \in \text{Children}(w'_j,f,\{w_1,\ldots,w_n\})$ we justify w_i's proof node by a window weakening (or simplification, or modal structural permutation) justification to the proof node of w'_j.

Label	WPState		Window	Justification/Abstraction/Refinement
L_{w_i}	WPS	$\vdash$	w_i	$\text{Weakening}(R',R'') : L_{w'_j}$
$L_{w'_j}$	WPS	$\vdash$	w'_j	

Finally, the proof nodes of the $w_i \in D$ are justified by a window weakening (or simplification, or modal structural permutation) justification without successor node.

Label	WPState		Window	Justification/Abstraction/Refinement
L_{w_i}	WPS	$\vdash$	w_i	$\text{Weakening}(R',R'')$

Again, all other open goals before rule application are justified by *Adapt-Window-Proof-State* to

adapt them to the new window proof state and thus ensuring the invariant of the proof representation.

Window Leibniz' Equality, Extensionality, and Boolean ζ-Expansion Rules. All these rules α-insert new subtrees R'' on subtrees R'. Only if there was a non-empty window structure f' in R', then an additional window is set to R'. Otherwise, the domain of the overall window structure is unchanged.

If there was a non-empty window structure in R', then assume $w_1, \ldots, w_n$ are the active windows with respect to f' in R' and w is the new window for R''. Then we justify each proof node of some w_i by a Leibniz' equality (respectively extensionality introduction and boolean ζ-expansion) justification to the two new proof nodes for w_i and w with respect to the new proof state. Thus, the new proof representation is:

Label	WPState		Window	Justification/Abstraction/Refinement
L_1	WPS	$\vdash$	w_1	Leibniz' equality$(R', R'') : L'_1, L$
$\vdots$	$\vdots$		$\vdots$	
L_n	WPS	$\vdash$	w_n	Leibniz' equality$(R', R'') : L'_n, L$
L'_1	WPS'	$\vdash$	w_1	
$\vdots$	$\vdots$		$\vdots$	
L'_n	WPS'	$\vdash$	w_n	
L	WPS'	$\vdash$	w	

If the window structure inside R' was empty, let w be the active window that governs R'. Then its proof node is justified by a Leibniz' equality (respectively extensionality introduction or boolean ζ-expansion) justification to a proof node for w with respect to the new proof state. Thus, the new proof representation is:

Label	WPState		Window	Justification/Abstraction/Refinement
L	WPS	$\vdash$	w	Leibniz' equality$(R', R'') : L'$
L'	WPS'	$\vdash$	w	

Again, in both cases, all other open goals before rule application are justified by *Adapt-Window-Proof-State* to adapt them to the new window proof state and thus ensuring the invariant of the proof representation.

Window Instantiation. The window instantiation rule replaces any literal node R' in which occurs an instantiated variable by an initial FVIF-tree R'' for the instantiated label. In the presence of a non-empty window structure f' inside R', the window tree structure f'' for R'' is obtained from f'. If f' denoted substructures below an instantiated higher-order variable, then f'' is a restriction of f'. This is analogous to the construction of the new window structure when applying the window weakening rule. Otherwise, f'' is isomorphic to f'.

For the definition of how the proof representation is changed, we consider the two cases: (1) the window structure f' inside R' is empty, and (2) the window structure f' inside R' is non-empty.

In the first case, let w be the active window that governs R'. Then the proof node for w is justified by an instantiation justification to the new proof node for w.

Label	WPState		Window	Justification/Abstraction/Refinement
L_w	WPS	$\vdash$	w	Instantiate$(\sigma) : L'_w$

L'_w	WPS	$\vdash$	w

For the second case, assume the active windows in R' with respect to f' are $w_1,\ldots,w_n$ and the active windows in R'' with respect to f'' are $w'_1,\ldots,w'_k$. Due to the definition of f'' either some of the w'_i are either in $w_1,\ldots,w_n$ or are a parent window of some w_j with respect to f' in R'. Assume $w'_1,\ldots,w'_l$ are those contained in $w_1,\ldots,w_n$ and $w'_{l+1},\ldots,w'_k$ are those that are parent windows of some $w_1,\ldots,w_n$. We denote by $\text{Children}(w'_i,f',\{w_1,\ldots,w_n\})$ all children of $w'_i, i \geq l+1$ with respect to f' in $\{w_1,\ldots,w_n\}$. Note that by the structure of the FVIF-tree and the definition of window structures the sets $\text{Children}(w'_i,f',\{w_1,\ldots,w_n\})$, $i \geq l+1$, are disjoint sets.

For each $w'_i \in \{w_1,\ldots,w_l\}$ there is before rule application a proof node

Label	WPState		Window	Justification/Abstraction/Refinement
L	WPS	$\vdash$	w'_i	

That proof node is then justified by an instantiation justification to the new proof node for w'_i with respect to the new proof state:

Label	WPState		Window	Justification/Abstraction/Refinement
L	WPS	$\vdash$	w'_i	$\text{Instantiation}(\sigma,R',R'') : L'$
L'	WPS'	$\vdash$	w'_i	

where WPS' is the new window proof state. For each w'_i, $i \geq l+1$, a new proof node for w'_i with respect to the new proof state WPS' is introduced.

Label	WPState		Window	Justification/Abstraction/Refinement
$L_{w'_i}$	WPS	$\vdash$	w'_i	

Furthermore, for all windows $w_{i_1},\ldots,w_{i_p} \in \text{Children}(w'_i,f',\{w_1,\ldots,w_n\})$ the proof node of w_{i_q} is justified by an instantiation justification to the new proof node for w'_i. Thus, we obtain the following proof representation:

Label	WPState		Window	Justification/Abstraction/Refinement
$L_{w_{i_1}}$	WPS	$\vdash$	w_{i_1}	$\text{Instantiation}(\sigma,R',R'') : L_{w'_i}$
$\vdots$	$\vdots$		$\vdots$	
$L_{w_{i_p}}$	WPS	$\vdash$	w_{i_1}	$\text{Instantiation}(\sigma,R',R'') : L_{w'_i}$
$L_{w'_i}$	WPS	$\vdash$	w'_i	

Note that unlike the window weakening rule, for the window instantiation rule there are no *deleted* windows; all windows are either preserved or closed to some parent window.

Again all other open goals before rule application are justified by *Adapt-Window-Proof-State* to adapt them to the new window proof state and thus ensuring the invariant of the proof representation.

Window Replacement Rule Applications. For the replacement rules we consider only the case of the resolution style replacement rule, since the rewriting style is a combination of that and other inference rules. Assume $WPS = [Q,\sigma \rhd_{\mathcal{L}} (R,f)]$ is the actual window proof state, $u \rightarrow \langle v'_1,\ldots,v'_m\rangle$ is

the rule to apply on some subtree R' of R. If the window structure $f_{\downarrow R'}$ in R' is non-empty, then the application condition requires that there is a v'_i that is isomorphic to R' up to some $v'^1_i, \ldots, v'^n_m$ and the substructures of R' denoted by the active windows with respect to $f_{\downarrow R'}$. If so an f' can be constructed for v'_i and there is a one-to-one correspondence of each active window $w_1, \ldots w_n$ of $f_{\downarrow R'}$ for R' to an active window w'_i of f' for v'_i. In addition to v'_i and f', the other v'_j, $i \neq j$, are β-inserted on a subtree R_j, and new windows are inserted for those if the subtree R_j has a non-empty window structure. Without loss of generality we can assume that windows $w_{v'_1}, \ldots, w_{v'_l}$ are inserted for $v'_1, \ldots, v'_l$ and $i > l$. Each proof node for w_i is justified by a replacement rule application to a proof node for w'_i and the proof nodes $v'_1, \ldots, v'_l$:

Label	WPState		Window	Justification/Abstraction/Refinement
L_1	WPS	$\vdash$	w_1	$\text{Apply}(u \to \langle v'_1, \ldots, v'_m \rangle) : L'_1, L_{v'_1}, \ldots, L_{v'_l}$
$\vdots$	$\vdots$	$\vdots$		
L_n	WPS	$\vdash$	w_1	$\text{Apply}(u \to \langle v'_1, \ldots, v'_m \rangle) : L'_n, L_{v'_1}, \ldots, L_{v'_l}$
L'_1	WPS'	$\vdash$	w'_1	
$\vdots$	$\vdots$	$\vdots$		
L'_1	WPS'	$\vdash$	w'_n	
$L_{v'_1}$	WPS'	$\vdash$	$w_{v'_1}$	
$\vdots$	$\vdots$	$\vdots$		
$L_{v'_l}$	WPS'	$\vdash$	$w_{v'_l}$	

If the window structure $f_{\downarrow R'}$ in R' is empty, then the window w that governs R' also governs v'_i in the new subtree. Like in the first case, in addition to v'_i, the other v'_j, $i \neq j$, are β-inserted on a subtree R_j, and new windows are inserted for those if the subtree R_j has a non-empty window structure. Again, without loss of generality we can assume that windows $w_{v'_1}, \ldots, w_{v'_l}$ are inserted for $v'_1, \ldots, v'_l$ and $i > l$. The proof node for w is justified by a replacement rule application to a proof node for w with respect to the new proof state and the proof nodes $v'_1, \ldots, v'_l$, also with respect to the new proof state:

Label	WPState		Window	Justification/Abstraction/Refinement
L	WPS	$\vdash$	w	$\text{Apply}(u \to \langle v'_1, \ldots, v'_m \rangle) : L', L_{v'_1}, \ldots, L_{v'_l}$
L'	WPS'	$\vdash$	w'	
$L_{v'_1}$	WPS'	$\vdash$	$w_{v'_1}$	
$\vdots$	$\vdots$	$\vdots$		
$L_{v'_l}$	WPS'	$\vdash$	$w_{v'_l}$	

Finally, all other open goals before rule application are justified by *Adapt-Window-Proof-State* to adapt them to the new window proof state and thus ensuring the invariant of the proof representation.

Window Cut. The window cut rule replaces a subtree R' by a subtree R_{Cut} which is of the form $\beta(\alpha(R^+_\varphi, R'), \alpha(R^-_\varphi, R''))$, where R^+_φ and R^+_φ are the subtrees for the cut-formula and R'' is a copy of R'. If the window structure on R' is empty, then no window structure for R_{Cut} is introduced. In that case assume w is the window that governs R'. Then the proof node for w is justified by a cut over φ on R' to a proof node for w with respect to the new proof state. Thus, we obtain:

Label	WPState		Window	Justification/Abstraction/Refinement
L	WPS	$\vdash$	w	$\text{Cut}(\varphi, R') : L'$
L'	WPS'	$\vdash$	w	

If there was a window structure $f \neq \{\}$ for R', then new top-level windows w_1^C, w_2^C are introduced for R_φ^+ and R_φ^+, the window structure on R' is preserved and an isomorphic window structure f' is defined for R''. Thus, if $w_1, \ldots, w_n$ are the active windows of f for R', these windows are still active windows in the new proof state. Furthermore, there is a one-to-one correspondence between $w_1, \ldots, w_n$ and the active windows $w'_1, \ldots, w'_n$ of f' for R''. That window cut proof step is then represented as follows:

Label	WPState		Window	Justification/Abstraction/Refinement
L_1	WPS	$\vdash$	w_1	$\text{Cut}(\varphi, R') : L'_1, L''_1, L_1^C, L_2^C$
$\vdots$	$\vdots$		$\vdots$	
L_n	WPS	$\vdash$	w_n	$\text{Cut}(\varphi, R') : L'_n, L''_n, L_1^C, L_2^C$
L_1^C	WPS'	$\vdash$	w_1^C	
L_2^C	WPS'	$\vdash$	w_2^C	
L'_1	WPS'	$\vdash$	w_1	
L''_1	WPS'	$\vdash$	w'_1	
$\vdots$	$\vdots$		$\vdots$	
L'_n	WPS'	$\vdash$	w_n	
L''_n	WPS'	$\vdash$	w'_n	

And, again, all other open goals before rule application are justified by *Adapt-Window-Proof-State* to adapt them to the new window proof state and thus ensuring the invariant of the proof representation.

This completes the list of CoRE window inference rules and their effects on the proof representation. Before motivating the use of hierarchies in the proof representation in Section 8.1.3, we briefly motivate the *methodological role* of successor nodes in justifications.

8.1.2 Roles of Window Proof Nodes

Consider the justification introduced by a window cut over some formula ψ of the proof node for w_1, that denotes the formula φ. That proof node is related to four proof nodes, namely

Label	WPState		Window	Justification/Abstraction/Refinement
L_1	WPS	$\vdash$	$w_1 := \varphi^+$	$Cut(\psi) : L_1^C, L'_1, L_2^C, L''_1$
L_1^C	WPS'	$\vdash$	$w_1^C := \psi^-$	
L'_1	WPS'	$\vdash$	$w_1 := \varphi^+$	
L_2^C	WPS'	$\vdash$	$w_2^C := \psi^+$	
L''_1	WPS'	$\vdash$	$w'_1 := \varphi^+$	

The windows w_1^C and w_1 are α-related between each other as well as the windows w_2^C and w'_1. From a pure logical point of view *all* successor proof nodes of L_1 subgoals, and each pair is an alternative to prove that subgoal. However, depending on the methodology that underlied the introduction of the cut, different roles can be attributed to these proof nodes: if the cut was used to perform a case analysis

over ψ, then w_1^C and w_2^C are the *case conditions*, while w_1 and w_1' are the subgoals. If the cut was used to perform a speculative proof step to speculate that the goal to prove φ^+ could be refined to the goal to prove ψ^+, then w_2^C is the major subgoal of that proof step, w_1' plays no (methodological) role, and w_2^C and w_1 represent the local lemma that needs to be proved to validate the speculative proof step. Thus, w_2^C and w_1 form the condition of the speculative step.

If this methodological information about the role of proof nodes within justifications would be explicit, then it would be visible to the user and the reasoning engines, and thus contribute to a better understanding of the proof status and the intentions of the proof. Furthermore, it could be exploited by the user and the reasoning engines to organise their proof search, for example to pursue an offensive proof search strategy by always tackling the "major" subgoals first and, only when the proof along those lines succeeds, tackling the various "conditions". Conversely, a defensive strategy can be designed, that always first tackles the "conditions" and then the "major" subgoals.

Furthermore, this information about the role of proof nodes is not only useful to organise the proof search, but also for proof presentation. Indeed, the information about "condition" and "major" subgoals can be fruitfully exploited to explain a completed or partial proof. Also it can be used to display a concise representation of the proof by omitting the proofs of "conditions" and only viewing the proofs of "major" subgoals.

Roles of proof nodes can not only be assigned for the window cut rule, but also for other rule applications, like for instance the application of a replacement rule. Take as an example a replacement rule $\text{Ordered}(X :: Y :: L)^- \rightarrow \langle (X \leq Y)^+, \text{Ordered}(Y :: L)^+ \rangle$ obtained from the definition of a predicate "Ordered" over lists of natural numbers. Assume an open goal in the proof is

Label	WPState		Window	Justification/Abstraction/Refinement
L_0	WPS	$\vdash$	$\text{Ordered}(1 :: 2 :: 3 :: [])^+$	

where :: is concatenation of natural numbers to lists and [] denotes the empty list. Applying the rule on that proof node results in the proof state

Label	WPState		Window	Justification/Abstraction/Refinement
L_0	WPS	$\vdash$	$\text{Ordered}(1 :: 2 :: 3 :: [])^+$	$Apply(\text{Ordered}(X :: Y :: L)^- \rightarrow \langle (X \leq Y)^+, \text{Ordered}(Y :: L)^+ \rangle) : L_1, L_2$
L_1	WPS'	$\vdash$	$(1 \leq 2)^+$	
L_2	WPS'	$\vdash$	$\text{Ordered}(2 :: 3 :: [])^+$	

Although the replacement rule introduces two subgoals, from a methodological point of view the first subgoal L_1 can be seen as a "condition", while L_2 is the "major" subgoal.

Thus, the formal definition of the proof datastructure in Section 8.2 will accommodate the representation of methodological information about the role of subgoals within justifications. Although so far we considered only "condition" and "major" subgoals, other classifications are possible, for instance having a hierarchy for the importance of subgoals. Thus, in the general definition of the proof datastructure we will simply assume a given set of role descriptions.

8.1.3 Hierarchies in Proofs

The proof datastructure is an important means to communicate information about the proof to the partners and it is complementary to the information provided by the proof states. Hierarchies in proofs have been pioneered by the proof planning approach to proof search [Cheikhrouhou & Sorge, 2000]

and have recently been formalised for proof planning in [Fiedler, 2001]. Like the role of proof nodes, the explicit use of hierarchies in proofs serves both the presentation – and thus a better understanding – of the proof as well as the organisation of the proof search. We distinguish two kinds of hierarchies in proofs: the *derivational hierarchy* and the *representational hierarchy*:

- *Derivational hierarchies* are caused by the hierarchies of proof procedures, like for example a proof procedure that causes the execution of another proof procedure is hierarchically higher than the latter. Thereby the causal relationship of the calls to reasoning procedures does not necessarily reflect the chronological order of the executions of the reasoning procedures. Indeed, while in tactical theorem proving, a hierarchically higher tactic is completed *after* the completion of its sub-tactics, in proof planning the high-level method is completed *before* its expansion. The former approach can be described by the introduction of abbreviations for parts of a lower level proof. This is how hierarchies are introduced in tactical theorem proving, where the portion of the proof obtained by the execution of a tactic is abbreviated by the name of that tactic. In the latter approach, parts of a proof are constructed beforehand on some higher level of abstraction by ignoring many logically relevant details. Thereby the refinement of the high-level proof incrementally tackles all the details that have been ignored in the first place. The explicit representation of the relationship between high-level proof steps and the subproof obtained from their refinement is the proof planning approach from which originates the hierarchical view on proofs.
- *Representational hierarchies* result from the use of representational abstractions. Thereby, the proof with respect to the abstract space is hierarchically higher than the corresponding proof steps with respect to the ground space. Although these hierarchies have a clear aspect of a derivational hierarchy, they are mainly due to the change of the representation.

8.1.3.1 Derivational Hierarchies

In this section we briefly sketch how derivational hierarchies are introduced by both proof planning methods and tactics.

The window cut rule supports the representation of high-level proof steps, thus supports the reasoning style advocated by the traditional proof planning paradigm. Taking a risk of over simplification, we can say that a proof planning method M states that from a given proof situation φ we can obtain a new proof situation ψ by using that method. Proof situations correspond to subtrees denoted by an active window. Thus given the following proof situation with the open goal to prove φ

Label	WPState		Window	Justification/Abstraction/Refinement
L	WPS	$\vdash$	φ^+	

we can represent the use of that method by applying a window cut over ψ on φ which results in

Label	WPState		Window	Justification/Abstraction/Refinement
L	WPS	$\vdash$	φ^+	$\mathrm{Cut}(\psi) : L_{\text{Lemma Goal}}, L', L_{\text{Subgoal}}, L_{\text{Lemma Hyp}}$
$L_{\text{Lemma Hyp}}$	WPS'	$\vdash$	ψ^-	
$L_{\text{Lemma Goal}}$	WPS'	$\vdash$	φ^+	
L_{Subgoal}	WPS'	$\vdash$	ψ^+	
L'	WPS'	$\vdash$	φ^+	

Subsequently we apply the window weakening in order to remove the occurrence of φ^+ in the proof node L', and finally obtain

Label	WPState		Window	Justification/Abstraction/Refinement
L	WPS	$\vdash$	φ^+	$\text{Cut}(\psi) : L_{\text{Lemma Goal}}, L', L_{\text{Subgoal}}, L_{\text{Lemma Hyp}}$
$L_{\text{Lemma Hyp}}$	WPS'	$\vdash$	ψ^-	
$L_{\text{Lemma Goal}}$	WPS'	$\vdash$	φ^+	
L_{Subgoal}	WPS'	$\vdash$	ψ^+	
L'	WPS'	$\vdash$	φ^+	Weakening

The actual proof planning step was to refine φ to ψ, i.e. the link from L to L_{Subgoal}, while a subproof for φ^+ under the additional hypothesis ψ^- justifying the node $L_{\text{Lemma Goal}}$ will be the so-called *expansion* of that proof-planning step. We denote that combination of window cut, window weakening, and the respective roles of the subwindow as an *oracle proof step*.

The actual proof planning step is represented by introducing an abstract justification of name M from L to L_{Subgoal} only, which results into

Label	WPState		Window	Justification/Abstraction/Refinement
L	WPS	$\vdash$	φ^+	$M(\psi) : L_{\text{Subgoal}}$ $\text{Cut}(\psi) : L_{\text{Lemma Goal}}, L', L_{\text{Subgoal}}, L_{\text{Lemma Hyp}}$
$L_{\text{Lemma Hyp}}$	WPS'	$\vdash$	ψ^-	
$L_{\text{Lemma Goal}}$	WPS'	$\vdash$	φ^+	
L_{Subgoal}	WPS'	$\vdash$	ψ^+	
L'	WPS'	$\vdash$	φ^+	Weakening

Structurally the proof planning justification $M(\psi, L_{\text{Subgoal}})$ is an abstraction of its expansion, i.e. the lower-level proof that consists of the window cut justification, the subsequent weakening justification, and additional further justifications for $L_{\text{Lemma Hyp}}$ and $L_{\text{Lemma Goal}}$. Note that the proof planning justification contains strictly less successor nodes than the lower-level proof that encodes the proof planning step. When the lemma that validates the proof planning step will be proved, then the proof planning step will be verified, i.e. the proof planning step will have an *expansion*.

We now consider the case of tactics and the hierarchies they introduce into the proof representation. Consider the following low-level partial proof generated by some tactic T:

Label	WPState		Window	Justification/Abstraction/Refinement
L_0	WPS	$\vdash$	$\text{Ordered}(1 :: 2 :: 3 :: [])^+$	$Apply(\text{Ordered}(X :: Y :: L)^- \rightarrow \langle (X \leq Y)^+, \text{Ordered}(Y :: L)^+ \rangle) : L_1, L_2$
L_1	WPS'	$\vdash$	$(1 \leq 2)^+$	
L_2	WPS'	$\vdash$	$\text{Ordered}(2 :: 3 :: [])^+$	$Apply(\text{Ordered}(X :: Y :: L)^- \rightarrow \langle (X \leq Y)^+, \text{Ordered}(Y :: L)^+ \rangle) : L_3, L_4$
L_3	WPS''	$\vdash$	$(2 \leq 3)^+$	
L_4	WPS''	$\vdash$	$\text{Ordered}(3 :: [])^+$	$Apply(\text{Ordered}(X :: []) \rightarrow \langle\rangle) : L_5$
L_5	WPS'''	$\vdash$	$True^+$	

This partial proof can be abbreviated by a single justification of name T from L_0 to *all* resulting subgoals, i.e. L_1, L_3, and L_5. This is represented by

Label	WPState		Window	Justification/Abstraction/Refinement
L_0	WPS	$\vdash$	$\text{Ordered}(1 :: 2 :: 3 :: [])^+$	$T : L_1, L_3, L_5$
				$Apply(\text{Ordered}(X :: Y :: L)^- \rightarrow \langle (X \leq Y)^+, \text{Ordered}(Y :: L)^+ \rangle) : L_1, L_2$
L_1	WPS'	$\vdash$	$(1 \leq 2)^+$	
L_2	WPS'	$\vdash$	$\text{Ordered}(2 :: 3 :: [])^+$	$Apply(\text{Ordered}(X :: Y :: L)^- \rightarrow \langle (X \leq Y)^+, \text{Ordered}(Y :: L)^+ \rangle) : L_3, L_4$
L_3	WPS''	$\vdash$	$(2 \leq 3)^+$	
L_4	WPS''	$\vdash$	$\text{Ordered}(3 :: [])^+$	$Apply(\text{Ordered}(X :: []) \rightarrow \langle\rangle) : L_5$
L_5	WPS'''	$\vdash$	$True^+$	

Note that the abbreviating justification $T : L_1, L_3, L_5$ contains exactly *all* open subgoals of the underlying subproof.

Remark 8.1.1 The abbreviation of proof sequences by single justifications is similar in nature to the introduction of definitions. However, in our understanding, definitions are used to introduce new major concepts, while abbreviations are just short-cuts and do not necessarily introduce a new concept.

8.1.3.2 Representational Hierarchies

Representational hierarchies in proofs result from the use of representational abstractions and refinements. Consider as an example the following lemma over the integers:

$$\forall n_{Nat} \,.\, n \geq 0 \Rightarrow n \times (n+1) = (\sum_{i=1}^{n} i) + (\sum_{i=1}^{n} i).$$

After focusing on the relevant parts of that formula, i.e. $n \geq 0$, $n \times (n+1)$, and $(\sum_{i=1}^{n} i) + (\sum_{i=1}^{n} i)$, we are in the following proof situation:

Label	WPState		Window	Justification/Abstraction/Refinement
L_1	WPS	$\vdash$	$n \times (n+1)$	
L_2	WPS	$\vdash$	$(\sum_{i=1}^{n} i) + (\sum_{i=1}^{n} i)$	
L_3	WPS	$\vdash$	$n \geq 0^-$	

where *WPS* is the actual window proof state. Using the abstraction function to diagrams (cf. Section 7.1) it maps $n \times (n+1)$ to $\text{square}(n, n+1)$, $(\sum_{i=1}^{n} i) + (\sum_{i=1}^{n} i)$ to $\text{triangle}(n,n) + \text{triangle}(n,n)$, and does not map $n \geq 0^-$, which is only used, for instance, to ensure the applicability of the abstraction. The resulting proof situation is then

Label	WPState		Window	Justification/Abstraction/Refinement
L_1	WPS	$\vdash$	$n \times (n+1)$	$\text{Diagrams} : L'_1$
L_2	WPS	$\vdash$	$(\sum_{i=1}^{n} i) + (\sum_{i=1}^{n} i)$	$\text{Diagrams} : L'_2$
L_3	WPS	$\vdash$	$n \geq 0^-$	
L'_1	WPS'	$\vdash$	$\text{square}(n,n)$	
L'_2	WPS'	$\vdash$	$\text{triangle}(n,n) + \text{triangle}(n,n)$	

Continuing the proof with respect to the diagrammatic representation transforms the windows to

equalise their content to obtain the proof situation

Label	WPState		Window	Justification/Abstraction/Refinement
L_1	WPS	$\vdash$	$n \times (n+1)$	Diagrams : L'_1
L_2	WPS	$\vdash$	$(\sum_{i=1}^{n} i) + (\sum_{i=1}^{n} i)$	Diagrams : L'_2
L_3	WPS	$\vdash$	$n \geq 0^-$	
L'_1	WPS'	$\vdash$	$\text{square}(n,n)$	DiagramProof : L''_1
L'_2	WPS'	$\vdash$	$\text{triangle}(n,n) + \text{triangle}(n,n)$	DiagramProof : L''_2
L''_1	WPS''	$\vdash$	$\text{triangle}(n,n) + \text{triangle}(n,n)$	
L''_2	WPS''	$\vdash$	$\text{triangle}(n,n) + \text{triangle}(n,n)$	

Finally the abstract proof is refined to a proof with respect to the original representation. That proof is for instance done by induction, in our case over the positive integers including 0. This transforms on the one hand $n \times (n+1)$ to both $0 \times (0+1)$ and $(n+1) \times ((n+1)+1)$ which are the refinements of L''_1. On the other hand it tranforms $(\sum_{i=1}^{n} i) + (\sum_{i=1}^{n} i)$ to $0 \times (0+1)$ and $(n+1) \times ((n+1)+1)$, which are refinements for L''_2.

For the refinement note that the abstraction relation (R_A in Definition 7.2.4) is $\{(L_1, L'_1), (L_2, L'_2)\}$ and the derivational relation (R_D in Definition 7.2.4) is $\{(L'_1, L''_1), (L'_2, L''_2)\}$. Thus, $R_D \circ R_A$ is the relation $\{(L_1, L''_1), (L_2, L''_2)\}$ and the refinements of the abstract proof step DiagrammProof must indeed refine L''_1 to both L'''_{11} and L'''_{12}, which are the successor nodes of L_1 (analogously for L''_2).

Label	WPState		Window	Justification/Abstraction/Refinement
L_1	WPS	$\vdash$	$n \times (n+1)$	Diagrams : L'_1 DiagProofRefine : L'''_{11}, L'''_{12}
L_2	WPS	$\vdash$	$(\sum_{i=1}^{n} i) + (\sum_{i=1}^{n} i)$	Diagrams : L'_2 DiagProofRefine : L'''_{21}, L'''_{22}
L_3	WPS	$\vdash$	$n \geq 0^-$	
L'_1	WPS'	$\vdash$	$\text{square}(n, n+1)$	DiagramProof : L''_1
L'_2	WPS'	$\vdash$	$\text{triangle}(n,n) + \text{triangle}(n,n)$	DiagramProof : L''_2
L''_1	WPS''	$\vdash$	$\text{triangle}(n,n) + \text{triangle}(n,n)$	DiagramRefine : L'''_{11}, L'''_{12}
L''_2	WPS''	$\vdash$	$\text{triangle}(n,n) + \text{triangle}(n,n)$	DiagramRefine : L'''_{21}, L'''_{22}
L'''_{11}	WPS'''	$\vdash$	$(\sum_{i=1}^{0} i) + (\sum_{i=1}^{0} i)$	
L'''_{12}	WPS'''	$\vdash$	$(\sum_{i=1}^{0} i) + (\sum_{i=1}^{0} i)$	
L'''_{21}	WPS'''	$\vdash$	$(\sum_{i=1}^{n+1} i) + (\sum_{i=1}^{n+1} i)$	
L'''_{22}	WPS'''	$\vdash$	$(\sum_{i=1}^{n+1} i) + (\sum_{i=1}^{n+1} i)$	

In the next section we formally define the hierarchical proof datastructure. It accommodates all the aspects of a hierarchical proof datastructure sketched so far. Additionally it explicitly represents the direction of the introduction of an abbreviation, like upwards for the abbreviation of a proof sequence generated by a tactic, or downwards like between a proof planning step and its expansion. This explicit representation of the directions in the construction of the hierarchy will serve as one formal property that allows to distinguish top-down proof constructions à la proof planning from bottom-up proof construction à la tactical theorem proving and other automatic proof search procedures. Furthermore, the distinction between abbreviating justifications that contain *all* open subgoals of the underlying proof versus abbreviating justifications that only contain *some* of them is another formal property which allows to distinguish between top-down and bottom-up proof construction paradigms. Those two properties resulting from a proof representation that accommodates both reasoning paradigms while

showing their differences may serve as a starting point for a comparison of the different reasoning paradigms.

8.2 Hierarchical Proof Datastructure

For the definition of the hierarchical proof datastructure we build upon the formalisation of the proof planning datastructure from [Fiedler, 2001] and adapt and extend it to fit our context.

The inference rules are the names in justifications: we distinguish between *formal* and *informal* inference rules, where the formal inference rules are the CORE window inference rules while informal inference rules are names or descriptions, like names of tactics or proof planning methods, or any kind of description for a portion of a proof provided for instance by the user.

Definition 8.2.1 (Inference Rules) The inference rules are given by a pair $\mathbb{I} = (F, I)$ of formal and informal inference rules. The formal inference rules are the CORE window inference rules. The informal inference rules are arbitrary descriptions. ∎

As sketched in Section 8.1.2 the successor proof nodes in justifications can have different methodological roles. For the general definition of the hierarchical proof datastructure we assume an arbitrary but fixed set of roles. We do not impose that there must be an order among the roles, for instance to indicate their relative importance. However, this is possible in order to allow for the distinction between "conditions" and "major" subgoals as sketched in Section 8.1.2. This can simply be modelled by using a binary set of roles $\{\text{Condition}, \text{SubGoal}\}$ together with the order $\text{Condition} < \text{SubGoal}$.

Definition 8.2.2 (Roles) The roles $\mathbb{R}$ are given by an arbitrary finite set, possibly with an ordering (not necessarily total) among the elements of $\mathbb{R}$. ∎

We now introduce the actual objects that define the hierachical proof datastructure. As sketched in the previous sections, it is composed of window proof nodes that are annotated by justifications. We first define *justifications* as follows:

Definition 8.2.3 (Justifications) Let $\mathbb{I}$ be a set of inference rules and $\mathbb{R}$ a set of roles. A *justification* is a 3-tuple $J = (R, \overline{P}, \overline{NR})$, where R is an inference rule from $\mathbb{I}$, $\overline{P}$ a list of parameters for R, and $\overline{NR}$ a list of window proof nodes annotated by roles, called *successor nodes*. A justification containing a CORE window calculus rule is called a *formal justification*. ∎

Definition 8.2.4 (Representational Abstraction & Refinement Applications) Let $\mathbb{A}$ be a set of representational abstractions (cf. Definition 7.2.3). A *representational abstraction application* is a 3-tuple $A = (a, \overline{P}, \overline{N})$, where a is a representational abstraction from $\mathbb{A}$, $\overline{P}$ a list of parameters for a, and $\overline{N}$ a list of window proof nodes, called *abstraction nodes*.

A *representational refinement application* is a 3-tuple $R = (a, \overline{P}, \overline{N})$, were a is a representational abstraction from $\mathbb{A}$, $\overline{P}$ a list of parameters for a, and $\overline{N}$ a list of window proof nodes, called *refinement nodes*. ∎

Based on this we define *window proof nodes*. In this definition we use the notion of *directed justification sequences*, which is only introduced afterwards. For the moment we can assume a directed justification sequence to be a set of justifications.

Definition 8.2.5 (Window Proof Nodes) A *window proof node* is a 5-tuple $N = (WPS, w, \overline{J}, \overline{A}, \overline{R})$, where WPS is a window proof state, w an active window of WPS, $\overline{J}$ a *directed justification sequence*, $\overline{A}$ representational abstraction applications, and $\overline{R}$ representational refinement applications. We say that any justification J in $\overline{J}$ *justifies* N. If N is justified by the axiom rule, i.e. has no successor nodes, then N is a *hypothesis*.

The *support nodes* of N is the transitive closure of all successor nodes of N. ∎

Remark 8.2.6 For the definition of representational refinements (cf. Definition 7.2.4) we assumed for a derivation $WPS \longmapsto_W^* WPS'$ the existence of a relationship R_D between active windows in WPS to active windows in WPS'. In the proof representation there is a proof node for each active window in WPS. During the derivation of WPS' those are related via justifications to new proof nodes, that belong to active windows in the new proof state WPS'. For each initial active window the active windows of the support nodes are all windows in that relation. Thus, the assumed relation R_D is statically determined from the proof representation, and can be used to check the representational refinement relationships.

Window proof nodes together with their justifications define the graph structure of the hierarchical proof datastructure. The hierarchies in proof datastructures are based on the following notion of proof graphs.

Definition 8.2.7 (Proof Graphs) Let $\mathbb{N}$ be a set of window proof nodes, let $\mathbb{S} \subseteq \mathbb{N}$ and $N \in \mathbb{N}$. $\mathcal{N}$ is a *proof graph of N from $\mathbb{S}$* if, and only if, one of the following holds:

1. $N \in \mathbb{S}$.
2. Let $N = (WPS, w, \overline{J}, \overline{A}, \overline{R})$.
   (a) For each justification $J = (R, \overline{P}, \overline{NR})$ in $\overline{J}$ and for each successor node N' from $\overline{NR}$ there is a set $\mathbb{N}' \subsetneq \mathbb{N}$ that is a proof graph of N' from $\mathbb{S}$,
   (b) For each representational abstraction application $A = (a, \overline{P}, \overline{N})$ in $\overline{A}$ and for each abstraction node N' from $\overline{N}$ there is a set $\mathbb{N}' \subsetneq \mathbb{N}$ that is a proof graph of N' from $\mathbb{S}$, and
   (c) For each representational refinement application $R = (a, \overline{P}, \overline{N})$ in $\overline{R}$ and for each refinement node N' from $\overline{N}$ there is a set $\mathbb{N}' \subsetneq \mathbb{N}$ that is a proof graph of N' from $\mathbb{S}$.

$\mathcal{N}$ is a *formal proof graph of N from $\mathbb{S}$* if, and only if, N and each support node of N in $\mathcal{N}$ have a formal justification. ∎

Since $\mathbb{N}' \subsetneq \mathbb{N}$ this clearly defines an acyclic graph.

Based on proof graphs we define the derivational hierarchy in a proof datastructure. The hierarchy is induced by the relationship between justifications and a proof graph for that justification. This induces an ordering $\prec$ among justifications, which is used to define directed justification sequences. Those are pairs of non-disjoint sets of justifications, each being partially ordered with respect to $\prec$ and their union is required to be totally ordered with respect to $\prec$. The motivation for having two sets of justifications is to represent the direction of the hierarchy. For two justifications J, J' that are in the first set and for which $J \prec J'$ holds, then J has been abstracted to J'. Conversely, if the justifications are from the second set and $J \prec J'$ holds, then J' has been refined to J.

Definition 8.2.8 (Directed Justification Sequences) Let $N = (WPS, w, \overline{J}, \overline{A}, \overline{R})$ be a window proof node, $J = (R, \overline{P}, \overline{NR})$ be a justification in $\overline{J}$, and $\mathbb{S} = \{N \mid N \text{ in } \overline{NR}\}$.

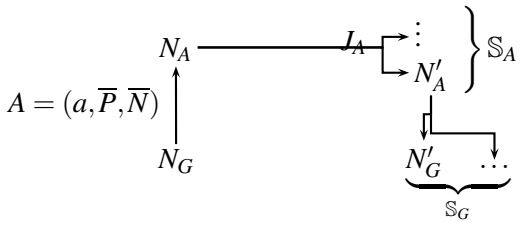


Figure 8.1: Representational expansion.

---

1. The *derivational expansion* of J is a set $\mathcal{E}_J$ of window proof nodes that constitutes a proof graph of $N^{\setminus J}$ from $\mathbb{S}$, where $N^{\setminus J}$ denotes N with J deleted from its directed justification sequence. We say the expansion $\mathcal{E}_J$ *refines* J. We say the expansion $\mathcal{E}_J$ is *complete* if, and only if, $\mathcal{E}_J$ is a formal proof graph of N from $\mathbb{S}$. If $\mathcal{E}_J$ is a complete derivational expansion, we say the expansion $\mathcal{E}_J$ of J *proves* J.

2. A justification $J = (R,\overline{P},\overline{NR})$ is *more abstract* than a justification $J' = (R',\overline{P}',\overline{NR}')$ (we write $J \succ J'$ or $J' \prec J$) if, and only if, there is a justification J'' such that J'' justifies N in the derivational expansion of J and either $J'' = J'$ or $J'' \succ J'$.

3. A *directed justification sequence* is a pair (J_A,J_R) of sets of justifications, such that J_A and J_R are partially ordered with respect to $\succ$. For any $J,J' \in J_A$ such that $J \succ J'$ we say that J' has been *abstracted* to J. Analogously, for any $J,J' \in J_R$ such that $J \succ J'$ we say that J has been *refined* to J'.

   We say that a directed justification sequence is *complete* if, and only if, $J_A \cup J_R$ is totally ordered with respect to $\succ$. ■

The distinction between complete and incomplete justification sequences in the above definition is needed, since we are interested in representing intermediate states of the proof, and not only completed proofs as in [Fiedler, 2001]. The notion of direct justification sequences, more specifically the notions of derivational expansions and the induced ordering among justifications, represent the *derivational hierarchies* in proofs. Analogously to the notion of expansions we define *representational expansions* of justifications, which correspond to the *representational hierarchies* in proofs.

Definition 8.2.9 (Representational Expansions) Let $N_G = (WPS_G,w_G,\overline{J}_G,\overline{A}_G,\overline{R}_G)$ be a window proof node, $A = (a,\overline{P},\overline{N})$ from $\overline{A}_G$, $N_A = (WPS_A,w_A,\overline{J}_A,\overline{A}_A,\overline{R}_A)$ a window proof node in $\overline{N}$, $J_A = (R_A,\overline{P}_A,\overline{NR}_A)$ a justification in $\overline{J}_A$, $\mathbb{S}_A = \{N'_A \mid N'_A \in \overline{NR}_A\}$, and $\mathbb{S}_G = \bigcup_{(WPS'_A,w'_A,\overline{J}'_A,\overline{A}'_A,\overline{R}'_A)\in\mathbb{S}_A} \{N'_G \mid N'_G \in \overline{R}'_A\}$ (cf. Figure 8.1). Then

1. The *representational expansion of* J_A is a set $\mathcal{ER}_{J_A}$ of window proof nodes that constitutes a proof graph of $N_G^{\setminus(a,\overline{P},\overline{N})}$ from $\mathbb{S}_G$, where $N_G^{\setminus(a,\overline{P},\overline{N})}$ denotes the node N_G with $(a,\overline{P},\overline{N})$ deleted from its representational abstraction applications.

2. The justification J_A is *representationally more abstract* than a justification J (we write $J_A \succ_A J$ or $J \prec_A J_A$) if, and only if, there is a justification J' that justifies N_G in the representational expansion of J_A and either $J' = J$ or $J' \succ J$. ■

Finally, we define the hierarchical proof datastructure as follows:

Definition 8.2.10 (Hierarchical Proof Datastructure) A *hierarchical proof datastructure* is a 3-tuple $P = (\varphi, C, \mathcal{N})$, where φ is a closed formula with respect to some logic $\mathcal{L}$, $\mathcal{N}$ is a set of window proof nodes, $C = (WPS, w, \overline{J}, \overline{A}, \overline{R})$ in $\mathcal{N}$ is a window proof node where *WPS* is the initial window proof state for φ with initial root window w. The *open goals* of P are those window proof nodes from $\mathcal{N}$ that have no directed justifications. ■

Definition 8.2.11 (Complete Hierarchical Proof Datastructures) Let $P = (\varphi, C, \mathcal{N})$ be a hierarchical proof datastructure and $\mathbb{H}$ the hypothesis in the support nodes of C. The hierarchical proof datastructure $P = (\varphi, C, \mathcal{N})$ is *complete* if, and only if, $\mathcal{N}$ constitutes a formal proof graph of C from $\mathbb{H}$. ■

Finally, we define the notion of a *pure* CORE *window proof*. A CORE window proof is obtained from a hierarchical proof datastructure by removing all non-formal justifications.

Definition 8.2.12 (Pure CORE Proof Datastructure) Let $P = (\varphi, C, \mathcal{N})$ be a hierarchical proof datastructure. Then P is a *pure* CORE *proof datastructure* if, and only if, all nodes in $\mathcal{N}$ have only formal justifications and no representational abstraction applications.

A pure CORE proof datastructure P is *complete* if, and only if, P is complete. ■

8.3 Proof Paths and Dependencies

In this section we introduce the notion of paths between window proof nodes and qualitative descriptions of the dependencies between window proof nodes. These notions are used in Section 9.5 for the interface between reasoning procedures and the CORE window reasoning rules.

Definition 8.3.1 (Proof Paths) Let $P = (\varphi, C, \mathcal{N})$ be a hierarchical proof datatstructure and $N, N' \in \mathcal{N}$. A *path* between N and N' is a possibly empty sequence of justifications $\langle J_1, \ldots, J_n \rangle$, such that

- if the sequence is empty ($n = 0$), then it must hold $N = N'$,
- otherwise $J_1 = (R, \overline{P}, \overline{NR})$ must be justification of N, and there is an $N'' \in \overline{NR}$, such that $\langle J_2, \ldots, J_n \rangle$ is a path from N'' to N'.

We say that N *depends on* N' if, and only if, there is a path from N to N'. ■

The dependencies between proof nodes can be further categorised: if between two window proof nodes N, N' there is path p from N to N' that consists only of *Adapt-Window-Proof-State*-justifications, then only the context of the windows along that path has been changed, but never the windows themselves. In that case we say that N is *passively* justified by N'. Otherwise, we say that N is *actively* justified by N' via the justifications in p.

Definition 8.3.2 (Active & Passive Proof Paths) Let $P = (\varphi, C, \mathcal{N})$ be a hierarchical proof datatstructure and $N, N' \in \mathcal{N}$ such that there is a non-empty path $\langle J_1, \ldots, J_n \rangle$ between N and N'. If all J_i are *Adapt-Window-Proof-State* justifications, then N is *passively* justified by N' via that path. Otherwise, N is *actively* justified by N' via that path. ■

8.4 Categories of Justifications

In the introduction we motivated a categorisation of proof construction steps. We introduced the categories of *intra-level* (respectively *horizontal*) proof steps and *inter-level* (respectively *vertical*) proof steps. In the following we apply this informal categorisation to the hierarchical proof datastructure introduced to represent CoRe proofs.

The intra-level proof steps where divided into those that verifiably refine a goal into subgoals (local lemma application) and those that speculate about the subgoals (local lemma speculation). Thus, all justifications that are introduced by CoRe window inference rules, i.e. the formal justifications, are local lemma application proof steps.

A justification that has been introduced by a proof planning step on some window proof node N is a local lemma speculation proof step as long as there is no derivational expansion for that justification. As soon as such a justification has a derivational expansion, the justification turns into a local lemma application proof step, since it is validated by the expansion. An expansion for a proof planning step is obtained by proving the condition of the cut proof step that encoded the proof planning step. Thus, the complete proof of the condition together with the cut justification are an expansion of the proof planning step. Proving the condition is a vertical refinement proof step, and the vertical refinement is explicitly represented by putting the window cut rule justification J_C and the justification J representing the proof planning step into the J_R set of justifications of N. As soon as the condition of the proof planning step is proved and thus the cut justification is the starting justification of an expansion of J, it holds $J \succ J_C$. Note that the proof of the condition needs not to be a formal proof, since further speculative steps may be involved in that proof. However, the proof planning step and its expansion are proof steps on different levels of abstractions, and thus it is legitimate to say that the proof planning step is a local lemma application proof step on its level of abstraction, which is validated by the proof graph that defines its expansion.

A justification that has been introduced as an abbreviation for a proof graph for some node N is a local lemma application proof step on a higher level of abstraction than the abbreviated proof graph. Examples for such abbreviations are the abbreviation of a portion of a proof that has been generated by some tactic. Another possibility is that the user abbreviates a portion of the proof by some high-level description for it. The action of abbreviating a portion of a proof is itself a vertical abstraction proof step, and thus the starting justification J_P of the abbreviated proof graph and the abbreviating justification J are inserted in the set J_A of the justifications of N. Since the proof graph is an expansion of J by definition of an abbreviation, it holds $J \succ J_A$.

The above discussion serves as a basis for the following definition of a static categorisation of justifications contained in a hierarchical proof datastructure.

Definition 8.4.1 (Categories of Justifications) Let $P = (\varphi, C, \mathcal{N})$ be a hierarchical proof datastructure, $N = (WPS, w, (J_A, J_R))$ and J, J' justifications from (J_A, J_R). Then we say

- J is a local lemma application justification if, and only if, either it is a formal justification or there is a justification J_E in J_R such that $J \succ J_E$, i.e. J_E is the starting justification of an *expansion* for J.
- J is a local lemma speculation justification if, and only if, it is not a local lemma application justification, i.e. it is neither a formal justification nor has an expansion.
- the pair (J, J') is a vertical refinement proof step of J to J' if, and only if, J, J' are in J_R and it holds $J \succ J'$.

– the pair (J, J') is a vertical abstraction proof step of J' to J if, and only if, J, J' are in J_A and it holds $J \succ J'$. ■

8.5 Backtracking

Proof attempts may fail and the need arises to resume the proof construction at an earlier stage. In computer based proof construction, whether interactive or fully automatic, these failures need to be recognised by the proof procedures and the underlying proof construction system needs to support backtracking parts of a proof. As usual, backtracking is achieved in CoRE via the proof history back to a specific window proof state. If a justification is pruned from the proof representation, all justifications that depend on it need to be pruned first. The proof history explicitly represented in the proof datastructure induces a total ordering among the window proof states. Each window proof state is a *backtracking point* and backtracking is achieved along the induced total ordering among window proof states. That chronological backtracking may remove more proof steps than necessary, as for instance there may be reasoning rule applications that are not conceptually relevant for the failed proof attempt. However, whether proof steps belong to a proof attempt or not requires to know the strategic information that underlied the proof attempts. That knowledge is not always explicitly represented and thus we leave the decision whether a justification belongs to a proof attempt or not to specific backtracking procedures. Backtracking procedures can be defined as explicit reasoning procedures or be part of say the implementation of a tactic programming language with a failure and success semantics. The backtracking procedure can decide which justifications are not part of a proof attempt, memorise those proof steps, and, after performing the chronological backtracking, automatically reconstruct these parts of the proof. This can for example be achieved by a replay mechanism integrated in the implementation of some tactic language (see also Section 9.1).

8.6 Summary

The explicit representation of the proof history provides complementary information about the proof to the user and the reasoning engines. The datastructure is a *hierarchical proof datastructure* that contains CoRE window reasoning proof steps as well as the proof hierarchies that are inherent in theorem proving. Furthermore, the informal categories of intra-level and inter-level proof construction steps from Section 1.1.2 can be formally defined in a natural way for the hierarchical proof datastructure. The *methodological roles* of subgoals introduced by proof steps are represented in the proof datastructure and ease the understanding of the proof. We refrained from giving a general formal definition of methodological roles as that information is purily heuristical. However, this does not hamper the formal definition of *specific* methodological roles for a specific theorem proving procedure. The role information can be exploited on the one hand to organise the proof search and during the design of proof search procedures. On the other hand this information provides a further flexibility for an intuitive presentation of a proof. For example the main idea of a proof can be presented by only presenting the "main" subgoals – according to the methodological categorisation – and omit the "conditions".

Applications

–

Part IV

Chapter 9

Interface for Reasoning Procedures

A theorem prover, whether interactive or fully automatic, consists of a set of reasoning procedures. In this chapter we describe the interface of CoRe to (semi-)automatic reasoning procedures. The major problem for the design of automatic reasoning procedures is the large number of possible replacement rules for each subtree. This large number of replacement rules is no surprise and is a sign for the flexibility during proof search provided by CoRe. However, the flexibility must be controlled when designing reasoning procedures. In order to put the proposed solutions into context, we sketch in Section 9.1 a sample tactic language [Schairer *et al*, 2001] implemented on top of CoRe which is used for the implementation of InKa 5.0 [Autexier *et al*, 1999]. The interplay of backtracking implemented in the tactic language and backtracking of the proof state and proof representation is described in Section 9.2. In Section 9.3 we prove that the number of replacement rules is exponential in the number of α- and β-type connectives, before defining the notion of filters in Section 9.4. Filters are inspired by heuristic categorisations implemented in [Hutter & Sengler, 1996] and they are the solution we propose for a flexible definition of control knowledge to restrict the selection of possible replacement rules. Finally, in Section 9.5 we show how the tactic executions can be represented in the hierarchical proof datastructure and how thereby the hierarchy among tactics – implicitly induced by tactic definitions – is made explicit in the hierarchies of the proof datastructure.

9.1 The Tactic Language

The semantics of tactics is defined in continuation passing style which is a standard technique by now (cf. [Abelson *et al*, 1996, Graham, 1994, Reynolds, 1993, Norvig, 1992, Elliott & Pfenning, 1991, Carlsson, 1984]). We only present a brief description of the tactic language here. The idea is to describe the evaluation of an expression e with respect to a continuation S which represents the future of the computation after e has been evaluated. A continuation S is a function of one argument. In order to evaluate e with respect to S, the continuation S is applied to the value of e. Evaluation of a literal l with respect to S is, therefore, described by $S(l)$. Evaluation of complex expressions is decomposed in such a way that a part of the expression is evaluated with respect to a new continuation which combines the rest of the evaluation of the complex expression and also considers the original continuation. As an example, if e is a list of expressions $e_1, e_2, \ldots, e_n$, its evaluation relative to S can be described by saying that e_1 should be evaluated (assume the result is h) with respect to the continuation that evaluates $e_2, \ldots, e_n$ (assume the result is t) and then applies the original continuation S to the list composed of the head h and the tail t. Evaluation of complex expressions can be decomposed until

the expression to evaluate is a literal or variable and the rest of the evaluation is encoded in the success continuation. This defines a non-standard interpreter for the tactic programming language in continuation passing style, i.e. we can view the definition operationally as the way in which a tactic expression is executed. The general idea is that evaluation is flattened (or sequentialised) into a sequence of basic evaluation steps. This is in contrast to the standard evaluation strategy, which evaluates an expression by evaluating its subexpressions along its abstract syntax tree, and does not explicitly sequentialise the evaluation.

Tactics can be defined and named, for examples as in "tactic $f(x) = e$", where the formal parameter x of the named tactic f may be free in the body e. Tactics may call other named tactics to solve subgoals of the current goal. Depending on the proof representation this necessitates to focus on the subproblem before evaluating the body of the called tactic. It may also necessitate cleaning up after evaluating the body, depending on the concrete proof representation. The semantics has to take care that setting and cleaning up are done appropriately, i.e. before and after a tactic is called. Note that the interpreter knows which named tactic is being called and the actual values it is called with when it sets up the proof state. So it is possible to make the proof state dependent on the tactic that is called. In Section 9.5 we present how this close relationship between the actual proof state and the called tactic is made explicit by introducing appropriate tactic justifications into the hierarchical proof datastructure of CoRe.

9.2 Backtracking

The basic tacticals are the window reasoning rules provided by CoRe. Further tacticals are defined in the tactic language that support the combination of named and unnamed tactics to define complex tactics. Among others, the tactic language includes a special operator `choose` which implements search. It evaluates its argument, which should be a list of alternatives, and then chooses the first alternative for which the remaining part of the evaluation is successful, i.e. does not fail. If there is no such alternative, `choose` fails. As usual this is realised by backtracking and can be done straightforwardly in the setting of continuations by using a second continuation F that encodes the future of the evaluation if the evaluation of an expression fails. Again this is a standard technique covered in the literature (cf. [Abelson *et al*, 1996]). Tactics can explicitly call `fail` which is an abbreviation for `choose`([]) and fails straightaway. In order for this to work properly, side effects of tactics need to be restricted such that they can be backtracked over.

In particular, because there is an interaction between the evaluation of tactic expressions and the proof representation, the semantics has to ensure that the proof representation is adjusted when backtracking occurs. This is important in particular when updating the proof representation is done by destructively updating proof data structures. In this case, since evaluation of a tactic expression can modify the proof state between the time a choice point is set up and the time the subsequent computation fails, the proof state has to be remembered before a choice is tried and has to be rewound to the remembered state before a new alternative is tried. This has been built into the semantics of `choose` in such a way that only one computation for each choice point, corresponding to trying the first successful choice from the list of alternatives, has left modifications in the proof representation.

The net effect is that the proof representation is always well-formed and only contains parts that correspond to successful evaluation paths. Furthermore, each successful execution of a tactic can be associated with a well-defined part of the proof representation (cf. Section 9.5).

For each backtracking point the actual proof state is stored. Each proof state corresponds to a set of proof nodes in the proof datastructure (see Chapter 8). Thus, for the purpose of backtracking it is

sufficient to save the proof state before tactic execution. If a tactic execution is backtracked, the proof state, and thus the the proof representation, is backtracked to the initial window proof state which is associated to that tactic (cf. Section 8.5). However, saving a whole proof state before each tactic execution is expensive, and in the implementation the proof state is transformed by rule applications and the differences between two proof states are saved. The differences are explicitly maintained in the proof representation and are used to backtrack to a previous proof state by applying the inverse differences.

9.3 Replacement Rules

As mentioned above, the basic tacticals are the CoRe window reasoning rules. While most of these rules involve a non-deterministic choice, like the selection of subtrees to open subwindows, the branching factor for the application of replacement rules has a particularly high complexity. It results from the worst case analysis for the number of replacement rules for some subtree of the FVIF-tree. We denote by $\mathrm{Rules}(R')$ the set of all replacement rules for some subtree R'.

Lemma 9.3.1 (Complexity) Let R be a FVIF-tree of depth d and with n_α α-type nodes and n_β β-type nodes. Then the cardinality of $\mathrm{Rules}(R)$ is $O(d^2 \times 2^{n_\alpha + n_\beta})$. ∎

Proof. For each subtree R' the number of possible left-hand sides for a replacement is limited by the number of α-related subtrees and their depth. A subtree of depth d contains $O(d^2)$ subtrees. For each of these subtrees R_I, there are at most n_β-related subtrees R_V to obtain the right-hand sides of a replacement rule. For each R_V there are at most $O(2^{n_\alpha})$ possible weakened subtrees. Thus, for each R_I there are at most $O((2^{n_\alpha})^{n_\beta}) = O(2^{n_\alpha + n_\beta})$ possible replacement rules. Hence, the worst case complexity for the number of replacement rules for each R' is $O(d^2) \times O(2^{n_\alpha+n_\beta}) = O(d^2 \times 2^{n_\alpha+n_\beta})$. ∎

This huge number of possible replacement rules must be handled during automatic proof search, as it is certainly not feasible to always generate all possible replacement rules. For the determination of a replacement rule from the context, two kinds of information can be used to restrict the search: first, restrictions can be imposed to determine possible left-hand sides of rules. Examples for this are that only leaf-nodes are used as left-hand sides, or only those subtrees are considered as left-hand sides, if their label shares certain constant symbols with the formula denoted by a window. Secondly, the possible right-hand sides of a rule can be restricted by providing information about how or how not to weaken the respective subtrees.

9.4 Filters

For the interface to automatic reasoning procedures with introduce the notion of *filters* that are used to generate only these replacement rules that fulfill the requirements described in the filter. A filter consists of two parts, one for the description of possible left-hand sides, and a second for the description how to weaken the β-related subtrees to determine the right-hand sides.

Left-hand sides of replacement rules are either FVIF-trees or left- or right-hand sides of ε-type leaf nodes. Thus, the description of possible left-hand sides of rules is defined as a predicate over FVIF-trees and ε-type nodes R of label $\varepsilon(s,t)$ together with a subterm occurrence π that denotes either s or t, i.e. $\pi = [1]$ or $\pi = [2]$. We denote by $\mathcal{R}$ the set of all FVIF-trees and by $\mathcal{E}$ all ε-type nodes in $\mathcal{R}$

annotated by either [1] or [2]. Given $R_{[1]} \in \mathcal{E}$ its *complementary* element is $R_{[2]}$ and vice-versa. Then the description of adequate left-hand side is a predicate over $\mathcal{R} \uplus \mathcal{E}$.

Right-hand sides of replacement rules are determined by the left-hand side and the weakening of the β-related formulas. If the left-hand side is from $\mathcal{E}$, then the right-hand sides contains the complementary element and the weakened subtrees of the β-related subtrees. Otherwise, the right-hand sides are composed of the weakened subtrees of subtrees β-related to the left-hand side. However, for each left-hand side there may be several possible lists of right-hand sides, due to weakening. Thus, the description of the right-hand sides is a binary predicate over $(\mathcal{R} \uplus \mathcal{E}) \times (\mathcal{R} \uplus \mathcal{E})^*$ that holds for those combination of left- and right-hand sides that are in the domain of the filter.

Definition 9.4.1 (Filter) A *filter* $F := \langle P_L, P_R \rangle$ is composed of a predicate P_L over $\mathcal{R} \uplus \mathcal{E}$ to filter admissible left-hand sides and a binary predicate P_R over $(\mathcal{R} \uplus \mathcal{E}) \times (\mathcal{R} \uplus \mathcal{E})^*$ to filter admissible replacement rules with admissible left-hand sides. The *filter function* for F is the function ∇_F that assigns to each subtree R the set of rules that are admissible with respect to the filter. The function is defined as follows:

$$\nabla_F(R) := \{u \rightarrow \langle v_1, \ldots, v_n \rangle \in \mathrm{Rule}(R) \mid \text{both } P_L(u) \text{ and } P_R(u, \{v_1, \ldots, v_n\}) \text{ hold}\}$$

■

Each tactic is a specification of a proof search procedure. For instance a simplification tactic tries to reduce the size of a term according to some given term ordering; an induction tactic tries to apply an induction axiom that is suitable for a given problem; a rippling tactic tries to enable the application of the induction hypothesis; or a tactic may implement a superposition strategy [Bachmair *et al*, 1992].

Each kind of tactic or set of tactics requires specific replacement rules. We view filters as the primary infrastructure to specify the different classes of replacement rules. For example for the simplification tactics, the required replacement rules $u \rightarrow \langle v_1, \ldots, v_n \rangle$ are either those where the right-hand side is empty, i.e. $n = 0$, or that result from ε-type leaf nodes, $n = 1$ and u is greater than v_1 with respect to a well-founded term ordering. For the Rippling tactic the categorisation relies on the notion of skeletons and the relative occurrences of contexts with respect to the skeleton part. For a superposition tactic, the left-hand side must be greater than any right-hand side, again, with respect to a well-founded term ordering.

The filter infrastructure is interfaced to the tactics by having a special operator in the tactic language named filters. Furthermore, the tactic language supports the assignment of filters to windows. The effect is that for each window, the set of replacement rules are determined with respect to all filters assigned to that window. Having a special operator in the tactic language allows us to integrate the change of assignments to windows into the description of complex tactics.

The replacement of active windows by new active windows during the application of CoRe window reasoning rules requires a general principle for the inheritance of assigned filters from the original set of active windows to the new set of active windows. To define this inheritance principle, the proof datastructure is used, which contains the relationship between the old and the new active windows. If a CoRe reasoning rule justifies a proof node for some window w by a set of proof nodes for new active windows $w_1, \ldots, w_n$, then the filters assigned to w are inherited to each w_i. Take as an example the subwindow reasoning rule (cf. Definition 6.3.2, page 102) that is applied on some active window w: the rule justifies the window proof node for w by a subwindow justification to the window proof nodes for the new subwindows:

Label	WPState		Window	Justification/Abstraction/Refinement
L_0	WPS	$\vdash$	w	*Subwindow-Open*$(L_1,\ldots,L_n)$
L_1	WPS_1	$\vdash$	w_1	
$\vdots$	$\vdots$		$\vdots$	
L_n	WPS_1	$\vdash$	w_n	

Thus, each w_i inherits the assigned filters from w. Furthermore, the rule adapts the window proof state of any other active window w' by introducing an *Adapt-Window-Proof-State* to the window proof node for w' with respect to the new window proof state:

Label	WPState		Window	Justification/Abstraction/Refinement
L	WPS	$\vdash$	w	*Adapt-Window-Proof-State*(L'_1)
L'	WPS_1	$\vdash$	w	

Thus, the assignment of filters is preserved for these windows.

9.5 Tactic Execution & Hierarchical Proof Datastructure

The hierarchies in proof datastructure for CoRe support the representation of a CoRe derivation at different levels of abstractions. Tactics, especially named tactics, defined in the tactic language are a hierarchical specification of proof search methodologies. Take as an example a tactic of the form

$$\texttt{f(t}_1\texttt{, ..., t}_n\texttt{) = ...g(t}_i\texttt{) ...}$$

which is the definition of a named tactic `f` with formal parameters $\texttt{t}_1,\ldots,\texttt{t}_n$, that calls some further tactic `g` with $\texttt{t}_i$. When the tactic f is executed it involves the execution of the tactic g. Thus, that call to f is hierarchically higher than the subsequent call to g, and it is desirable to represent this fact in the proof datastructure, for example in order to allow a "post-mortem" analysis of the effectiveness of the used tactics. Note that this is also possible for recursive calls to tactics: in that case the first call to some tactic f is hierarchically higher than subsequent calls to the same tactic.

In order to represent the hierarchical dependencies between tactic calls, we need to determine the window proof nodes respectively before and after the call to some tactic f. For those initial window proof nodes that are *actively* related (cf. Definition 8.3.2) to window proof nodes N after tactic execution we introduce for N a new justification that describes that tactic call. The new justification is an abbreviation of these paths and thus it is added to the set J_A of the directed justifications of N.

As already mentioned above, the window proof states before and after the call of a tactic are saved by the tactic interpreter. Thus, defining the call to a named tactic f with actual parameters $t_1,\ldots,t_n$ we can assume that the window proof states are known, and called WPS before tactic execution and WPS' after tactic execution. Then, let $\mathcal{W}$ be the active windows in WPS and $\mathcal{W}'$ be the active windows in WPS'. For each $w \in \mathcal{W}$ whose window proof node N_w is actively justified by some window proof node with respect to WPS', let $\mathcal{N}_w$ be all window proof nodes with respect to WPS' of which N_w depends (cf. Definition 8.3.1). Then we introduce an additional abbreviating justification $J_{f(t_1,\ldots,t_n)} :=$ (Call $f,\{t_1,\ldots,t_n\}) : \mathcal{N}_w$ for the window proof node N_w.

Along the same lines abbreviating justifications can be introduced for the tacticals used in the body of named tactics. Doing so, the hierarchical proof datastructure is an explicit representation of *all* levels of abstractions contained in tactics that have contributed to the derivation. This explicit

information is not only interesting for the user, but also for the monitoring of tactic executions. The latter is especially valuable for a user in order to analyse the efficiency of the proof search procedure he specified in the tactics. Furthermore, it could serve as a basis for optimisation of tactic specifications.

9.6 Summary

In this chapter we have sketched the interface of CoRe to automatic reasoning procedures using the tactic language of [Autexier *et al*, 1999] to design reasoning procedures. We sketched the relationships between on the one hand proof development and basic tacticals, and on the other hand between proof backtracking and failure continuations in the implementation of the tactic language. In order to handle the large number of possible replacement rules, we introduced the notion of a filter for the selection of replacement rules. Filters and their assignment to windows are further primitives in the tactic language. Also we described the uniform inheritance mechanism for filters during rule applications. Finally, we showed how the implicit tactic hierarchies can be made explicit by appropriate justifications. Thereby the hierarchy of tactic calls that contributed to the proof is explicitly represented in the proof datastructure. This representation supports a better understanding of how the proof was constructed and it can also be exploited to analyse the efficiency of tactics.

Chapter 10

Sequent Calculus Style Interface

In this chapter we show how a sequent style calculus [Gentzen, 1969] can be defined on top of CoRe. The sequents required by these calculi are defined with respect to the active windows of a CoRe window proof state. Roughly, sequents are lists of active windows that are α-related, but the active windows of a window proof state must satisfy further properties, called the *sequentiality property*. The sequentiality property enforces a specific structure of the FVIF-tree of a CoRe window proof state with respect to the active windows. The preservation of that structure during the application of the sequent calculus style inference rules requires a specific β-decomposition rule that allows for the decomposition of internal β-type signed formulas, while preserving the context surrounding those formulas. Intuitively, it supports the derivation of the formula $\beta(\varphi(A),\varphi(B))$ from a formula $\varphi(\beta(A,B))$. Such a rule has been proposed by Schütte in [Schütte, 1977] and we prove in Section 10.1 the admissibility of this rule in the CoRe calculus. Based on a window version of this rule we present the formal definition of sequents on top of a CoRe window proof state and the implementation of the sequent style calculus in Section 10.2. Finally, we discuss the relationship of this calculus to the theorem proving modulo calculus [Dowek *et al*, 1998, Dowek, 2000] and provide some evidence that the sequent style calculus together with the reasoning capabilities of CoRe not only subsumes, but also extend the features of the calculus.

10.1 Schütte's β-Decomposition Rule

In this section we present the β-decomposition rule from [Schütte, 1977] and prove that it is a derived reasoning rule in CoRe. The derived reasoning rule is subsequently used in order to define the β-decomposition rule for the sequent style calculus defined in this chapter.

Exploiting the uniform notation introduced for the representation of signed formulas, the decomposition of β-type formulas in [Schütte, 1977] is defined as follows:

$$\frac{\varphi(A^{p_A})^p \qquad \varphi(B^{p_B})^p}{\varphi(\beta(A^{p_A},B^{p_B}))^p}\ \beta\text{-Decompose}$$

where φ is any higher-order predicate of type $o \to o$ and of the form $\lambda x_o \,.\, \psi$, such that x occurs exactly once with a defined polarity in ψ^p, for any $p \in \{-,+\}$. In [Schütte, 1977] this rule is restricted to situations where φ contains no β-type formulas. However, the rule is sound for the general case, and its generalised version can be defined for CoRe proof states as follows:

Definition 10.1.1 (Schütte β-Decomposition Rule) Let $[Q,\sigma \rhd_{\mathcal{L}} R(\varphi(\beta(A^{p_A},B^{p_B}))^p)]$, $p \in \{-,+\}$, be a proof state with a subtree R' of label $\varphi(\beta(A^{p_A},B^{p_B}))^p$ and φ is of the form $\lambda x_o \,.\, \psi$, such that x occurs exactly once with a defined polarity in ψ^p. Then the *Schütte β-decomposition of R'* is the proof step $[Q,\sigma \rhd_{\mathcal{L}} R(\varphi(\beta(A^{p_A},B^{p_B}))^p)] \longmapsto [Q,\sigma \rhd_{\mathcal{L}} R(\beta(\varphi(A^{p_A}),\varphi(B^{p_B}))^p)]$. ■

This β-decomposition rule is admissible in the CoRE calculus, i.e. whenever we have a proof state where in its FVIF-tree occurs a subtree of label $\varphi(\beta(A^{p_A},B^{p_B}))^p$, then we can find a sequence of CoRE reasoning steps transforming that subtree to a subtree of the form $\beta(\varphi(A^{p_A}),\varphi(B^{p_B}))^p$.

Lemma 10.1.2 (Admissibility of Schütte β-Decomposition Rule) Each application of the Schütte β-decomposition rule can be simulated by a sequence of CoRE calculus rules. ■

Proof. The higher-order predicate φ is of the form $\lambda x \,.\, \psi$, where x occurs exactly once in ψ and has a defined polarity whenever ψ has a defined polarity.

The proof of this lemma is by induction over the structure of ψ:

Base Case *$\psi := x$: thus $\varphi := \lambda x \,.\, x$ and $\varphi(\beta(A^{p_A},B^{p_B}))^p$ is $\beta(A^{p_A},B^{p_B})^p$ and the proof state is not changed by the rule.*

Induction Step: *By induction hypothesis we can assume that the conjecture holds for some ψ', resp. $\varphi' := \lambda x \,.\, \psi'$. We prove the induction step for the case where the polarity p is positive, i.e. $p := +$. The proofs for $p := -$ are analogous. The proof principle for each case is to encode the β-decomposition proof step in a sequence of CoRE proof steps: first, we perform a cut on R' over the new subformula $\beta(\varphi(A^{p_A}),\varphi(B^{p_B}))$, prove one side of the formula representing the cut by replacement rule application, and reduce the subtree to the desired "shape" by simplification and weakening.*

1. *$\psi := \neg(\psi')$: thus, $\varphi := \lambda x \,.\, \neg(\psi')$*

$$\begin{array}{ll} & [Q,\sigma \rhd_{\mathcal{L}} R(\neg(\varphi'(\beta(A^{p_A},B^{p_B})))^+)] \\ \overset{I.H.}{\longmapsto^*} & [Q,\sigma \rhd_{\mathcal{L}} R(\neg(\vee(\varphi'(A^{p_A})^-,\varphi'(B^{p_B})^-)))] \end{array}$$

At this stage we cut over the formula $\neg(\varphi'(A^{p_A})) \wedge \neg(\varphi'(B^{p_B}))$. This introduces a positive and a negative version of that formula, i.e. $\wedge(\neg(\varphi'(A^{p_A})),\neg(\varphi'(B^{p_B})))^+$ and $\wedge(\neg(\varphi'(A^{-p_A})),\neg(\varphi'(B^{-p_B})))^-$.

Thus, we obtain the proof state:

$$[Q',\sigma' \rhd_{\mathcal{L}} R\left(\begin{array}{c} \wedge(\neg(\varphi'(A^{p_A})^+),\neg(\varphi'(B^{p_B})^-))^- \vee \overbrace{\neg(\vee(\varphi'(A^{p_A})^-,\varphi'(B^{p_B})^-))^+}^{Weakening} \\ \wedge \\ \wedge(\neg(\underbrace{\varphi'(A^{-p_A})^+}_{R1}),\neg(\underbrace{\varphi'(B^{-p_B})^+}_{R2}))^- \Rightarrow \neg(\vee(\underbrace{\varphi'(A^{p_A})^-}_{A1},\underbrace{\varphi'(B^{p_B})^-}_{A2}))^+ \end{array}\right)]$$

Subsequently, we apply the unconditional replacement rules obtained from $R1$ and $R2$ respectively to the subtrees $A1$ and $A2$. This replaces these subtrees by $False^-$ which allows to prove this part of the cut-formula by simplification. The upper occurrence of the original signed formula is weakened and final simplification of the whole subtree results in the proof state:

$$[Q',\sigma' \rhd_{\mathcal{L}} R(\wedge(\neg(\varphi'(A^{p_A})),\neg(\varphi'(B^{p_B})))^+)]$$

2. $\psi := \wedge(C, \psi')$: *thus,* $\varphi := \lambda x \,.\, \wedge(C, \psi')$:

$$\begin{array}{ll} & [Q, \sigma \rhd_{\mathcal{L}} R(\wedge(C, \varphi'(\beta(A^{p_A}, B^{p_B})))^+)] \\ \overset{I.H.}{\longmapsto}{}^* & [Q, \sigma \rhd_{\mathcal{L}} R(\wedge(C, \wedge(\varphi'(A^{p_A})^+, \varphi'(B^{p_B})^+)))] \end{array}$$

At this stage we perform a cut over the formula for $\wedge(\wedge(C, \varphi'(A^{p_A})), \wedge(C, \varphi'(B^{p_B})))$. *This introduces a positive and a negative version of that formula, i.e.* $\wedge(\wedge(C, \varphi'(A^{p_A})), \wedge(C, \varphi'(B^{p_B})))^+$ *and* $\wedge(\wedge(C, \varphi'(A^{-p_A})), \wedge(C, \varphi'(B^{-p_B})))^-$

Thus, we obtain the proof state:

$$[Q', \sigma' \rhd_{\mathcal{L}} R\left(\begin{array}{c} \wedge(\wedge(C, \varphi'(A^{p_A})), \wedge(C, \varphi'(B^{p_B})))^+ \vee \overbrace{\wedge(C, \wedge(\varphi'(A^{p_A})^+, \varphi'(B^{p_B})^+))^+}^{Weakening} \\ \wedge \\ \wedge(\wedge(\underbrace{C}_{R0}, \underbrace{\varphi'(A^{-p_A})}_{R1}), \wedge(C, \underbrace{\varphi'(B^{-p_B})}_{R2}))^- \Rightarrow \wedge(\underbrace{C}_{A0}, \wedge(\underbrace{\varphi'(A^{p_A})^+}_{A1}, \underbrace{\varphi'(B^{p_B})^+}_{A2}))^+ \end{array}\right)]$$

(1) (2) (3)

Subsequently, we apply the unconditional replacement rules obtained from $R0$, $R1$ *and* $R2$ *respectively to the subtrees* $A0$, $A1$ *and* $A2$. *This replaces these subtrees by* $True^+$ *which allows to prove this part of the cut-formula by simplification. The upper occurrence of the original signed formula is weakened and final simplification of the whole subtree results in the proof state:*

$$[Q', \sigma' \rhd_{\mathcal{L}} R(\wedge(\wedge(C, \varphi'(A^{p_A})), \wedge(C, \varphi'(B^{p_B})))^+)]$$

3. $\psi := C \vee \psi'$: *thus,* $\varphi := \lambda x \,.\, \vee(C, \psi')$:

$$\begin{array}{ll} & [Q, \sigma \rhd_{\mathcal{L}} R(\vee(C, \varphi'(\beta(A^{p_A}, B^{p_B})))^+)] \\ \overset{I.H.}{\longmapsto}{}^* & [Q, \sigma \rhd_{\mathcal{L}} R(\vee(C, \wedge(\varphi'(A^{p_A})^+, \varphi'(B^{p_B})^+)))] \end{array}$$

At this stage we perform a cut over the formula for $\wedge(\vee(C, \varphi'(A^{p_A})), \vee(C, \varphi'(B^{p_B})))$. *This introduces a positive and a negative version of that formula, i.e.* $\wedge(\vee(C, \varphi'(A^{p_A})), \vee(C, \varphi'(B^{p_B})))^+$ *and* $\wedge(\vee(C, \varphi'(A^{-p_A})), \vee(C, \varphi'(B^{-p_B})))^-$

Thus, we obtain the proof state:

$$[Q', \sigma' \rhd_{\mathcal{L}} R\left(\begin{array}{c} \wedge(\vee(C, \varphi'(A^{p_A})), \vee(C, \varphi'(B^{p_B})))^+ \vee \overbrace{\vee(C, \wedge(\varphi'(A^{p_A})^+, \varphi'(B^{p_B})^+))^+}^{Weakening} \\ \wedge \\ \wedge(\vee(C, \underbrace{\varphi'(A^{-p_A})}_{R1}), \vee(C, \underbrace{\varphi'(B^{-p_B})}_{R2}))^- \Rightarrow \vee(\underbrace{C}_{A0}, \wedge(\underbrace{\varphi'(A^{p_A})^+}_{A1}, \underbrace{\varphi'(B^{p_B})^+}_{A2}))^+ \end{array}\right)]$$

(1) (2)

Subsequently, we apply the replacement rules from $R1$ and $R2$ respectively to $A1$ and $A2$, which are both replaced by $\neg(C^-)^+$. On these two new subtrees we apply the unconditional replacement rule from $A0$ to C^-, which replaces these subtrees by $False^-$ and $False^-$, which allows to prove this part of the cut-formula by simplification. The upper occurrence of the original signed formula is weakened and final simplification of the whole subtree results in the proof state:

$$[Q', \sigma' \rhd_{\mathcal{L}} R(\wedge(\vee(C, \varphi'(A^{p_A})), \vee(C, \varphi'(B^{p_B})))^+)]$$

4. *The cases for $\psi := C \Rightarrow \psi'$ and $\psi := \psi' \Rightarrow C$ can be proved analogously.*

5. *$\psi := \Diamond\psi'$, and thus $\varphi := \lambda x.\Diamond(\psi')$: note that in the FVIF-tree the subtree for $\Diamond_{Q'}\varphi'(\beta(A^{p_A}, B^{p_B}))$ has a reference to the subtree Q' of Q from which stems this modal quantifier. Thus we have:*

$$\begin{array}{ll} & [Q, \sigma \rhd_{\mathcal{L}} R(\Diamond_{Q'}\varphi'(\beta(A^{p_A}, B^{p_B})))^+)] \\ \overset{I.H.}{\longmapsto}{}^* & [Q, \sigma \rhd_{\mathcal{L}} R(\Diamond_{Q'}(\wedge(\varphi'(A^{p_A})^+, \varphi'(B^{p_B})^+)))] \end{array}$$

At this stage instead of performing a cut, we apply to $\Diamond_{Q'}(\wedge(\varphi'(A^{p_A})^+, \varphi'(B^{p_B})^+))$ the modal structural permutation rule to obtain

$$[Q, \sigma \rhd_{\mathcal{L}} R(\wedge(\Diamond_{Q'}(\varphi'(A^{p_A})), \Diamond_{Q'}(\varphi'(B^{p_B}))))]$$

which proves that case. The other case where $\psi := \Box(\psi')$ is analogous. □

We now define a window version of the Schütte β-decomposition rule.

Definition 10.1.3 (Window Schütte β-Decomposition Rule) Let $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ be a window proof state, R' a subtree of R of the form $\varphi(\beta(A^{p_A}, B^{p_B}))^p$. Let further $R'' := \beta(\varphi(A^{p_A})^p, \varphi(B^{p_B})^p)^p$ be the FVIF-tree that replaces R' in the Schütte β-decomposition rule, and $\iota_A : \mathcal{S}(R') \setminus \mathcal{S}(\beta(A^{p_A}, B^{p_B})) \rightarrow \mathcal{S}(\varphi(A^{p_A})^p) \setminus \mathcal{S}(A^{p_A})$ and $\iota_B : \mathcal{S}(R') \setminus \mathcal{S}(\beta(A^{p_A}, B^{p_B})) \rightarrow \mathcal{S}(\varphi(B^{p_B})^p) \setminus \mathcal{S}(B^{p_B})$ the obvious isomorphisms. Finally, let f_A^φ and f_B^φ be the window structures such that $f_A^\varphi \circ \iota_A = \iota_A \circ f_{\downarrow \mathcal{S}(R') \setminus \mathcal{S}(\beta(A^{p_A}, B^{p_B}))}$ and $f_B^\varphi \circ \iota_B = \iota_B \circ f_{\downarrow \mathcal{S}(R') \setminus \mathcal{S}(\beta(A^{p_A}, B^{p_B}))}$ hold. Then the *window Schütte β-decomposition rule* is defined by

$$\frac{[Q, \sigma \rhd_{\mathcal{L}} (R, f)]}{[Q', \sigma \rhd_{\mathcal{L}} (R, f)_{R' \leftarrow (R'', f'')}]} \text{ Schütte β-Decomposition}$$

where if there is an n such that $f(n) = R'$, then $f'' := f_A^\varphi \oplus f_B^\varphi \oplus f_{\downarrow A^{p_A}} \oplus f_{\downarrow B^{p_B}} \oplus \{n \rightarrow R''\}$; otherwise $f'' := f_A^\varphi \oplus f_B^\varphi \oplus f_{\downarrow A^{p_A}} \oplus f_{\downarrow B^{p_B}}$. ■

10.2 Sequents and Sequent Style Inference Rules

Let us now introduce the notion of sequents [Gentzen, 1969] on top of the CoRe window reasoning rules. We use the window reasoning capabilities of CoRe and define sequents as a list of windows, that (1) denote subtrees that are α-related, and (2) in the smallest subtree that contains all windows defining the sequent, there are no β-related subtrees to the windows. Furthermore, in order to ensure that the sequents only denote formulas and atoms, we require the windows used in sequents to not denote inner substructures (see Definition 6.2.1).

Definition 10.2.1 (Sequents) Let $WPS := [Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ be a window proof state and $w_1, \ldots, w_n$ active windows, such that $w_1, \ldots, w_i$ have negative polarity and $w_{i+1}, \ldots, w_n$ have positive polarity. Let further R' be the smallest subtree that contains all subtrees denoted by $w_1, \ldots, w_n$. Then $w_1, \ldots, w_i \vdash w_{i+1}, \ldots, w_n$ is a *sequent with respect to WPS* if, and only if,

1. all w_i do not denote inner substructures of the R (cf. Definition 6.2.1),
2. all w_i are α-related between each other, and
3. there is no subtree in R' that is β-related to any w_i.

We say that a sequent $w_1, \ldots, w_i \vdash w_{i+1}, \ldots, w_n$ is *proved* if, and only if, at least one of the w_i denotes a subtree that is proved, i.e. is either $True^+$, $False^-$, or $\zeta(s,s)$. ■

Notation 10.2.2 In the sequel, we agree to denote a sequent $w_1, \ldots, w_i \vdash w_{i+1}, \ldots, w_n$ also by simply writing the list of window $w_1, \ldots, w_i, w_{i+1}, \ldots, w_n$, since the sequent-structure is uniquely determined up to permutations of windows by the polarities of the windows. Furthermore, we may write $\varphi_1^{p_1}, \ldots, \varphi_n^{p_n}$ to denote a sequent composed of n windows, each denoting a subtree of label φ_i and polarity p_i. ■

Based on this definition of sequents we define the sequent-style calculus rules. The decomposition rules are based on the opening of subwindows, except for the β-decomposition rule, which relies on the window Schütte β-decomposition rule. Note that there are neither γ- nor δ-rules, since FVIF-trees do not contain quantified formulas (except for quantifiers without polarities).

A proof state for sequent calculus consists of a standard CoRe window proof state $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ together with a set of sequents. In order to ensure that the sequents cover all parts of the proof state, we introduce the notion of a spanning set of sequents. A set of sequents S is spanning with respect to $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ if, and only if, any literal node in R is contained in a subtree denoted by one of the windows in some sequent from S.

Definition 10.2.3 (CoRe Sequent Proof State) Let $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ be valid CoRe window proof state and S a set of sequents with respect to (R, f). Then $[Q; \sigma; (R, f) \rhd_{\mathcal{L}} S]$ is a valid CoRe *sequent proof state* if, and only if, S is spanning for R, i.e. any literal node in R is contained in a subtree denoted by one of the windows in some sequent from S. A sequent proof state is proved if, and only if, all sequents of that sequent proof state are proved. ■

In order for a set of sequents to be spanning, the active windows in the respective window proof state must also be spanning, i.e. all leaf nodes of the FVIF-tree must be contained in a subtree denoted by some active window. Thus, in order to allow for a static determination of a spanning set of sequents, we aim at the definition of an invariant, which encompasses the invariant that the active windows are always spanning.

In order to motivate the definition of that invariant assume a given window proof state with a spanning set of active windows. The sequents with respect to these windows should be uniquely determined by partitioning the active windows into sets containing an active window and all other active windows that are (1) α-related to that window and (2) all the windows in a sequent should be unconditional between each other. The second property means that none of the windows contained in a sequent has a β-related part in the (smallest) subtree containing all windows that form that sequent. Thus, we define the invariant as follows by the notion of sequential active windows:

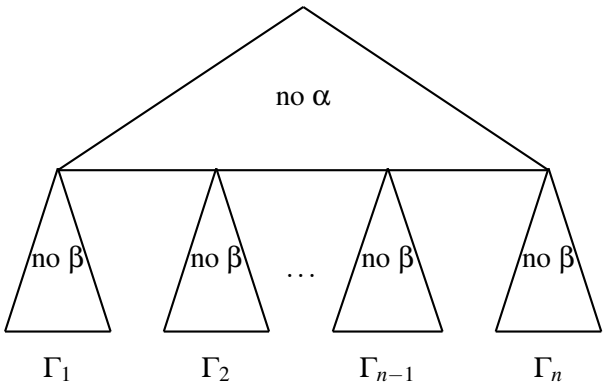


Figure 10.1: Structure of the FVIF-tree enforced by the sequentiality property.

Definition 10.2.4 (Sequential Active Windows) Let $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$ be a window proof state. The active windows from f are sequential if, and only if, (1) the active windows are spanning for R and (2) each α-related window w' of some active window w is unconditional with respect to w'. ■

The sequentiality property enforces a specific structure of the FVIF-tree, which is shown in Figure 10.1: the FVIF-tree consist of an upper part, where no α-type node occurs and a list of lower parts where no β-type node occurs above the subtrees denoted by the active windows. At the bottom of the lower parts occur the subtrees denoted by the active windows in the Γ_i. Enforcing this invariant during sequent style reasoning allows us to statically determine the sequents from the sequential active windows by partitioning these as describe above. Obviously, for each set of sequential active windows there is only one possible partition and thus the sequents are uniquely determined. In the sequel we show that this invariant holds in the initial state and that it is preserved during sequent calculus rule applications. Thus, we can state the following lemma that captures the relationship between window proof states with sequential active windows and CoRe sequent proof states.

Lemma 10.2.5 Let *WPS* be a window proof state with sequential active windows and S be the sequents obtained by partitioning these active windows. Then $[Q; \sigma; (R, f) \rhd_{\mathcal{L}} S]$ is a CoRe sequent proof state. ■

Proof. Follows directly from the definitions. □

The initial CoRe sequent proof state consists of a single sequent composed of the single initial top-level window of the window proof state. It holds trivially that this singleton active window forms a set of sequential active windows, and the only possible sequent for it is $\vdash w$. Thus, for some closed formula φ, if $[Q, \mathrm{id} \rhd_{\mathcal{L}} (R, \{w \mapsto R\})]$ is the initial window proof state for φ, then $[Q; \mathrm{id}; (R, \{w \mapsto R\}) \rhd_{\mathcal{L}} \vdash w]$ is the initial CoRe sequent proof state.

10.2.1 SK Style Axiom Rule

The axiom rule in a sequent calculus closes an open sequent of the forms: $\Gamma, A \vdash A, \Delta$ or $\Gamma \vdash s = s, \Delta$. In our representation the first sequent is not yet proved but the latter is. Thus, the axiom rule shall

"prove" sequents of the form $\Gamma, A \vdash A, \Delta$. Due to the structure of sequents, the two windows denoting the positive and negative occurrences of A are α-related between each other and there are no β-related formulas to those in the subtree containing both of them. Thus, there is an unconditonal replacement rule from A^- that can be applied on A^+ and replaces it by $True^+$ (or vice versa). The application of that rule transforms the sequent into $\Gamma, A \vdash True, \Delta$, which is a proved sequent. In order to make that explicit, the CORE simplification rule is applied on the subtree that contains the sequent $\Gamma, A \vdash True, \Delta$; it simplifies that subtree to $Proved^p$ (i.e. $True^+$ or $False^-$) and closes all the windows on that subtree. It remains a single active window on $Proved^p$ and the sequentiality of the active windows is trivially preserved.

10.2.2 SK Style Weakening Rule

The sequent calculus weakening rules are respectively[1]

$$\frac{\Gamma, \Delta}{\Gamma, A^p, \Delta} \text{ Weakening}$$

In our setting the weakened parts correspond to the subtrees in the FVIF-tree denoted by the windows A and B, respectively. According to the definition of the window weakening rule (Definition 6.3.9) we can achieve the effect of the sequent weakening rule by applying the window weakening rule to the subtree that contains the denoted window. Thereby, both the subtree and the window are removed and we end up with the new sequent $\Gamma \vdash \Delta$.

For the definition of the weakening rule we must determine a parent node of the subtree R' denoted by the window w that shall be weakened for which it holds: the parent node must have primary type α and in the subtree containing R' there occurs no other window than the window w. By the structure required by sequential active windows, this is only possible if the to-be-weakened window is not the only window defining the sequent. In other words, we can only weaken A^p in Γ, A^p, Δ, if either Γ or Δ are non-empty. This is not a serious restriction, since the weakening of A in a sequent like $A \vdash .$ results in the empty sequent $. \vdash .$, which is not provable.

Definition 10.2.6 (SK Weakening Rule) Let *WPS* be a window proof state with sequential active windows, w an active window and Γ, w, Δ a sequent with respect to *WPS*, and $\Gamma \cup \Delta \neq \emptyset$. Let further be R' the smallest parent node of the subtree R_w denoted by w that contains no other window than w and whose parent node is of type α. Then the *SK weakening rule* applies the window weakening rule on the parent of R' by weakening R'. ■

Lemma 10.2.7 (Accuracy of Definition 10.2.6) The SK weakening rule yields a window proof state, that also has sequential active windows and contains the sequent Γ, Δ, $\Gamma \cup \Delta \neq \emptyset$, instead of the sequent Γ, w, Δ. ■

Proof. Due to the structure of a sequent – ensured by the sequentiality property of active windows – if there are at least two windows in a sequent, there must be such an α-type parent node R'. The weakening of that node by replacing it with that subtree not containing the weakened window preserves the other windows and removes w. Additionally it preserves the sequentiality of the remaining active windows. Thus, in the new window proof state we have the sequent Γ, Δ. □

[1] The notation for sequents is defined in Notation 10.2.2.

10.2.3 SK Style Contraction Rule

The contraction rules in a sequent calculus are

$$\frac{\Gamma, A^p, A^p, \Delta}{\Gamma, A^p, \Delta}\ \text{Contraction}$$

The contraction formulas correspond to active windows in our setting and the actual contraction operation corresponds to the application of the window contraction rule on the respective window. By definition, that rule also creates a new window for the copied subtree. Since the contraction is performed by α-inserting the copied subtree on the original subtree, this trivially preserves the sequentiality of the active windows.

Definition 10.2.8 (SK Contraction Rule) Let *WPS* be a window proof state with sequential active windows, w an active window, and Γ, w, Δ a sequent with respect to *WPS*. The *SK contraction of* w consists in applying the window contraction rule to w which results in a new active window w' and the sequent Γ, w, w', Δ. ■

Lemma 10.2.9 (Accuracy of Definition 10.2.8) The SK contraction rule yields a window proof state that has sequential active windows and contains the sequent Γ, w, w', Δ instead of Γ, w, Δ. ■

Proof. Follows directly from the definition of the window contraction rule and the sequentiality of the original window proof state. □

10.2.4 SK Style α-Decomposition Rule

The sequent calculus rules for α-decomposition are respectively

$$\frac{\Gamma, A^{p_A}, B^{p_B}, \Delta}{\Gamma, \alpha(A^{p_A}, B^{p_B})^p, \Delta}\ \alpha\text{-Binary-Decompose} \qquad \text{and} \qquad \frac{\Gamma, A^{-p}, \Delta}{\Gamma, \neg(A^{-p})^p, \Delta}\ \alpha\text{-Unary-Decompose}$$

In our setting the $\alpha(A^{p_A}, B^{p_B})^p$ (respectively $\neg(A^{-p})^p$) is the content of an active window and the α-decomposition corresponds simply to the opening of subwindows for the subtrees A^{p_A} and B^{p_B} (respectively only for A^{-p}). For the binary case, since the direct parent node $\alpha(A^{p_A}, B^{p_B})^p$ of A^{p_A} and B^{p_B} is of primary type α, this preserves the sequentiality of the active windows. For the unary case, the sequentiality is preserved anyway. Furthermore, if the window w denoting $\alpha(A^{p_A}, B^{p_B})^p$ (resp. $\neg(A^{-p})^p$ was in a sequent Γ, w, Δ, then the subwindows w_1, w_2 for A^{p_A} and B^{p_B} (respectively the subwindow w' for A^{-p}) occur in the sequent Γ, w_1, w_2, Δ (respectively in Γ, w', Δ).

Definition 10.2.10 (SK α-Decomposition Rule) Let *WPS* be a window proof state with sequential active windows, and w an α-type window that occurs in a sequent Γ, w, Δ. The *SK* α*-decomposition rule* consists of opening subwindows for the direct children of w. ■

Lemma 10.2.11 (Accuracy of Definition 10.2.10) The SK α-decomposition rule yields a window proof state, that also has sequential active windows and contains instead of the sequent $\Gamma, \alpha(A^{p_A}, B^{p_B})^p, \Delta$ (respectively $\Gamma, \alpha(A^{p_A})^p, \Delta$) the sequent $\Gamma, A^{p_A}, B^{p_B}, \Delta$ (respectively Γ, A^{p_A}, Δ). ■

Proof. Directly from the definition and the sequentiality of the active windows of the window proof state before rule application. □

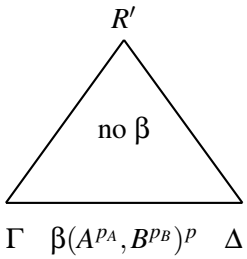


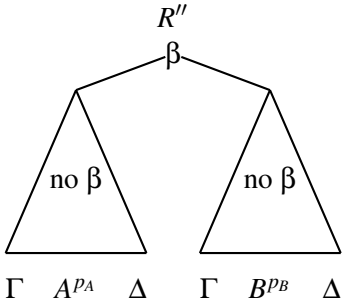
Figure 10.2: Situation before β-decomposition: the smallest subtree R' that contains all windows forming the sequent has no β-type node.

10.2.5 SK Style β-Decomposition Rule

The sequent calculus rule for β-decomposition is

$$\frac{\Gamma, A^{p_A}, \Delta \qquad \Gamma, B^{p_B}, \Delta}{\Gamma, \beta(A^{p_A}, B^{p_B})^p, \Delta}\ \beta\text{-Decompose}$$

Again, in our setting $\beta(A^{p_A}, B^{p_B})^p$ is the content of an active window. However, simply opening subwindows on the two child subtrees A^{p_A} and B^{p_B} like in the α-decomposition case would result in a window proof state where the active windows are not sequential. The situation is viewed in Figure 10.2: the sequentiality property ensures that there is a smallest subtree R' containing the windows of the sequent with $\beta(A^{p_A}, B^{p_B})^p$ and that contains no β-type node. After focusing it would contain the β-type node $\beta(A^{p_A}, B^{p_B})^p$, which violates the sequentiality condition. In order to remedy that situation the β-connective must be moved above R', which his achieved by applying the window Schütte β-decomposition rule. This rule transforms the subtree R' into



Note that the rule also "copies" all the windows in the appropriate manner in order to obtain the two new sequents Γ, A^{p_A}, Δ and Γ, B^{p_B}, Δ.

Definition 10.2.12 (SK β-Decomposition Rule) Let *WPS* be a window proof state with sequential active windows, $\beta(A^{p_A}, B^{p_B})^p$ a β-type active window that occurs in some sequent $\Gamma, \beta(A^{p_A}, B^{p_B})^p, \Delta$ and R' the smallest subtree that contains all windows of that sequent. The *SK* β-*decomposition rule* consists of applying the window Schütte β-decomposition rule to that window with respect to R'. It results in a new window proof state that contains the sequents Γ, A^{p_A}, Δ and Γ, B^{p_B}, Δ instead of the sequent $\Gamma, \beta(A^{p_A}, B^{p_B})^p, \Delta$. ■

Lemma 10.2.13 (Accuracy of Definition 10.2.12) The SK β-decomposition rule results in a proof state, that also has the sequentiality property and contains the sequents Γ, A^{p_A}, Δ and Γ, B^{p_B}, Δ instead of the sequent $\Gamma, \beta(A^{p_A}, B^{p_B})^p, \Delta$. ■

Proof. Directly from the sequentiality of the original proof state and the definition of the window Schütte β-decomposition rule. □

10.2.6 SK Style ν- and π-Decomposition Rules

The sequent calculus ν- and π-decomposition rules are

$$\frac{\Gamma, A^p \Delta}{\Gamma, \nu(A^p)^p, \Delta}\ \nu\text{-Decomposition} \quad \text{and} \quad \frac{\Gamma', A^p, \Delta'}{\Gamma, \pi(A^p)^p, \Delta}\ \pi\text{-Decomposition}$$

where $\Gamma' := \{\nu A \mid \nu A \in \Gamma\}$ and $\Delta' := \{\nu A \mid \nu A \in \Delta\}$. In our setting, both rules are realised simply by opening a subwindow for the respective subtree A^p. However, for the π-rule, instead of obtaining the sequent Γ', A^p, Δ' we obtain the sequent Γ, A^p, Δ, like in the ν case. Although this may appear strange, it is perfectly sound, since the applicability of the axiom rule checks the prefixes of the formulas, and those formulas that should have been removed by the π-rule are simply not applicable. Furthermore, mimicking the π-rule in that way offers more flexibility during proof search, which is illustrated by the following example: assume we are given the following modal sequent $(\Box A \wedge \Box B) \vdash \Box(A)$. The application of the standard π-decomposition rule on $\Box(A)$ would leave us with the non-provable sequent $\vdash \Box(A)$, which is due to the purely syntactical determination of Γ' and Δ'. In order to avoid that, we should have first applied the α-decomposition rule to obtain $\Box A, \Box B \vdash \Box(A)$, and only then apply the π-decomposition rule to obtain the "right" sequent $\Box A, \Box B \vdash A$. This necessary detour is avoided by the way we realised the π-decomposition rule, since the π-decomposition on $(\Box A \wedge \Box B) \vdash \Box(A)$ results in $(\Box A \wedge \Box B) \vdash A$, which is the provable sequent.

Definition 10.2.14 (SK ν- and π-Decomposition Rules) The SK π- and ν-decomposition rules consist of opening a subwindow on the subtree of the active π-type (respectively ν-type) window. ■

10.2.7 SK Style Instantiation

The CoRe instantiation rule applies an admissible (combined) substitution (σ_Q, σ_M) on the whole FVIF-tree and does not affect the structure of the window tree, since there are no windows below the literal level. Thus, β-type nodes can only be created by instantiating higher-order set variables, which are all inside the active windows. This ensures that no β-nodes are inserted above the active windows. The modal part σ_M of a substitution is applied on the indexed formula tree and affects the prefixes. The object variable substitution σ_Q is both applied on the indexed formula tree and the FVIF-tree, and thereby instantiates the content of the windows. Thus, the sequentiality property of the active windows is preserved during instantiation and all sequents are instantiated in parallel and the new window proof state contains the sequents $\sigma_Q(\Gamma)$, where Γ was a sequent in the original window proof state.

Definition 10.2.15 (SK Instantiation) The SK Instantiation rule consists in the application of a (combined) substitution (σ_Q, σ_M) on the window proof state.

10.2.8 SK Style Increase of Multiplicities

Multiplicities are increased in order to preserve uninstantiated formulas. From a standard sequent calculus perspective it corresponds to an a posteriori insertion in the actual sequent calculus derivation of a γ-quantifier elimination rule and subsequent application of the analogous sequent decomposition rules. The a posteriori γ-quantor elimination corresponds to the CoRe rule to increase multiplicities, and more specifically to the change in the indexed formula tree Q of a proof state $[Q, \sigma \rhd_{\mathcal{L}} (R, f)]$. The effects of that rule on the FVIF-tree R together with the copying of respective windows from f reflects the a posteriori introduction of the sequent decomposition on that new copy, analogously to the other formulas resulting from that same main γ-type formula.

From a standard sequent calculus perspective this is not the application of a sequent calculus rule, but rather a proof transformation rule. However, the availability of that kind of rule also increases the amenities of the sequent style calculus for interactive theorem proving: neither the number of γ-decomposition rules must be guessed by the time that quantor is eliminated, nor must a quantified formula be copied before instantiation, but the multiplicities can be ajusted by the needs arising during further proof search.

Example 10.2.16 We illustrate the benefit of the dynamic increase of multiplicities with the following example: consider two axioms about addition (+) over the natural numbers:

$$\forall n, m_{Nat} \,.\, \mathrm{s}(n) + m = \mathrm{s}(n+m) \tag{10.1}$$

$$\forall n, m_{Nat} \,.\, n + \mathrm{s}(m) = \mathrm{s}(n+m) \tag{10.2}$$

where s denotes the successor function on natural number. The sequent style representation for these two axioms is

$$\mathrm{s}(N_1) + M_1 = \mathrm{s}(N_1 + M_1), N_2 + \mathrm{s}(M_2) = \mathrm{s}(N_2 + M_2) \vdash$$

where the N_i and M_i are the meta-variables introduced for the respective bound variables. From (10.2) in the sequent we obtain the replacement rule $N_2 + \mathrm{s}(M_2) \rightarrow \langle \mathrm{s}(N_2 + M_2) \rangle$, which can be applied by unification (substitution is $\{N_1/N_2, \mathrm{s}(M_2)/M_1\}$) to $N_1 + M_1$ to obtain:

$$\mathrm{s}(N_1) + \mathrm{s}(M_2) = \mathrm{s}(\mathrm{s}(N_1 + M_2)), N_1 + \mathrm{s}(M_2) = \mathrm{s}(N_1 + M_2) \vdash$$

The meta-variables in the resulting equation $\mathrm{s}(N_1) + \mathrm{s}(M_2) = \mathrm{s}(\mathrm{s}(N_1 + M_2))$ are those of $\mathrm{s}(N_1) + M_1 = \mathrm{s}(N_1 + M_1)$ and $N_2 + \mathrm{s}(M_2) = \mathrm{s}(N_2 + M_2)$. The increase of multiplicities of the binding nodes of these meta-variables allows us to generate arbitrary many copies of the *derived* equation. In order to have the same convenience in a standard sequent calculus we would have to introduce the formula $\forall n, m_{Nat} \,.\, \mathrm{s}(n) + \mathrm{s}(m) = \mathrm{s}(\mathrm{s}(n+m))$ as a lemma by cut. This requires either to anticipate the result of the equation application to generate the lemma, or if we devise a derived formula to be used like a lemma, then we have to transform the proof by introducing it as a lemma at the appropriate place in order to avoid to have to reprove it again. The increase of multiplicities overcomes these problems and allows us to simply continue the proof with any derived formula.

The sequentiality property of the active windows is preserved during the increase of multiplicities. To see this, consider the structure of the FVIF-tree viewed in Figure 10.1 on page 156. This structure is enforced by the sequentiality property of the active windows. The increase of multiplicities copies independent subtrees of the whole subtree, where the copying also renames the references into the indexed formula tree appropriately. If a subtree rooted in the upper part is copied, the sequentiality is preserved, but additional sequents arise from it. If a subtree rooted in some of the lower parts is copied, for instance in the β-subtree containing the sequent Γ_2, then the sequentiality is also preserved and new windows are added to the sequent Γ_2.

10.2.9 SK Style Leibniz' Equality Introduction

The Leibniz' Equality introduction rule in sequent calculus is simply the expansion of the definition of equality. Thus, the sequent calculus rules for it are

$$\frac{\Gamma,(\forall P\,.\,P(A)\Rightarrow P(B))^{-},\Delta}{\Gamma,\varepsilon(A,B),\Delta}\ \text{Leibniz' Equality} \quad \text{and} \quad \frac{\Gamma,(\forall P\,.\,P(A)\Rightarrow P(B))^{+},\Delta}{\Gamma,\zeta(A,B),\Delta}\ \text{Leibniz' Equality}$$

The respective CORE reasoning rule α-inserts the new subtree for $P(A)\Rightarrow P(B)$ on the subtree $\overset{\xi}{\zeta}(A,B)$. If an active window is on $\overset{\xi}{\zeta}(A,B)$, then the window version of the CORE rule introduces a new active window for the new subtree. This obviously preserves the sequentiality of the active windows, and the new sequent obtained from $\Gamma,\overset{\xi}{\zeta}(A,B)^{p},\Delta$ is then $\Gamma,\overset{\xi}{\zeta}(A,B)^{p},(P(A)\Rightarrow P(B))^{p},\Delta$.

10.2.10 SK Style Extensionality Introduction

The extensionality rule in CORE corresponds to the ξ-rule[2]. The sequent calculus rules for the ξ-rule are:

$$\frac{\Gamma,\varepsilon(\lambda x\,.\,G,\lambda x\,.\,H),\Delta}{\Gamma,\gamma x\,.\,\varepsilon(G,H),\Delta}\ \text{Ext. Introduction} \quad \text{and} \quad \frac{\Gamma,\zeta(\lambda x\,.\,G,\lambda x\,.\,H),\Delta}{\Gamma,\delta x\,.\,\zeta(G,H),\Delta}\ \text{Ext. Introduction}$$

Analogously to the Leibniz' equality introduction rule, the CORE extensionality introduction rule α-inserts the subtree for $\overset{\xi}{\zeta}(\lambda x\,.\,G,\lambda x\,.\,H)^{p}$ on the original equality $\overset{\xi}{\zeta}(\lambda x\,.\,G,\lambda x\,.\,H)$. Note that the FVIF-tree does not contain the quantifiers. This syntactic requirement in the sequent calculus is subsumed by the locality property of the variable x, which is checked on the indexed formula tree and is more flexible since it does not require the quantifier to be immediately in front of $\overset{\xi}{\zeta}(G,H)$. Again, if an active window is on $\overset{\xi}{\zeta}(G,H)$, then a new window is introduced for $\overset{\xi}{\zeta}(\lambda x\,.\,G,\lambda x\,.\,H)$ and the sequentiality property of the active windows is preserved. The application of the CORE window extensionality introduction rule transforms the sequent $\Gamma,\overset{\xi}{\zeta}(G,H),\Delta$ into $\Gamma,\overset{\xi}{\zeta}(G,H),\overset{\xi}{\zeta}(\lambda x\,.\,G,\lambda x\,.\,H),\Delta$.

10.2.11 SK Style ζ-Expansion Rule

The sequent calculus ζ-expansion rule is

$$\frac{\Gamma,(A\Rightarrow B)^{+},\Delta \qquad \Gamma,(B\Rightarrow A)^{+},\Delta}{\Gamma,\zeta(A,B),\Delta}\ \zeta\text{-Expansion}$$

The ζ-expansion rule for booleans α-inserts on a subtree of label $\zeta(A_{\mathrm{o}},B_{\mathrm{o}})$ a subtree for the signed formula $((A\Rightarrow B)\wedge(B\Rightarrow B))^{+}$. The window version of that rule adds a new active window for $((A\Rightarrow B)\wedge(B\Rightarrow A))^{+}$, if there was an active window on $\zeta(A_{\mathrm{o}},B_{\mathrm{o}})$. Thus, the application of that CORE window ζ-expansion rule preserves the sequentiality of the active windows. Its application on the sequent $\Gamma,\zeta(A_{\mathrm{o}},B_{\mathrm{o}}),\Delta$ results in the sequent

[2] Due to the $\beta\eta$ long normal form used in the term representation, this is equivalent to the f-rule, as shown in [Benzmüller *et al*, 2002a].

$$\Gamma, \zeta(A_o, B_o), ((A \Rightarrow B) \wedge (B \Rightarrow A))^+, \Delta$$

From that sequent we can obtain the two sequents required by the above sequent calculus ζ-expansion rule by first weakening the subtree containing $\zeta(A,b)$ and secondly applying the window Schütte β-decomposition rule to $((A \Rightarrow B) \wedge (B \Rightarrow A))^+$. As already shown this preserves the sequentiality of the active windows and results in the two sequents

$$\Gamma, (A \Rightarrow B)^+, \Delta \quad \text{and} \quad \Gamma, (B \Rightarrow A)^+, \Delta$$

10.2.12 SK Style Cut rule

The sequent calculus Cut rule is

$$\frac{\Gamma, A^+, \Delta \qquad \Gamma, A^-, \Delta}{\Gamma, \Delta} \text{ SK Cut}$$

In order to implement that rule, we use the CoRE window cut rule, which must be applied in a way that the sequentiality of active windows is preserved. To this end we apply the window cut rule to the subtree that contains all active windows forming the sequent Γ, Δ. Assume that subtree has the label φ^p; then that subtree is replaced by a subtree for $\beta(\alpha(A^+, \varphi^p), \alpha(A^-, \varphi^p))$. Since φ^p contains no β-type node, so do $\alpha(A^+, \varphi^p)$ and $\alpha(A^-, \varphi^p)$. Furthermore, the window structure inside the original φ^p is copied for both new occurrences of φ^p and new active windows for A^- and A^+ are inserted. Thus, the sequentiality of the active windows is preserved and we have obtained the new sequents Γ, A^+, Δ and Γ, A^-, Δ.

This completes the sequent calculus style reasoning based on CoRE window calculus. Except for the instantiation and the multiplicity increasing rules all rules can be written as sequent calculus rule, as shown in Figure 10.3. The instantiation and the multiplicity increasing rules are proof transformation rules and cannot be written as pure sequent calculus rules.

In any case the CoRE reasoning rules are still available during sequent style reasoning. Especially the application of replacement rules inside sequents is naturally supported. This is commented in more detail in Section 10.3. Finally, we have restricted the window reasoning capabilities of CoRE by forbidding the opening of subwindows below the literal level. However, it is not a surprise that if we weaken this restriction we obtain the window inference capabilities defined in [Staples, 1995] for the Isabelle system.

10.3 A Note on Deduction Modulo

Theorem proving modulo [Dowek *et al*, 1998, Dowek, 2000] is a technique to integrate deduction with respect to some standard calculus like for example sequent calculus and term rewriting systems. It extends the calculus rules by an equivalence relation provided by a background theory to check the equality of formulas and terms during the application of the calculus rules. Take as an example a standard sequent calculus β-decomposition rule like

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash (A \wedge B), \Delta} \wedge\text{-R}$$

$$\frac{\textit{Proved}}{\Gamma, A^+, A^-, \Delta}\ \text{Axiom} \qquad \frac{\textit{Proved}}{\Gamma, \zeta(s,s), \Delta}\ \text{Axiom}$$

$$\frac{\Gamma, \Delta}{\Gamma, A^p, \Delta}\ \text{Weakening} \qquad \frac{\Gamma, A^p, A^p, \Delta}{\Gamma, A^p, \Delta}\ \text{Contraction}$$

$$\frac{\Gamma, A^{p_A}, \Delta \qquad \Gamma, B^{p_B}, \Delta}{\Gamma, \beta(A^{p_A}, B^{p_B})^p, \Delta}\ \beta\text{-Decompose} \qquad \frac{\Gamma, A^{p_A}, B^{p_B}, \Delta}{\Gamma, \alpha(A^{p_A}, B^{p_B})^p, \Delta}\ \alpha\text{-binary-Decompose}$$

$$\frac{\Gamma, A^{-p}, \Delta}{\Gamma, \alpha(A^{-p})^p, \Delta}\ \alpha\text{-unary-Decompose} \qquad \frac{\Gamma, A^p, \Delta}{\Gamma, (\nu A)^p, \Delta}\ \nu\text{-Decompose}$$

$$\frac{\Gamma, A^p, \Delta}{\Gamma, (\pi A)^p, \Delta}\ \pi\text{-Decompose} \qquad \frac{\Gamma, \overset{\xi}{\underset{\zeta}{}}\,(A,B)^p, (P(A) \Rightarrow P(B))^p, \Delta}{\Gamma, \overset{\xi}{\underset{\zeta}{}}\,(A,B)^p, \Delta}\ \text{Leibniz}$$

where P is a new free variable, if p is negative; otherwise P is a new Eigenvariable.

$$\frac{\Gamma, \overset{\xi}{\underset{\zeta}{}}\,(A,B)^p, \overset{\xi}{\underset{\zeta}{}}\,(\lambda x.A, \lambda x.B)^p, \Delta}{\Gamma, \overset{\xi}{\underset{\zeta}{}}\,(A,B)^p, \Delta}\ \text{Extensionality} \qquad \frac{\Gamma, (A \Rightarrow B)^+, \Delta \quad \Gamma, (B \Rightarrow A)^+, \Delta}{\Gamma, \zeta(A,B), \Delta}\ \zeta\text{-Expansion}$$

if x is local for $\overset{\xi}{\underset{\zeta}{}}\,(A,B)^p$

$$\frac{\Gamma, A^+, \Delta \qquad \Gamma, A^-, \Delta}{\Gamma, \Delta}\ \text{Cut}$$

Figure 10.3: Sequent calculus style reasoning rules supported by CoRe.

Assume further that $\mathcal{R}$ denotes boolean rewrite rules and $\mathcal{E}$ a term rewriting system, both obtained from a background theory $\mathcal{T}$. The extension of the above calculus rule that takes the theory $\mathcal{T}$ into account is then

$$\frac{\Gamma \vdash_{\mathcal{RE}} A, \Delta \quad \Gamma \vdash_{\mathcal{RE}} B, \Delta}{\Gamma \vdash_{\mathcal{RE}} C, \Delta}\ \wedge\text{-R} \qquad \text{if } C \equiv_{\mathcal{RE}} (A \wedge B)$$

where $\equiv_{\mathcal{RE}}$ is the equivalence relation implemented by the rewrite rules from $\mathcal{R}$ and $\mathcal{E}$. The integration principle thereby relies on the use of standard term rewriting techniques for $\mathcal{E}$ and an extended narrowing and resolution principle for the boolean rewrite rules in $\mathcal{R}$. However, the whole approach is restricted to unconditional rewrite rules, both for $\mathcal{E}$ and $\mathcal{R}$. The main result is the completeness result of deduction modulo expressed by

$$\Gamma \vdash_{\mathcal{RE}} A \Leftrightarrow \mathcal{T}, \Gamma \vdash A$$

The sequent style calculus on top of CoRe is an extension of the theorem proving modulo approach, as we shall see now. The key observation is that the set of rules $\mathcal{E}$ and $\mathcal{R}$ obtained from $\mathcal{T}$

are a subset of the CoRe replacement rules that result from the theory $\mathcal{T}$. Furthermore, the notion of replacement rules is not restricted to unconditional rules as in theorem proving modulo. In theorem proving modulo only boolean rewrite rules of the form $A^p \rightarrow \langle B^p \rangle$ resulting from equivalence and refinement relations are considered, as for example in the sequent $B \Rightarrow A, \Gamma \vdash \Delta$. It is not possible to use the rule $A^+ \rightarrow \langle B^+, C^+ \rangle$ in the sequent $C \Rightarrow (B \Rightarrow A), \Gamma \vdash \Delta$. Furthermore, for the rewrite rules on terms, only unconditional equations can be used, as for example in $s = t, \Gamma \vdash \Delta$. The notion of replacement rules provided by CoRe allow further the use of conditional rules like $s^\circ \rightarrow \langle t^\circ, A^+ \rangle$ in the sequent $A \Rightarrow s = t, \Gamma \vdash$. Finally, the theorem proving modulo approach provided by CoRe is applicable to all logics considered in CoRe, namely the modal logics, but also to higher-order logic with extensionality.

10.4 Summary

The sequent style calculus relies on a β-decomposition rule that is a generalisation of the β-decomposition defined by Schütte in [Schütte, 1977] and the sequentiality property of active windows in a CoRe window proof state. The sequents are lists of active windows that are α-related, and the sequentiality property ensures the accuracy of these sequent definitions. Sequentiality is preserved by using a version of the β-decomposition rule from [Schütte, 1977], which is admissible in the CoRe calculus. The infrastructure underlying the sequent style calculus and provided by the CoRe calculus supports, from a sequent calculus perspective, complex proof transformation steps (cf. Example 10.2.16) that deal with the a posteriori increase of multiplicities of γ- and ν-type formulas. As a result, we obtain a sequent style calculus that overcomes the well-known problems with fixing the order to eliminate quantifiers during proof construction in standard formulations of this calculus. Further optimisations that result from the underlying CoRe framework are the support for window inference reasoning style from [Robinson & Staples, 1993, Staples, 1995] and a built-in support of theorem proving modulo [Dowek *et al*, 1998, Dowek, 2000] due to the CoRe replacement rule application rules.

The structure of a sequent calculus proof is represented inside a signed formula by using windows in the appropriate manner. Thus it supports a *proofs as formulas* paradigm where the proof state can always be viewed as a single formula which represents the sequent calculus proof structure. Furthermore, the windows used to implement sequents can be closed at any stage of the derivation with the effect to invert the sequent decomposition rules, without actually having to explicitly reconstruct the formulas.

Chapter 11

Sample Proofs in CoRe

In this chapter we present now some examples of CoRe proofs using the proof representation from Chapter 8. From the hierarchical proof datastructure we can generate an elementary natural language presentation of a proof. The presentation relies on the idea to group together α-related windows to form one case of a proof, where the negative windows are the assumptions of that case and the positive windows the disjunctive goals. The reasoning rules are the focusing rules and the CoRe calculus rules. Thereby focusing below a β-type connective introduces new cases, while focusing below α-type connectives introduces new assumptions or alternative goals. The basic language constructs for the proof presentation language are:

- "*Case*" to describe the case of a case analysis;
- "*Assume*" to list the assumptions of a single case;
- "*by rule*" to describe the application of a CoRe calculus rule.

Using these constructs allows us to present the sample proofs in a natural language style. Note that although the language is not formally defined, it extends the language introduced in [Abel *et al*, 2001] for first-order logic assertion level proofs.

11.1 Higher-Order Logic Proofs

In this section we present the proof for the higher-order logic theorems $p_{o\to o}(a_o \wedge b_o) \Rightarrow p(b \wedge a)$, and $\forall p_{o\to o}.\lambda x.p(p(p(x))) = \lambda x.p(x)$. For the presentation of the proofs, especially for the formulas in the FVIF-trees, we agree to denote γ-type variables in capital letter, as for example X, and δ-type variables by lower case letter, for example y.

11.1.1 Proof of $\mathbf{p}_{o\to o}(\mathbf{a}_o \wedge \mathbf{b}_o) \Rightarrow \mathbf{p}(\mathbf{b} \wedge \mathbf{a})$

Example Proof 11.1.1 of $\mathbf{p}_{o\to o}(\mathbf{a}_o \wedge \mathbf{b}_o) \Rightarrow \mathbf{p}(\mathbf{b} \wedge \mathbf{a})$
We have to prove $p_{o\to o}(a_o \wedge b_o) \Rightarrow p(b \wedge a)$.
By cut over the lemma $a \wedge b = b \wedge a$ we obtain the following cases:

Case 1: *We have to prove the disjunctive goals (1.1) $a \wedge b = b \wedge a$ and (1.2) $p_{o\to o}(a_o \wedge b_o) \Rightarrow p(b \wedge a)$.*

By weakening with discard the goal (1.2).

By ζ-expansion to (1.1) we reduce it to the new goal

$$((a \wedge b) \Rightarrow (b \wedge a)) \wedge ((b \wedge a) \Rightarrow (a \wedge b)) \tag{11.1}$$

In that formula we apply $a^- \rightarrow \langle True^+ \rangle$ and $b^- \rightarrow \langle True^+ \rangle$ to the left occurrence of the subformula $(b \wedge a)$, and similarly for the second occurrence of the subformula $(a \wedge b)$, to obtain

$$((a \wedge b) \Rightarrow (True \wedge True)) \wedge ((b \wedge a) \Rightarrow (True \wedge True)) \tag{11.2}$$

This goal is trivially simplified to the goal True, which completes the proof of this case.

Case 2: *Assume $a \wedge b = b \wedge a$, the goal is to prove $p_{o \rightarrow o}(a_o \wedge b_o) \Rightarrow p(b \wedge a)$.*

By the assumption $a \wedge b = b \wedge a$ the goal is reduced to $p_{o \rightarrow o}(a_o \wedge b_o) \Rightarrow p(a \wedge b)$.

From the left hand side of the implication we obtain the replacement rule $p(a \wedge b)^- \rightarrow \langle True^+ \rangle$ whose application on the right hand side leaves us with $p_{o \rightarrow o}(a_o \wedge b_o) \Rightarrow True$.

This goal is trivially simplified to True, which proves that case. □

The detailed version of that proof using the hierarchical proof datastructure from Chapter 8 is given in Appendix B.1.

CoRe Sequent Calculus Proof. We now present a proof of that theorem in the CoRe sequent calculus style interface by mixing it with the underlying CoRe calculus rules inside the windows that form the sequents. On the initial sequent $\vdash p(a \wedge b) \Rightarrow p(b \wedge a)$ we perform a cut over $a \wedge b = b \wedge a$, which results in the two sequents:

$$\vdash a \wedge b = b \wedge a, p(a \wedge b) \Rightarrow p(b \wedge a) \quad \text{and} \quad a \wedge b = b \wedge a \vdash p(a \wedge b) \Rightarrow p(b \wedge a)$$

The first sequent is proved as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\vdash True}\ \text{Axiom}}{\vdash (a \wedge b) \Rightarrow (True \wedge True)}\ \text{Simplify}}{\vdash (a \wedge b) \Rightarrow (b \wedge a)}\ \text{Apply}\ a^- \rightarrow \langle True^+ \rangle,\ b^- \rightarrow \langle True^+ \rangle \qquad \dfrac{\dfrac{\dfrac{-}{\vdash True}\ \text{Axiom}}{\vdash (a \wedge b) \Rightarrow (True \wedge True)}\ \text{Simplify}}{\vdash (b \wedge a) \Rightarrow (a \wedge b)}\ \text{Apply}\ a^- \rightarrow \langle True^+ \rangle,\ b^- \rightarrow \langle True^+ \rangle}{\vdash ((a \wedge b) \Rightarrow (b \wedge a)) \wedge ((b \wedge a) \Rightarrow (a \wedge b))}\ \beta\text{-Decomposition}}{\vdash a \wedge b = b \wedge a}\ \zeta\text{-Expansion}}{\vdash a \wedge b = b \wedge a, p(a \wedge b) \Rightarrow p(b \wedge a)}\ \text{Weakening}$$

The second sequent is proved as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{a \wedge b = b \wedge a \vdash True}\ \text{Axiom}}{a \wedge b = b \wedge a \vdash p(a \wedge b) \Rightarrow True}\ \text{Simplify}}{a \wedge b = b \wedge a \vdash p(a \wedge b) \Rightarrow p(a \wedge b)}\ \text{Apply}\ p(a \wedge b)^- \rightarrow \langle True^+ \rangle}{a \wedge b = b \wedge a \vdash p(a \wedge b) \Rightarrow p(b \wedge a)}\ \text{Apply}\ a \wedge b \rightarrow \langle b \wedge a \rangle$$

11.1.2 Proof of $\forall \mathbf{p}_{o\to o} . \lambda \mathbf{x} . \mathbf{p}(\mathbf{p}(\mathbf{p}(\mathbf{x}))) = \lambda \mathbf{x} . \mathbf{p}(\mathbf{x})$

CoRe Window Proof. For the proof of that theorem we use the following lemmata, which can be easily proven. The first is a simple lemma which states that if a formula is not true, then it is false. The second lemma is an instance of a boolean extensionality property.

$$\forall x_o . \neg(x = True) \quad \Rightarrow \quad x = False \tag{11.3}$$

$$\forall Q, P_{o\to o} . (\forall x_o . Q(x) \Rightarrow P(x)) \quad \Rightarrow \quad \lambda x . Q(x) = \lambda x . P(x) \tag{11.4}$$

The proof of the theorem $\forall p_{o\to o} . \lambda x . p(p(p(x))) = \lambda x . p(x)$ is performed by case analysis over $x = True$, and the values of $p(True)$ and $p(False)$.

Example Proof 11.1.2 $\forall \mathbf{p}_{o\to o} . \lambda \mathbf{x} . \mathbf{p}(\mathbf{p}(\mathbf{p}(\mathbf{x}))) = \lambda \mathbf{x} . \mathbf{p}(\mathbf{x})$
By lemma (11.4) to $\lambda x . p(p(p(x))) = \lambda x . p(x)$ we obtain the new goal

$$p(p(p(x))) = p(x) \tag{11.5}$$

By cut over $x = True$ we obtain two cases:

Case 1: *Assuming $x = True$ we have to prove $p(p(p(x))) = p(x)$.*

By the assumption $x = True$ (i.e. the replacement rule $x \to \langle True \rangle$) the goal is reduced to

$$p(p(p(True))) = p(True). \tag{11.6}$$

By cut over $p(True) = True$ we obtain two cases:

Case 1.a: *Assuming $p(True) = True$ we have to prove $p(p(p(True))) = p(True)$.*
Applying the assumption four times reduces it to the trivially provable goal $True = True$, which completes the proof of this case.

Case 1.b: *Assuming $\neg(p(True) = True)$ we have to prove the goal $p(p(p(True))) = p(True)$.*
By lemma (11.3) on the assumption we obtain

$$p(True) = False. \tag{11.7}$$

Applying (11.7) (i.e. the replacement rule $p(True) \to \langle False \rangle$) twice reduces the goal to

$$p(p(False)) = False. \tag{11.8}$$

By cut over $p(False) = True$ we obtain two cases:

Case 1.b.i: *Assuming $p(False) = True$, we have to prove $p(p(False)) = False$.*
By the assumptions $p(False) = True$, $p(True) = False$ this goal is reduced to $False = False$, which is trivially provable and thus completes the proof of that case.

Case 1.b.ii: *Assuming $\neg(p(False) = True)$, we have to prove $p(p(False)) = False$.*
By lemma (11.3) on the assumption we can derive $p(False) = False$.
Applying $p(False) = False$ twice to the goal leaves us with the trivial goal $False = False$.

Case 2: *Assuming $\neg(x = True)$ we have to prove $p(p(p(x))) = p(x)$.*

The proof of that case is similar than for the first case using lemma (11.3) and is conducted by case analysis over $p(False) = True$ and $p(True) = True$. □

The detailed version of that proof using the hierarchical proof datastructure from Chapter 8 is given in Appendix B.2.

CoRe Sequent Calculus Proof. The proof for the same theorem with the CoRe sequent calculus style interface of CoRe from Chapter 10 is performed as follows: on the initial sequent[1] $\vdash \lambda x\,.\,p(p(p(x))) = \lambda x\,.\,p(x)$ we first apply the lemma (11.4) to obtain the sequent

$$\vdash p(p(p(x))) = p(x)$$

By cut over $x = \mathit{True}$ we obtain the two sequents

$$x = \mathit{True} \;\vdash\; p(p(p(x))) = p(x) \tag{11.9}$$
$$\vdash\; x = \mathit{True}, p(p(p(x))) = p(x) \tag{11.10}$$

We only show the proof for (11.9); the proof for (11.10) is analogous.

On (11.9) we proceed by case analysis over $p(\mathit{True}) = \mathit{True}$, which results in

$$x = \mathit{True}, p(\mathit{True}) = \mathit{True} \;\vdash\; p(p(p(x))) = p(x) \tag{11.11}$$
$$x = \mathit{True} \;\vdash\; p(\mathit{True}) = \mathit{True}, p(p(p(x))) = p(x) \tag{11.12}$$

To (11.11) we apply the rewriting replacements rules obtained from the formulas in the antecedent of the sequent to reduce that sequent to $x = \mathit{True}, p(\mathit{True}) = \mathit{True} \vdash \mathit{True}$, which is trivially provable by the axiom rule.

To (11.12) we apply the resolution replacement rule from (11.3) at the positive formula $p(\mathit{True}) = \mathit{True}$ to obtain, after $\neg_R$-elimination,

$$x = \mathit{True}, p(\mathit{True}) = \mathit{False} \vdash p(p(p(x))) = p(x)$$

After final case analysis over $p(\mathit{False}) = \mathit{True}$ this results in the two sequents

$$x = \mathit{True}, p(\mathit{True}) = \mathit{False}, p(\mathit{False}) = \mathit{True} \;\vdash\; p(p(p(x))) = p(x) \tag{11.13}$$
$$x = \mathit{True}, p(\mathit{True}) = \mathit{False} \;\vdash\; p(\mathit{False}) = \mathit{True}, p(p(p(x))) = p(x) \tag{11.14}$$

The first sequent is trivially provable by using the replacement rules that result from the formulas in the antecedent of the sequent. For the second sequent we first have to apply again (11.3) to obtain, after $\neg_R$-elimination,

$$x = \mathit{True}, p(\mathit{True}) = \mathit{False}, p(\mathit{False}) = \mathit{False} \vdash p(p(p(x))) = p(x)$$

which is also trivially provable using the replacement rules from the antecedent formulas.

11.2 Irrationality of Square Root of 2

In this section we present the CoRe proofs for the prominent first-order logic theorem about the irrationality of the square root of 2. The axioms and lemmas we assume for that proof are:

[1] Note that this sequent does not contain the quantifier for p, since the sequents are implemented via windows which denote (parts of) the FVIF-tree, while the quantifiers are maintained in the background by the corresponding indexed formula tree.

$$\forall x,y,z : \Re \,.\, (x > y \wedge y > z) \Rightarrow x > z \tag{11.15}$$
$$\forall x \,.\, s(x) > x \tag{11.16}$$
$$nat(0) \tag{11.17}$$
$$\forall x : \Re \,.\, nat(x) \Rightarrow nat(s(x)) \tag{11.18}$$
$$\forall x : \Re \,.\, nat(x) \Rightarrow nat(x^2) \tag{11.19}$$
$$\forall x : \Re \,.\, nat(x) \Rightarrow x \geq 0 \tag{11.20}$$
$$\forall x : \Re \,.\, nat(x) \Rightarrow \neg(s(x) = 0) \tag{11.21}$$
$$2 = s(s(0)) \tag{11.22}$$
$$\forall r : \Re \,.\, rat(r) \Leftrightarrow \left(\begin{array}{l} \exists n,m : \Re \,.\, nat(n) \wedge nat(m) \wedge m \times r = n \wedge \\ \neg(\exists d : \Re \,.\, nat(d) \wedge cd(n,m,d)) \end{array} \right) \tag{11.23}$$
$$\forall n,m,p : \Re \,.\, (\neg(n = 0) \wedge m \times n = p \times n) \Rightarrow m = p \tag{11.24}$$
$$\forall m,n : \Re \,.\, (m \times n^2) = (m \times n) \times n \tag{11.25}$$
$$\forall m,n : \Re \,.\, (m^2 \times n^2) = (m \times n)^2 \tag{11.26}$$
$$\forall m,m : \Re \,.\, m = n \Rightarrow m^2 = n^2 \tag{11.27}$$

$$\forall n : \Re \,.\, n \geq 0 \Rightarrow \sqrt{n^2} = n \tag{11.28}$$
$$\forall n,m,d : \Re \,.\, cd(n,m,d) \Leftrightarrow \left(\begin{array}{l} nat(n) \wedge nat(m) \wedge nat(d) \wedge \exists q_1,q_2 : \Re \,.\, nat(q_1) \\ \wedge nat(q_2) \wedge n = q_1 \times d \wedge m = q_2 \times d \end{array} \right) \tag{11.29}$$
$$\forall x : \Re \,.\, nat(x) \Rightarrow (even(x) \Leftrightarrow \exists y \,.\, nat(y) \wedge x = y \times 2) \tag{11.30}$$
$$\forall x : \Re \,.\, even(x^2) \Leftrightarrow even(x) \tag{11.31}$$

The conjecture is then $\neg(rat(\sqrt{2}))$.

Example Proof 11.2.1 of $\neg(\mathbf{rat}(\sqrt{2}))$ *To prove $\neg(rat(\sqrt{2}))$, we assume $rat(\sqrt{2})$ and derive a contradiction.*
By (11.23), the definition of rat, we derive from $rat(\sqrt{2})$

$$nat(N) \wedge nat(M) \wedge M \times \sqrt{2} = N \wedge \neg(nat(D) \wedge cd(N,M,D)) \tag{11.32}$$

where N,M, and D are instantiable variables.
By considering $nat(D) \wedge cd(N,M,D)$ we obtain:
Assuming $nat(N)$, $nat(M)$, $M \times \sqrt{2} = N$, we have to prove $nat(D) \wedge cd(N,M,D)$.
By (11.27), (11.26) and (11.28) we obtain from $M \times \sqrt{2} = N$ the new assumption $M^2 \times 2 = N^2$ and the additional goal $2 \geq 0$. That goal is proved by using (11.22), (11.15), and (11.16).
We duplicate the new assumption $M^2 \times 2 = N^2$ to preserve it for later.
By (11.30) on one of them we derive $even(N^2)$ and obtain the new goals $nat(N^2), nat(M^2)$. These goals are proved using the assumptions $nat(N)$, $nat(M)$ and lemma (11.19).
By (11.31) and (11.30) to $even(n^2)$ we obtain the new assumptions

$$N = m' \times 2 \tag{11.33}$$
$$nat(m') \tag{11.34}$$

where m' is a new parameter.
By (11.33) on the assumption $M^2 \times 2 = N^2$ we obtain $M^2 \times 2 = (m' \times 2)^2$.
By (11.26), (11.25), and (11.24) we obtain $(M^2 = m'^2 \times 2)$ and the additional goal $\neg(2 = 0)$. This goal is proved by (11.22), (11.21), and (11.18).
By (11.30) to $(M^2 = m'^2 \times 2)$ we obtain $even(M^2)$ and the side-goals $nat(M^2)$ and $nat(m'^2)$, which are easily proven by (11.19) and the assumptions $nat(M)$ and $nat(m')$.
By (11.31) and (11.30) we obtain the new assumptions

$$M = m'' \times 2 \tag{11.35}$$
$$nat(m'') \tag{11.36}$$

where m'' is a new parameter.
By (11.29), the definition of cd, to goal $nat(D) \wedge cd(N,M,D)$ we obtain

$$nat(N) \wedge nat(M) \wedge nat(D) \wedge nat(Q_1) \wedge nat(Q_2) \wedge N = Q_1 \times D \wedge M = Q_2 \times D \tag{11.37}$$

By instantiation $\{m'/Q_1, m''/Q_2, 2/D\}$ and by the assumptions (11.33) and (11.35) the goal is reduced to True, which completes the proof. □

The detailed version of that proof using the hierarchical proof datastructure from Chapter 8 is given in Appendix B.3.

CoRe Sequent Calculus Proof. Again we use the CoRe sequent calculus from Chapter 10 together with the underlying contextual reasoning capabilities provided by CoRe. The initial sequent is $\vdash \neg(rat(\sqrt{2}))$, which is reduced to $rat(\sqrt{2}) \vdash$ by the $\neg_R$-rule. The application of (11.23) reduces that sequent to

$$nat(n), nat(m), m \times \sqrt{2} = n, \neg(nat(D) \wedge cd(n,m,D)) \vdash$$

Further application of the $\neg_R$-rule and application of (11.27) and (11.26) results in

$$nat(n), nat(m), m^2 \times \sqrt{2}^2 = n^2 \vdash nat(D) \wedge cd(n,m,D)$$

The application of (11.28) results in

$$nat(n), nat(m), (2 \geq 0 \Rightarrow m^2 \times 2 = n^2) \vdash nat(D) \wedge cd(n,m,D)$$

which after $\Rightarrow_L$-elimination reduces to the two subgoals

1. $nat(n), nat(m), m^2 \times 2 = n^2 \vdash nat(D) \wedge cd(n,m,D)$, and
2. $nat(n), nat(m), \vdash 2 \geq 0, nat(D) \wedge cd(n,m,D)$.

The second subgoal can be easily proved by (11.22), (11.20), (11.18), and (11.17). The first sequent is reduced by contraction of $m^2 \times 2 = n^2$ to

$$nat(n), nat(m), m^2 \times 2 = n^2, m^2 \times 2 = n^2 \vdash nat(D) \wedge cd(n,m,D)$$

By application of (11.30) it is reduced to

$$nat(n), nat(m), nat(m^2) \Rightarrow nat(n^2) \Rightarrow even(n^2), m^2 \times 2 = n^2 \vdash nat(D) \wedge cd(n,m,D)$$

which after elimination of all implications is reduced to the three subgoals

1. $nat(n), nat(m), m^2 \times 2 = n^2 \vdash nat(m^2), nat(D) \wedge cd(n,m,D)$

2. $nat(n), nat(m), m^2 \times 2 = n^2 \vdash nat(n^2), nat(D) \wedge cd(n,m,D)$

3. $nat(n), nat(m), even(n^2), m^2 \times 2 = n^2 \vdash nat(D) \wedge cd(n,m,D)$

The first two subgoals are easily provable from $nat(m)$ and $nat(m)$ by (11.19). The third subgoal is reduced into two subgoals as follows:

$$
\begin{array}{cl}
\begin{array}{l} nat(n), nat(m), n = m' \times 2, \\ m^2 = m'^2 \times 2 \\ \vdash nat(D) \wedge cd(n,m,D) \end{array} \qquad \begin{array}{l} nat(n), nat(m), n = m' \times 2, \\ \vdash \neg(2 = 0), nat(D) \wedge cd(n,m,D) \end{array} & \\
\hline
nat(n), nat(m), n = m' \times 2, \neg(2 = 0) \Rightarrow m^2 = m'^2 \times 2, \vdash nat(D) \wedge cd(n,m,D) & \Rightarrow_L \\
\hline
nat(n), nat(m), n = m' \times 2, m^2 \times 2 = (m'^2 \times 2) \times 2, \vdash nat(D) \wedge cd(n,m,D) & \text{Apply (11.24)} \\
\hline
nat(n), nat(m), n = m' \times 2, m^2 \times 2 = m'^2 \times 2^2, \vdash nat(D) \wedge cd(n,m,D) & \text{Apply (11.25)} \\
\hline
nat(n), nat(m), n = m' \times 2, m^2 \times 2 = (m' \times 2)^2 \vdash nat(D) \wedge cd(n,m,D) & \text{Apply (11.26)} \\
\hline
nat(n), nat(m), n = m' \times 2, m^2 \times 2 = n^2 \vdash nat(D) \wedge cd(n,m,D) & Apply(n = m' \times 2) \\
\hline
nat(n), nat(m), nat(n) \Rightarrow n = m' \times 2, m^2 \times 2 = n^2 \vdash nat(D) \wedge cd(n,m,D) & \text{Apply } (nat(n)) + Simplify \\
\hline
nat(n), nat(m), even(n), m^2 \times 2 = n^2 \vdash nat(D) \wedge cd(n,m,D) & \text{Apply (11.30)} \\
\hline
nat(n), nat(m), even(n^2), m^2 \times 2 = n^2 \vdash nat(D) \wedge cd(n,m,D) & \text{Apply (11.24)}
\end{array}
$$

Again, the second subgoal can be proved by (11.22), (11.21), (11.18), and (11.17). The first subgoal is further reduced by lemma application as follows:

$$
\begin{array}{cl}
nat(n), nat(m), n = m' \times 2, nat(m^2) \Rightarrow (nat(m'^2) \Rightarrow (nat(m) \Rightarrow m = m'' \times 2)) \vdash nat(D) \wedge cd(n,m,D) & \\
\hline
nat(n), nat(m), n = m' \times 2, nat(m^2) \Rightarrow (nat(m'^2) \Rightarrow even(m)) \vdash nat(D) \wedge cd(n,m,D) & \text{Apply (11.30)} \\
\hline
nat(n), nat(m), n = m' \times 2, nat(m^2) \Rightarrow (nat(m'^2) \Rightarrow even(m^2)) \vdash nat(D) \wedge cd(n,m,D) & \text{Apply (11.31)} \\
\hline
nat(n), nat(m), n = m' \times 2, m^2 = m'^2 \times 2 \vdash nat(D) \wedge cd(n,m,D) & \text{Apply (11.30)}
\end{array}
$$

The resulting sequent reduces to four subgoals by successive elimination of all implications:

1. $nat(n), nat(m),$
   $n = m' \times 2, m = m'' \times 2 \vdash nat(D) \wedge cd(n,m,D)$

2. $nat(n), nat(m),$
   $n = m' \times 2, \vdash nat(m^2), nat(D) \wedge cd(n,m,D)$

3. $nat(n), nat(m),$
   $n = m' \times 2, \vdash nat(m'^2), nat(D) \wedge cd(n,m,D)$

4. $nat(n), nat(m),$
   $n = m' \times 2, \vdash nat(m), nat(D) \wedge cd(n,m,D)$

Again, the last three subgoals are easily provable in a few steps and we omit the detailed presentations of those subproofs. The first subgoal is finally proved by

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{-}{nat(n),nat(m),n=m'\times 2,m=m''\times 2\vdash True}\,Axiom}{nat(n),nat(m),n=m'\times 2,m=m''\times 2\vdash nat(2)\wedge nat(2)}\,(11.22),2\times(11.18),(11.17)}{\begin{array}{l}nat(n),nat(m),n=m'\times 2,m=m''\times 2\\ \vdash nat(2)\wedge nat(n)\wedge nat(m)\wedge nat(2)\wedge nat(m')\\ \wedge nat(m'')\wedge n=m'\times 2\wedge m=m''\times 2\end{array}}\,Apply\text{-}Context}{\begin{array}{l}nat(n),nat(m),n=m'\times 2,m=m''\times 2\\ \vdash nat(D)\wedge nat(n)\wedge nat(m)\wedge nat(D)\wedge nat(Q_1)\\ \wedge nat(Q_2)\wedge n=Q_1\times D\wedge m=Q_2\times D\end{array}}\,Inst(m'/Q_1,m''/Q_2,2/D)}{nat(n),nat(m),n=m'\times 2,m=m''\times 2\vdash nat(D)\wedge cd(n,m,D)}\,\text{Apply (11.29)}$$

11.3 First-Order Modal Logics

In this section we present a proof of a theorem in first-order modal logic S4.

11.3.1 Proof of $\exists \mathbf{x}\,.\,\Box(\Box\varphi(\mathbf{x})\vee\psi(\mathbf{y}))\Leftrightarrow\Box\exists\mathbf{x}\,.\,(\Box\varphi(\mathbf{x})\vee\psi(\mathbf{y}))$

The theorem is taken from [Hughes & Cresswell, 1996]. Note that $\psi(y)$ is used to indicate that the variable x does not occur in that subformula, and y is an arbitrary constant. Again, we first present a high-level CORE proof for that theorem before giving the CORE sequent calculus proof.

Example Proof 11.3.1 *We have to prove* $\exists x\,.\,\Box(\Box\varphi(x)\vee\psi(y))\Leftrightarrow\Box\exists x'\,.\,(\Box\varphi(x')\vee\psi(y))$.
By the ζ*-expansion rule it is reduced to*

$$\Box(\Box\varphi(x)\vee\psi(y))\Rightarrow\Box(\Box\varphi(X')\vee\psi(y))\Box(\Box\varphi(x')\vee\psi(y))\Rightarrow\Box(\Box\varphi(X)\vee\psi(y)) \tag{11.38}$$

We consider the two subformulas on the right hand side of the implication which results in two cases:

Case 1: *Assuming* $\Box_P(\Box_{P'}\varphi(x)\vee\psi(y))$ *we have to prove* $\Box_c(\Box_{c'}\varphi(X')\vee\psi(y))$.

By combined substitution $[c/P,c'/P',x/X']$ *and subsequent application of the assumption* $\Box_c(\Box_{c'}\varphi(x)\vee\psi(y))$ *the goal is reduced to the trivially provable formula True.*

Case 2: *Assuming* $\Box_{P''}(\Box_{P'''}\varphi(x')\vee\psi(y))$ *we have to prove* $\Box_{c''}(\Box_{c'''}\varphi(X)\vee\psi(y))$.

By combined substitution $[c''/P'',c'''/P''',x'/X]$ *and subsequent application of the assumption* $\Box_{c''}(\Box_{c'''}\varphi(x')\vee\psi(y))$ *the goal is reduced to the trivially provable formula True.* □

The detailed version of that proof using the hierarchical proof datastructure from Chapter 8 is given in Appendix B.4.

CORE Sequent Calculus Proof. The sequent calculus proof is analogous to the CORE window proof. Again, for sequent calculus derivation we use the CORE contextual reasoning capabilities on subformulas of sequents. The initial sequent is

$$\vdash\exists x\,.\,\Box(\Box\varphi(x)\vee\psi(y))\Leftrightarrow\Box\exists x'\,.\,(\Box\varphi(x')\vee\psi(y))$$

By application of the ζ-expansion rule and subsequent decomposition of the right-hand side conjunction we obtain the two subgoals:

1. $\Box_P(\Box_{P'}\varphi(x) \vee \psi(y)) \Rightarrow \Box_c(\Box_{c'}\varphi(X') \vee \psi(y))$
2. $\Box_{c''}(\Box_{c'''}\varphi(x') \vee \psi(y)) \Rightarrow \Box_{c''}(\Box_{c'''}\varphi(x') \vee \psi(y))$

The first sequent is then proved as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Box_c(\Box_{c'}\varphi(x) \vee \psi(y)) \vdash \mathit{True}}\ \text{Axiom}}{\Box_c(\Box_{c'}\varphi(x) \vee \psi(y)) \vdash \Box_c(\Box_{c'}\mathit{True} \vee \mathit{True})}\ \text{Simplify}}{\Box_c(\Box_{c'}\varphi(x) \vee \psi(y)) \vdash \Box_c(\Box_{c'}\mathit{True} \vee \psi(y))}\ \text{Apply } \psi(y) \rightarrow \langle \mathit{True} \rangle}{\Box_c(\Box_{c'}\varphi(x) \vee \psi(y)) \vdash \Box_c(\Box_{c'}\varphi(x) \vee \psi(y))}\ \text{Apply } \varphi(x) \rightarrow \langle \mathit{True} \rangle}{\Box_P(\Box_{P'}\varphi(x) \vee \psi(y)) \vdash \Box_c(\Box_{c'}\varphi(X') \vee \psi(y))}\ \text{Inst}\{c/P, c'/P', x/X'\}}{\vdash \Box_P(\Box_{P'}\varphi(x) \vee \psi(y)) \Rightarrow \Box_c(\Box_{c'}\varphi(X') \vee \psi(y))}\ \Rightarrow_R$$

The sequent proof for the second subgoal is analogous.

Conclusion

–

Part V

Chapter 12

Related work

12.1 Contextual Reasoning

The representation of context and the formalisation of contextual reasoning has played and still plays an important role in multi-agent systems and natural language dialogues. Two major perspectives on the problem of context have emerged: the so-called *metaphysical* perspective which considers contexts as being part of the structure of the world, and the so-called *cognitive* perspective that considers contexts as parts of the local cognitive states, such as an agent's cognitive state. The metaphysical approaches lie in the model-theoretic tradition from Tarski [Tarski, 1936] to Kaplan [Kaplan, 1978] and try to model all contexts within a same model, while the cognitive approaches attempt to formalise the heterogenous combination of distributed models for each local context. The research in the cognitive modelling paradigm has been initiated by McCarthy [McCarthy, 1993] and has coined the notion of local model semantics and calculi for these so-called multi-context systems. They are based on two general logical principles:

The principle of locality which advocates that everything that can be expressed and inferred is local to a context.

The principle of compatibility which advocates that two contexts may be related in such a way that reasoning in a context may affect reasoning in other contexts.

The notion of context used in this thesis is not an explicit modelling of contexts as in the approaches above. It is rather an implicit modelling of contexts that relies on the uniform notation. Furthermore, compared to the two approaches, the notion of context provided by CoRe is a hybrid of both perspectives. The notion of context for FVIF-trees follows the metaphysical approach since the contexts are inferred from the structure, and especially the uniform notation and polarities contained in the FVIF-trees. However, the window structures that are added onto the FVIF-trees are in the tradition of the cognitive perspective. The windows are based on the principle of locality by supporting the use of contextual information within a window. The logical context of a window is conditioned into replacement rules by exploiting the global or metaphysical contexts contained in FVIF-trees. Furthermore, the principle of compatibility is observed by the window structure, since derivations with respect to one window affect the context of other windows, namely those that are α-related. Thus, reasoning in one context may affect the reasoning in other contexts. However, no explicit rules are necessary to transfer knowledge from one context to another, since this is implicitly achieved by the (metaphysical) background representation.

12.1.1 Window Inference Reasoning

Window inference has been proposed in [Robinson & Staples, 1993] as a formalisation of a hierarchical structure of practical mathematical reasoning. It supports the temporary focusing on arbitrary substructures of a formula by the definition of decomposition and recomposition rules, which also explicitly construct the logical context of the substructures. It results in a hierarchy of subproblems that co-exist at a single stage of the proof. Window inference [Robinson & Staples, 1993] also allows the use of context within a window and follows strictly the cognitive approach. Although window inference rules to focus on formulas in the context of a given window have been defined in [Grundy, 1992], the context of a window is essentially static and is not directly affected by transformations of α-related windows. Furthermore, it does not provide a uniform conditioning of the information contained in a logical context as this is done in CoRe by the notion of replacement rule. Each transformation of the content of an active window results in a local lemma that needs to be tackled afterwards, while the uniform notion of replacement rule and their application in CoRe directly ensures the soundness of the respective window transformation proof steps in CoRe.

12.2 Hierarchical Reasoning

Hierarchies play an important role in any kind of reasoning. Therefore different techniques to support hierarchical reasoning are integrated in CoRe: hierarchies occur in the structure of subgoals which are addressed by the window inference capabilities. Derivational hierarchies during proof construction are addressed in the design of the hierarchical proof datastructure that allows the explicit representation of the abbreviation of partial proofs and the expansion of speculative, high level proof steps. Finally, hierarchies introduced by changing the problem representation are also taken into account by supporting the explicit representation of the arising abstraction and refinement relationships between proofs with respect to different representations. Related work with respect to the three kinds of hierarchies are discussed in the following three sections. The proof datastructure that plays an important role in the representation of hierarchies is an extension of the proof datastructure in [Cheikhrouhou & Sorge, 2000]. The differences are discussed in Section 12.2.4.

12.2.1 Hierarchies of Subproblems

Window inference [Robinson & Staples, 1993] advocates the practical advantage of having a hierarchy of subproblems without actually having to decompose the original formula. Thus, by revoking the window hierarchy the original form of the problem is preserved which is suitable for user interaction. The window inference mechanisms in CoRe exhibit the same features as it is inspired by window inferencing from [Robinson & Staples, 1993]. However, while the hierarchy is preserved by the window reasoning rules from [Robinson & Staples, 1993], the various reasoning rules in CoRe can change the hierarchical structure of the windows. This allows for a more flexible reasoning style and we have defined these changes in a uniform manner and designed the effects such that they adequately support an intuitive reasoning style.

12.2.2 Derivational Hierarchies

Derivational hierarchies are an important information for the communication of intentions about proofs. They come in two kinds, that can also be mixed at different levels: abstract proof steps can be refined or expanded into partial proofs for that abstract proof step, and partial proofs can be

abbreviated by macro-inference steps that (intuitively) describe the purpose of the proof part. The first kind of abstraction corresponds to the proof planning approach that speculates on intermediate goals by using methods, and the refinement of the abstract proof step is achieved by executing the tactic that is wrapped inside the method. This subsumes the capabilities of the hierarchical proof datastructure [Cheikhrouhou & Sorge, 2000] implemented in the ΩMEGA-proof planner [Siekmann *et al*, 2002a]. The second kind of abstraction corresponds to abstracting a partial derivation, built by some tactic T, into a macro-inference step annotated by the name of a tactic and possibly some actual parameters. This kind of abstraction of proof parts is not possible in [Cheikhrouhou & Sorge, 2000], or at least it is not possible to distinguish the abstraction relationships from the refinement relationships in [Cheikhrouhou & Sorge, 2000]. Both kinds of derivational abstractions can be represented in the hierarchical proof datastructure defined for CORE. It especially accommodates the explicit representation of the directions in the construction of abstraction relationships, i.e. top-down for the refinement of abstract proof steps and bottom-up for the abbreviation of partial proofs by macro inference steps. Thereby, it allows the categorisation of these relationships into vertical abstraction relationships, versus vertical refinement relationships introduced earlier, as informal categories of proof steps that serve the communication of information about the proof and proof intentions.

12.2.3 Representational Hierarchies

Representation is typically domain specific and it is often changed in order to simplify a given problem by mapping it into a different representation. After the mapping, a proof is constructed with respect to the new representation which can be used as a plan to perform the actual proof with respect to the original representation of the problem. The hierarchical proof datastructure in CORE supports the explicit representation of the necessary abstraction and refinement steps that occur in theorem proving by representational abstraction. Furthermore, the formal notion of a proof accommodates these relationships. The most similar work in this respect has been conducted in the context of the ABSFOL system [Giunchiglia & Villafiorita, 1996]. It supports the declarative specification of representational abstractions by sets of rewrite rules, using them for abstraction based theorem proving and for refining the proof sketch that results from the abstract proof. With the exception of the specification of abstractions, the whole scenario is supported by CORE. The specification language for abstractions in ABSFOL relies on rewrite rules. This results in that only a certain class of abstractions can be described [Plaisted, 1981]. It is insufficient to support the definition of, for instance, abstractions that take the problem into account, i.e. goal-dependent abstractions such as [Autexier & Hutter, 1997, Autexier, 1997], or those used in inductive theorem proving that rely on more subtle information annotated to symbol occurrences [Hutter, 2000a]. For these reasons we have refrained from fixing a specific specification language for abstraction functions.

12.2.4 Proof Datastructure in ΩMEGA

The hierarchical proof datastructure is an adaptation and extension of the proof datastructure used in the ΩMEGA proof-planner. Although we have already mentioned some differences between these proof datastructures in previous sections, we recapitulate all the differences in this section for the sake of completeness.

Proof Nodes. The ΩMEGA proof-planner is based on a natural deduction calculus for classical higher-order logic. Thus, the proof nodes are essentially natural deduction sequents while in CORE they consist of a window proof state and one active window with respect to that proof state. ΩMEGA

proof nodes are justified by a sequence of justifications, which are annotated by inference rules. This is essentially the same for CoRe proof nodes, though the notion of inference rule differs. Each proof node may have a sequence of justifications, and the notion of proof graphs induces a hierarchy among the justifications. However, in the ΩMEGA proof datastructure only the hierarchy of proof steps is visible, while it is not possible to distinguish whether a hierarchy has been build bottom-up, for instance by abbreviating a portion of a proof generated by some tactic with the name of the tactic, or top-down, as for instance the refinement of a speculative proof-planning step. As this is an interesting information about the proof and also serves for backtracking purposes, the CoRe proof datastructure has been designed to distinguish between abbreviation and expansion hierarchies. For this purpose the CoRe proof node justifications are split into two sequences of justifications that represent the directions in the hierarchy: the first set indicates which justification has been abbreviated by another, and the second set indicates which justification has been expanded into another. This explicitly distinguishes bottom-up hierarchies from top-down hierarchies.

In addition to the justifications, the CoRe proof nodes may also contain representational abstractions and refinements that link a proof to proof nodes with respect to a different representation. These relationships do not exist in the proof datastructure of the ΩMEGA proof-planner.

Inference Rules. The inference rules in ΩMEGA consist of the basic natural deduction inference rules and the names of proof planning methods together with actual parameters. In CoRe we distinguish between formal and informal inference rules. Formal inference rules are the CoRe window reasoning rules and they correspond to the natural deduction calculus rules in ΩMEGA. The informal inference rules encompass the names of proof planning methods, but also the names of tactics and arbitrary descriptions.

Justifications. Justifications in both systems are annotated with inference rules, although the notions of inference rule differ. Furthermore, CoRe justifications allow the assignment of so-called *methodological roles* to the subgoals of a justification. They can be used to explicitly represent information about the methodological role a subgoal plays for a particular justification, such as for instance that it is a major subgoal or only a condition, which can be used for both proof search strategies and proof presentation. This does not exist in ΩMEGA justifications.

12.3 Replacement Rules

The replacement rules play a major role in the reasoning style provided by the CoRe window reasoning calculus. There are several concepts related to the notion of replacement rules which we now discuss in more detail.

12.3.1 Modifiers in INKA

In a previous version of the inductive theorem prover INKA [Hutter & Sengler, 1996] a large set of tactics relied on the notion of *modifiers*. That version of the INKA theorem prover is based on a resolution and paramodulation calculus for classical first-order logic and used a specific normal form of clauses. Modifiers were introduced both as an abstract concept that encompasses resolution and paramodulation rules and as a specification of the operational behaviour of the application of these rules. They can be thought of as conditional rewrite rules of the form $[\varphi_1, \ldots, \varphi_n]u \rightarrow v$, where u and v could be any term or literal and the φ_i are the conditions of that rule. The determination of

these rules was based on the syntax of the formulas and their application was based on resolution and paramodulation.

With respect to the INKA concept of a modifier the replacement rules are a generalisation by removing the methodological role attributed to the formulas occurring as conditions in the modifiers and consider all these formulas as normal subgoals like *v*. Thus we separated the methodological roles of formulas from the pure logical role of these formulas, lifting the methodological role of these formulas to the proof datastructure. Additionally, the binary category of roles in INKA, i.e. "condition" and "subgoal", has been generalised to support any kind of methodological categorisation. Finally, while modifiers are restricted to literals and equations, replacement rules are defined for arbitrary formulas. This was possible due to the use of polarities and uniform notation in FVIF-trees. The use of the uniform notation was also the basis for the definition of uniform notions of contexts and replacement rules, which enabled the carrying over of that notion to further logics.

12.3.2 Assertion Level for Proof Presentation

The assertion level has been introduced in [Huang, 1996] as an abstraction from the pure natural deduction calculus. It relies on the notion of assertion level proof steps, where an assertion subsumes axioms, definitions, lemmas, and theorems. An assertion level proof step consists of the application of an assertion in some specific proof situation. Replacement rules are a generalisation of the notion of an assertion level proof step, and especially provide a concise formalisation for this, solving the problem that "introspection seems impossible to reveal the internal structure of the interpreter applying assertions" [Huang, 1996].

The assertion level is the basis for the generation of proof descriptions in natural language. The net benefit of replacement rules is that they support the construction of a proof directly on the level of assertions, which overcomes the need to build an assertion level proof by abstraction of a standard calculus proof, like natural deduction.

12.3.3 Higher-Order Rewriting

Higher-Order rewriting [Nipkow, 1991, Wolfram, 1993, Prehofer, 1994, Baader & Nipkow, 1998] has been defined for pure unconditional equational theories using a primitive notion of equality while the technique for rewriting used in this thesis relies on Leibniz' definition of equality and extensionality. Furthermore, it is designed for conditional rewriting in arbitrary theories. It is an extension of the higher-order rewriting technique, though the aspects of term-orderings and completion is outside the scope of this thesis. Note that the extensionality introduction required to support higher-order rewriting in this thesis is implicit in the higher-order rewriting techniques from [Nipkow, 1991, Wolfram, 1993, Prehofer, 1994, Baader & Nipkow, 1998] since extensionality introduction is always possible in unconditional equational theories.

12.3.4 Deduction Modulo

Theorem proving modulo [Dowek *et al*, 1998, Dowek, 2000] is a technique to integrate deduction with respect to some standard calculus, such as, for example, sequent or natural deduction calculi and term rewriting systems. The application of replacement rules is at the heart of the CoRe proof theory and thus the framework is an adequate basis for deduction modulo. Since the notion of replacement rule includes conditional rewriting as well as logical refinement of formulas to lists of subgoals it is a strict extension of the deduction modulo approach.

12.3.5 Focusing Proof Construction

Andreoli [Andreoli, 1992] introduced the notion of focusing proofs for linear logic, in order to reduce the non-determinism in proof construction by alternating phases of invertible and non-invertible steps. While focusing proofs where first introduced in [Andreoli, 1992] by means of a "triadic" sequent system that made explicit the alternating phases, the simpler presentation in [Andreoli, 2000] is better suited for a comparison with replacement rules. [Andreoli, 2000] defines the *Focusing interpretation* of a formula F as a set of derived sequent calculus inference rules. Take as an example the MALL formula $F = a^{\perp} \otimes b^{\perp}((c \,\&\, d) \parr e) \otimes f$. The Focusing interpretation of this formula is the (set of) inference rules

$$\frac{\Gamma, c, e \quad \Gamma, d, e \quad \Delta, f}{\Gamma, \Delta, a, b, F}$$

where Γ and Δ range over arbitrary multisets of formulas. Thus, the Focusing interpretation can be viewed as the possible application directions of an assertion F.

The focusing proof construction technique has also been defined, among others, for constructive first-order logic in [Abel *et al*, 2001], where it is used for proof checking lemma applications. The focusing proof steps correspond to the criteria used in the definition of admissible replacement rules. However, the major difference between the focusing proof steps and the replacement rules introduced in this thesis is that the inference rules obtained by Focus interpretation can only be determined for top-level formulas and only applied to top-level formulas, while the replacement rules can be determined for and applied to subformulas. Furthermore, the replacement rules include the treatment of conditional equivalences and equations.

12.4 Calculi

12.4.1 Schütte's Proof Theory

The proof theory presented by Schütte [Schütte, 1977] exploits the same proof theoretic properties of conjectures that lead to the definition of polarities and uniform notation to define simple calculi for classical and intuitionistic logic. Compared to sequent and natural deduction calculi it does not enforce a top-down decomposition approach, but rather supports the inner decomposition of β-type formulas. The calculus rules rely on the notions of *P-forms* and *N-forms* that can be defined as follows by using polarities: we first define unconditional forms to be $\mathcal{U}(x) := \varphi(x)$ where x is a boolean variable that occurs exactly once in φ and if $\varphi(x)$ is assigned a positive polarity, then x has polarity p, and there are no β-related formulas for x in $\varphi(x^p)^+$. All such formulas where $p = +$ are called P-forms (i.e. $\varphi(x^+)^+$, and otherwise N-forms (i.e. $\varphi(x^-)^+$ if $p = -$. Finally, an NP-form is $\mathcal{L}(x,y) := \varphi(x,y)$ such that for any formula ψ, $\mathcal{L}(\psi, y)$ is an N-form for y and $\mathcal{L}(x,\psi)$ is a P-form for x. The axioms of the classical sentential calculus from [Schütte, 1977] are $\mathcal{L}(\psi,\psi)$, $\mathcal{U}(\perp^-)$ and the only propositional inner decomposition rule is

$$\frac{\mathcal{U}(A^p)^+ \quad \mathcal{U}(B^{p'})^+}{\mathcal{U}(\beta(A^p, B^{p'})^{p''})^+}\ \beta\text{-Decompose}$$

where the formula denoted by $\mathcal{U}(A^p)^+$ is $\mathcal{U}(\neg(A^p))^+$ if $p \neq p''$, and otherwise $\mathcal{U}(A^p)^+$.

The axioms are immediately provable with the CoRE reasoning rule (simplification only for $\mathcal{U}(\perp^-)$ and replacement rule application and simplification for $\mathcal{L}(\psi,\psi)$), while the β-decomposition rule has been proved to be admissible with respect to CoRE reasoning rules in Section 10.1.

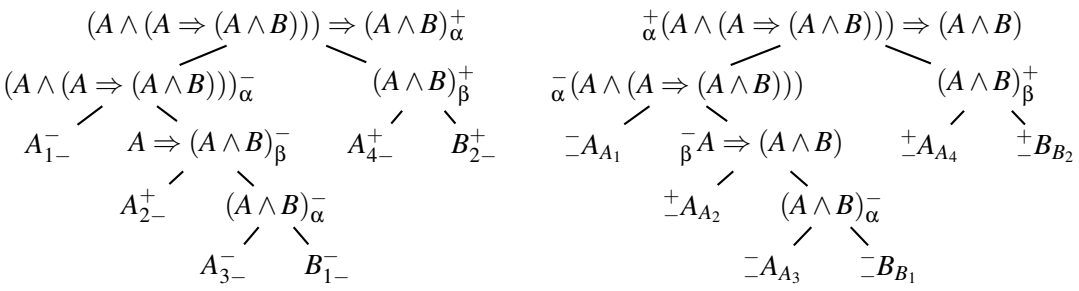


Figure 12.1: Initial indexed formula tree and initial FVIF-tree for $((A \wedge (A \Rightarrow (A \wedge B))) \Rightarrow (A \wedge B))^{+}$.

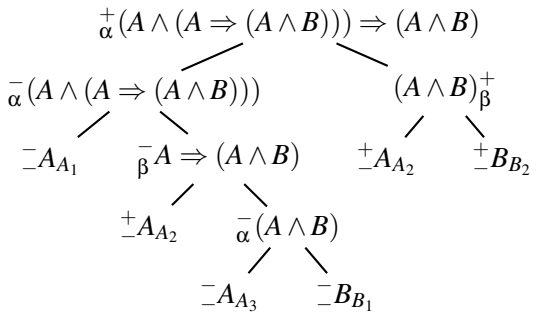
The reasoning style provided by the inference rules of the Schütte proof theory is also intuitive with respect to not enforcing the complete decomposition of the original formula. We believe it is an essential feature of a calculus for intuitive reasoning to support the transformation of parts of a formula without actually being forced to decompose the formula. In that respect the inference rules of Schütte's proof theory are a clear contribution. However, within this thesis we showed that replacement rules accommodate theorem proving on the assertion level [Huang, 1996]. Thus replacement rules are a further key technique to support an intuitive reasoning style. That notion of replacement rules is entirely absent in Schütte's proof theory, while the CoRe proof theory supports both the use of replacement rules and a generalisation of the β-formula decomposition of Schütte.

12.4.2 Matrix Calculi

The CoRe proof theory and matrix calculi (respectively expansion tree proofs) rely on the same logical foundations, but the styles of proof construction are inherently different. Consider an initial CoRe proof state $[Q, \mathsf{id} \rhd_{\angle} R]$ for some conjecture. Matrix proof search fixes the initial multiplicities of the γ- and ν-type nodes in Q and then searches for a spanning set of connections for Q, possibly, in the case of higher-order logic, by adjusting the multiplicities using Issar's path-focused duplication procedure [Issar, 1990]. The CoRe style of proof search also supports the increase of multiplicities, but otherwise consists of transforming R into either $True^{+}$ or $False^{-}$ rather than searching for a spanning set of connections. The path-focused duplication technique from [Issar, 1990] was developed for higher-order logic and supports the increase of the multiplicity of an arbitrary γ-type node. However, rather than considering all resulting new paths, the effects are localised to the path that initiated the copying. Since in our framework we rely on indexed formula trees and not on a path representation, we cannot localise the effect to some single path, and hence Issar's technique is not applicable in our context. The multiplicity increasing rule from Section 4.11 copies and renames a whole subtree and thus we are forced to consider all resulting paths. Especially, we have to carry over all proof information established for the original subtree, which is achieved by determining a convex set of subtrees before copying. It determines a set of subtrees which have to be copied in order to allow to carry over all proof information, especially established connections, to the new paths. Note that this is not necessary for the Issar's technique, since the path that triggered the adjustment of multiplicities is assumed to not contain connections.

The instantiation, Leibniz' equality introduction and extensionality introduction rules are analogous in both approaches. The CoRe weakening and contraction rules are unnecessary in a matrix

proof search. Thus, the major difference is in determining spanning connections versus application of replacement rules. At first sight there seems to be a relationship between the insertion of a connection and an application of a replacement rule; indeed, the application of a replacement rule also requires that the left-hand side of the rule and the subtree the rule is applied to have opposite polarities, the same modal prefix and the same label. This corresponds to the condition to establish a connection. The fact that replacement rules do not necessarily operate on literal nodes is not a problem, since both subtrees have isomorphic structures due to the equality of their labels and thus the relationship can be inherited along the tree structure. However, the major difference is that while in a matrix proof search both literals are α-related in Q, in the CoRE approach the original literals in Q may be α- or β-related. This is due to the fact that subformulas that are initially β-related become α-related in R via rule application. Consider as an example the formula $(A \wedge (A \Rightarrow (A \wedge B))) \Rightarrow (A \wedge B)$: the initial indexed formula tree and the corresponding FVIF-tree for the positive formula are shown in Figure 12.1 (p. 185), where we assigned numbers to the literal nodes in the indexed formula tree to distinguish them explicitly. On $A^{+}_{4\_}$ we can apply the replacement rule ${}^{-}_{\_}A_{A_3} \rightarrow \langle {}^{+}_{\_}A_{A_2} \rangle$, which results in the FVIF-tree



and introduces a connection between A_4 and A_3. On the new occurrences if A_2 we could again apply ${}^{-}_{\_}A_{A_3} \rightarrow \langle {}^{+}_{\_}A_{A_2} \rangle$; however we cannot inherit the connection information, since A_2 and A_3 are β-related in the indexed formula tree. A similar problem occurs when applying the replacement rule ${}^{-}_{\alpha}({}^{-}_{\_}A_{A_3} \wedge {}^{-}_{\_}B_{B_1}) \rightarrow \langle {}^{+}_{\_}A_{A_2} \rangle$ to ${}^{+}_{\beta}({}^{+}_{\_}A_{A_2} \wedge {}^{+}_{\_}B_{B_2})$: While it is possible to inherit the connection to (B_2, B_1), it is not possible to inherit it to (A_3, A_2). Thus, there are two kinds of connections: those between α-connected literals in Q and those between β-connected nodes in Q. The former corresponds to a standard matrix connection while the other does not. In the literature both types of relationships are known as *c-links* and *d-links* which have been used to define path resolution [Murray & Rosenthal, 1987a] and path dissolution [Murray & Rosenthal, 1987b]. That relationship has already been exploited in the completeness proof in Section 5.4, although in a slightly different way. To return to matrix proof search, the problem persists that we would have to show that by inheriting the connection information to Q during replacement rule application (in R) we obtain a spanning set of connections for Q, once R has been reduced to $True^{+}$ (respectively $False^{-}$). Due to the fact above, this is a non-trivial problem and thus is left to future work.

12.4.3 Sequent Calculus

One motivation for the development of the CoRE proof theory was to overcome the need for formula decomposition as enforced by sequent and natural deduction calculi in order to support an intuitive reasoning style. In Chapter 10 we have presented how a sequent style calculus can be simulated in CoRE. The net benefits are that the underlying CoRE proof theory not only provides a natural basis for deduction modulo for this calculus, but also supports non-trivial and practically convenient sequent calculus proof transformation operations that result from the flexible increase of multiplicities. The

proof transformation operation is also the major reason why an ongoing CoRE derivation can not be directly translated into a sequent calculus derivation. However, it should be possible to translate a completed CoRE proof into a derivation with respect to these calculi. This would result in a simple mechanism for independent proof checking of CoRE proofs, although the proof development can be done with the intuitive CoRE reasoning rules. For the definition of such a translation we envision using the techniques to generate sequent calculus proofs from completed matrix proofs, as for instance given in [Pfenning, 1987]. However, this pre-requires that a complete matrix proof can be obtained from a CoRE proof which, unfortunately, is not yet possible (see previous section).

12.4.4 Resolution and Paramodulation based Calculi

Resolution and paramodulation calculi typically rely on clausal normal form which is obtained by skolemising the Prenex normal form of the negated conjecture. The major inference rules are resolution, a generalisation of Modus Ponens, and paramodulation that can be viewed as a kind of conditional rewriting. These calculi are machine oriented calculi and they are not particularly suitable for interactive proof search. The structure of the original conjecture is lost in the normal form. The CoRE replacement rules can be viewed as a generalisation of resolution and paramodulation to quantifier free formulas that are not in normal form. The admissibility of substitutions is ensured by the quantifier structure which contrasts with the "*occur-check*" used in these calculi that relies on skolemisation. Due to the relationship between replacement rules and resolution and paramodulation rules, it should be simple to define ordering based search space restrictions for replacement rule applications analogous to those in superposition calculi [Bachmair *et al*, 1992].

Chapter 13

Conclusion

The computer-based development of mathematical proofs requires interaction of the user with the theorem proving system. Synergetic cooperation of the user and the reasoning procedures inside a theorem proving system relies especially on the quality of the interface, which must address the different requirements that arise from both sides. CoRE encompasses most aspects of the communication that range from the presentation of the proof state, via the supply of relevant contextual information about possible proof continuations, to the support for a hierarchical proof development.

The communication infrastructure has to provide a uniform interface adequate for both human users and automatic reasoning procedures. The communication infrastructure of CoRE implements this requirement and addresses the aforementioned three aspects of the communication through the following CoRE features:

1. Simultaneous presentation of the proof state as a single formula, or a list of goals and the possible alternative goals. Either style of presentation of the proof state, and especially the possibility for their simultaneous presentation, provides a complete and complementary view on the proof state.

2. Complete, contextual proof continuation information for every part of the proof state in a uniform proof construction step format. The proof construction steps allow for an intuitive reading that subsumes the assertion level proofs from [Huang, 1996] and suits both the user and declarative high-level proof planning procedures. Furthermore, they allow for an operational reading as a general inference rule, which accommodates the integration of procedural proof procedures like tactics.

3. Support for hierarchical reasoning, such as (1) the speculation of subgoals as performed, for instance, in proof planning, (2) the hierarchical reasoning style advocated by window inference, (3) the change of the representation formalism by abstraction, and (4) the explicit representation of derivational and representational hierarchies that arise during proof construction in order to adequately convey the encoded proof intentions to the user and the reasoning procedures.

The key feature for the development of the communication infrastructure is the new (meta) proof theory CoRE for contextual reasoning which is sound and complete for a variety of logics. The proof theory includes a cut rule which is admissible for all logics but higher-order logic with Henkin semantics. The pillars of the meta proof theory are:

- a uniform calculus for indexed formula trees (respectively expansion tree proofs) which is sound and complete for the whole class of logics and is used as a concise representation of variable and modal quantifier dependencies. In order to meet the requirements of interactive theorem proving, we developed a technique to dynamically increase the multiplicities of quantifiers by preserving any type of proof information. The resulting calculus rule can be used on demand during proof construction and overcomes the problem that in these calculi the multiplicities of quantifiers must either be set beforehand or that the proof information is not preserved when increasing quantifier multiplicities.

- uniform calculus extended by a free variable representation of the formula contained in an indexed formula tree and fully annotated by proof theoretic information such as polarities and uniform types. A CoRe proof state is always a pair consisting of an indexed formula tree and an actual formula with free variables. The meta proof theory consists of 12 proof rules, and a proof is completed if the formula in the proof state is $True^+$. In this way, a CoRe proof state can always be viewed as a single formula and the uniform types provide all the necessary information to determine subgoals and alternatives.

- types that provide the basis for a uniform definition and implementation of the logical context of any subformula and the replacement rules that result from formulas contained in a logical context. The notion of context and the derived replacement rules are the major developments that smoothed the way for supporting a uniform contextual reasoning style. On the one hand, replacement rules operationalise and subsume the assertion level proof rules and thus support the direct and intuitive proof development at the assertion level. On the other hand, they are generalised resolution and paramodulation rules that suit the integration of procedural proof procedures, such as tactics or superposition based proof procedures. Thus, replacement rules are the uniform proof construction step format that allow for both an intuitive and an operational reading. Providing them to the user and the reasoning procedures in the respective format is the central information about verifiably sound proof continuations. Other proof continuations, whose soundness cannot be immediately verified, such as proof planning steps and, more generally, any speculative proof construction steps, are represented using the cut rule.

We developed a window calculus to support the focusing on subparts of a formula. The CoRe window calculus supports the hierarchical reasoning style advocated by window inference. Since the underlying CoRe calculus provides all the necessary features to uniformly determine replacement rules from the logical context of any subformula, it extends previous implementations of window inference by the uniform determination of replacement rules. As an application of that window calculus we presented the implementation of a sequent calculus based on the window calculus. The resulting sequent calculus overcomes the limitations of standard implementations of these calculi, such as selecting the right order for decomposing formulas and eliminating quantifiers. Furthermore, from the underlying CoRe calculus it inherits the ability to soundly and adequately increase multiplicities of quantifiers, which corresponds to powerful proof transformation operations in standard sequent calculi. Finally, the sequent calculus inherits all contextual reasoning capabilities from the CoRe calculus and thus subsumes the theorem proving modulo approach [Dowek *et al*, 1998, Dowek, 2000].

We also provided support for the definition of reasoning domains and representational abstractions. These concepts are the formal bases supporting representational abstractions at any stage of the proof construction. Finally, we defined a hierarchical proof datastructure to represent CoRe

window proofs together with all derivational and representational hierarchies that arise during proof construction. It explicitly represents all relations between the different hierarchies and is the central representation of the history of a (partial) proof. It is the uniform representation of all information about completed proof parts as well as proof intentions for open goals.

All the techniques and solutions developed in this thesis have been implemented in the CoRe system, which is used as the communication infrastructure to support the integration of the multi-strategy proof planner Multi [Melis & Meier, 2000, Meier, 2003], the agent-based proof system Ω-ants [Benzmüller & Sorge, 2000, Hübner, 2003], and the inductive theorem prover InKa [Hutter & Sengler, 1996, Autexier *et al*, 1999].

13.1 Future Work

The work presented in this thesis provides a basis for future work in many respects: proof checking, automation of proof search, user interaction and foundational research.

Proof Checking. Proof checking of proofs constructed within a theorem proving environment is an important issue in order to independently certify these proofs. The reasoning rules of the CoRe meta proof theory can not easily be checked by pure syntactical properties, as they, for instance, rely on uniform notation, polarities and may have non-trivial global effects. The CoRe proof state contains an indexed formula tree and thus if it would be possible to construct a matrix proof during a CoRe proof construction, it would in principle allow one to use the techniques that generate sequent or natural deduction calculus proofs from completed matrix proofs. The formalisation of the close relationship between replacement rule application and the insertion of connections discussed in Section 12.4.2 together with an appropriate proof ensuring that any completed CoRe proof state corresponds to a spanning set of connections for its indexed formula tree, is the first future work step in the direction of proof checking. Although the formalisation and the proof are sufficient from a theoretical point of view, the actual implementation of a procedure that generates a sequent calculus proof still needs to be done and is a non-trivial task.

An alternative direction towards proof checking is the development of an assertion level proof representation that is suitable for proof checking. A possible starting point could, for instance, be the assertion level proof calculus developed for first-order logic proofs in [Abel *et al*, 2001]. The development of variants of this calculus for the logics considered in this thesis would make it conceivable to define an adequate transformation of a (partial) CoRe window calculus proof.

Automation of Proof Search. The embedding of a strong sequent calculus in CoRe together with the replacement rules that behave like assertion level proof steps on the one hand and generalised resolution and paramodulation proof steps on the other hand, makes it plausible that the proof search automation techniques developed for these calculi can be adapted to the CoRe proof theory. This includes tactical theorem proving for sequent calculus including the theorem proving modulo approach, or proof planning over tactics. Furthermore, the assertion level character of replacement rules supports the definition of proof planning directly on the intuitive and human-oriented assertion level. Also the term-ordering based automated theorem proving procedures for resolution and paramodulation calculi should be transferable to the CoRe setting. Finally, the close relationship of the internal CoRe proof state representation to matrix calculi gives rise to a further interesting line of research concerned with the transfer of proof search automation techniques from that area.

User Interaction. Natural language presentation of proofs translates formal logic proofs into a style as can be found in a mathematical textbook. The assertion level of [Huang, 1996] is the basis for these techniques and usually must be reconstructed from the given sequent or natural deduction proofs. These techniques also support the natural language presentation of partial proofs. The CoRe reasoning rules and especially the replacement rules allow for a direct construction of proofs at the assertion level.

Foundational Research. In contrast to the more application oriented research suggested above, there is also interesting foundational future work. The CoRe meta proof theory relies on the existing Wallen-style matrix characterisations [Wallen, 1990], which hampers the addition of new logics. Foundational work should therefore support the definition of new logics inside CoRe, for example by specifying the sequent calculus rules for that logic, analogous to the definition of logics in logical frameworks [Harper *et al*, 1987, Pfenning, 1996, Pfenning & Schürmann, 1999]. The automatic generation of a kind of Wallen style matrix characterisation from these descriptions would allow intuitive and contextual reasoning capabilities for the new logics. But achieving this task is likely as difficult as it is attractive. However, a good starting point would be to attempt to integrate the logics for which Wallen style matrix characterisations already exist [Wallen, 1990, Mantel & Kreitz, 1998].

References

[Abel *et al*, 2001] Abel, Andreas, Chang, Bor-Yuh Evan and Pfenning, Frank. (June 2001). Human-readable machine-verifiable proofs for teaching constructive logic. In Egly, Uwe, Fiedler, Armin, Horacek, Helmut and Schmitt, Stephan, (eds.), *Proceedings of the Workshop on Proof Transformations, Proof Presentations and Complexity of Proofs (PTP'01)*. Universitá degli studi di Siena.

[Abelson *et al*, 1996] Abelson, H., Sussman, G. and Sussman, J. (1996). *Structure and Interpretation of Computer Programs*. MIT Press.

[Andreoli, 1992] Andreoli, Jean-Marc. (1992). Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3).

[Andreoli, 2000] Andreoli, Jean-Marc. (2000). Focussing and proof construction. *Annals of Pure and Applied Logic*, 107(1):131–163.

[Andrews, 1972] Andrews, Peter B. (June 1972). General models, descriptions, and choice in type theory. *The Journal of Symbolic Logic*, 37(2):385–397.

[Andrews, 1981] Andrews, Peter B. (April 1981). Theorem proving via general matings. *Journal of the Association for Computing Machinery*, 28(2):193–214.

[Andrews, 1989] Andrews, Peter B. (1989). On connections in higher-order logic. *Journal of Automated Reasoning*, 5:257–291.

[Andrews, 2002] Andrews, Peter B. (2002). *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, volume 27 of *Gabbay, Dov M. and Barwise, Jon, editor, Applied Logic Series*. Kluwer Academic Publishers, Dordrecht / Boston / London, second edition.

[Andrews *et al*, 1990] Andrews, Peter B., Issar, Sunil, Nesmith, Dan and Pfenning, Frank. (July 1990). The TPS Theorem Proving System. In Stickel, Mark E., (ed.), *Proceedings 10^{th} International Conference on Automated Deduction (CADE)*, volume 449 of *LNAI* , pages 641–642. Springer Verlag.

[Andrews *et al*, 2000] Andrews, Peter B., Bishop, Matthew and Brown, Chad E. (2000). System Description: TPS: A Theorem Proving System for Type Theory. volume 1831 of *Lecture notes in computer science*, pages 164–169. Springer.

[Autexier & Hutter, 1997] Autexier, Serge and Hutter, Dieter. (October 1997). Equational proof-planning by dynamic abstraction. In Bonacina, Maria Paola and Furbach, Ulrich, (eds.), *Proceedings of FTP97: International Workshop First-Order Theorem Proving*, number 97-50 in Report Series, pages 1–6, Johannes Kepler Universität, 4040 Linz, Austria. RISC-Linz.

[Autexier, 1997] Autexier, Serge. (June 1997). An abstraction for proof-planning: The $\mathcal{S}$-abstraction. SEKI Report SR-97-05, Universität des Saarlandes, Fachbereich Informatik, Postfach 15 11 50, D–66041 Saarbrücken.

[Autexier *et al*, 1998] Autexier, S., Hutter, D., Langenstein, B., Mantel, H., Rock, G., Schairer, A., Stephan, W., Vogt, R. and Wolpers, A. (september 1998). VSE: Formal methods meet industrial needs. *International Journal on Software Tools for Technology Transfer, Special issue on Mechanized Theorem Proving for Technology, Springer Verlag*.

[Autexier *et al*, 1999] Autexier, Serge, Hutter, Dieter, Mantel, Heiko and Schairer, Axel. (1999). System description: Inka 5.0 – a logic voyager. In Ganzinger, H., (ed.), *Proceedings of the 16th International Conference on Automated Deduction (CADE)*, LNAI 1632, Trento, Italy. Springer.

[Autexier *et al*, 2002] Autexier, Serge, Hutter, Dieter, Mossakowski, Till and Schairer, Axel. (September 2002). The development graph manager MAYA. In Kirchner, Hélène and Ringeissen, Christophe, (eds.), *Proceedings 9th International Conference on Algebraic Methodology And Software Technology (AMAST'02)*, volume 2422 of *LNCS*. Springer.

[Baader & Nipkow, 1998] Baader, Franz and Nipkow, Tobias. (1998). *Term* Rewriting *and* All That. Cambridge University Press.

[Bachmair *et al*, 1992] Bachmair, Leo, Ganzinger, Harald, Lynch, Christopher and Snyder, Wayne. (June 1992). Basic paramodulation and superposition. In Kapur, Deepak, (ed.), *Proceedings of the 11th International Conference on Automated Deduction (CADE)*, volume 607 of *LNAI* , Saratoga Springs, NY. Springer.

[Barendregt, 1984] Barendregt, Henk P. (1984). *The Lambda Calculus – Its Syntax and Semantics*. North Holland.

[Benzmüller & Sorge, 1999] Benzmüller, Christoph and Sorge, Volker. (21–24, September 1999). Critical Agents Supporting Interactive Theorem Proving. In Barahona, P. and Alferes, J. J., (eds.), *Progress in Artificial Intelligence, Proceedings of the 9th Portuguese Conference on Artificial Intelligence (EPIA-99)*, volume 1695 of *LNAI*, pages 208–221, Évora, Portugal. Springer Verlag, Berlin, Germany.

[Benzmüller & Sorge, 2000] Benzmüller, Christoph and Sorge, Volker. (2000). Ω-OANTS – an open approach at combining interactive and automated theorem proving. In Kerber, Manfred and Kohlhase, Michael, (eds.), *Symbolic Computation and Automated Reasoning*, pages 81–97. A.K.Peters.

[Benzmüller *et al*, 2002a] Benzmüller, Christoph, Kohlhase, Michael and Brown, Chad E. (2002a). Higher Order Semantics and Extensionality. Technical report, Carnegie Mellon University, Pittsburgh, PA.

[Benzmüller *et al*, 2002b] Benzmüller, Christoph, Kohlhase, Michael and Brown, Chad E. (2002b). Semantic techniques for cut-elimination in higher order logic. Technical report, Carnegie Mellon University, Pittsburgh, PA.

[Bernays, 1937] Bernays, Paul. (1937). A system of axiomatic set-theory. *Journal of Symbolic Logic*, 2:65–77.

[Bernays, 1941] Bernays, Paul. (1941). A system of axiomatic set-theory. *Journal of Symbolic Logic*, 6:1–17.

[Beth, 1965] Beth, Evert W. (1965). *The foundations of mathematics : a study in the philosophy of science*. Studies in logic and the foundations of mathematics. North-holland, 2nd rev. ed. edition.

[Boole, 1847] Boole, George. (1847). *The Mathematical Analysis of Logic*. Macmillan, Barclay, Cambridge, UK, Reprinted by Basil Blackwell, Oxford, UK, 1965.

[Boyer & Moore, 1979] Boyer, Robert S. and Moore, J Strother. (1979). *A computational logic*. ACM monograph series. Academic Press.

[Brouwer, 1914] Brouwer, Luitzen Egbertus Jan. (1914). Intuitionism and Formalism. *Bulletin of the American Mathematical Society*, 20:81–96.

[Brouwer, 1925] Brouwer, Luitzen Egbertus Jan. (1925). Zur Begründung der intuitionistischen Mathematik. *Mathematische Annalen*, 93:244–257.

[Bundy, 1988] Bundy, Alan. (1988). The use of explicit plans to guide inductive proofs. In Lusk, R. and Overbeek, R., (eds.), *Proceedings 9^{th} International Conference on Automated Deduction (CADE)*, LNAI , pages 111–120. Springer.

[Bundy *et al*, 1990a] Bundy, A., van Harmelen, F., Horn, C. and Smaill, A. (July 1990). The Oyster-Clam system. In Stickel, Mark E., (ed.), *Proceedings 10^{th} International Conference on Automated Deduction (CADE)*, volume 449 of *LNAI* , pages 647–648. Springer Verlag.

[Bundy *et al*, 1990b] Bundy, A., van Harmelen, F., Smaill, A. and Ireland, A. (July 1990). Extension to the rippling-out tactic for guiding inductive proofs. In Stickel, Mark E., (ed.), *Proceedings 10^{th} International Conference on Automated Deduction (CADE)*, volume 449

of *LNAI* , pages 132–146, Kaiserslautern, Germany. Springer Verlag.

[Bundy *et al*, 2003] Bundy, Alan, Basin, David, Hutter, Dieter and Ireland, Andrew. (2003). *Rippling: Meta-Level Guidance for Mathematical Reasoning*. Cambridge University Press.

[Carlsson, 1984] Carlsson, M. (1984). On implementing prolog in functional programming. *New Generation Computing*, 2(4).

[Cheikhrouhou & Sorge, 2000] Cheikhrouhou, Lassaad and Sorge, Volker. (march 2000). PDS – a three-dimensional data structure for proof plans. In *Proceedings of the International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000)*.

[Church, 1936] Church, Alonzo. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics*.

[Church, 1940] Church, Alonzo. (1940). A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68.

[Constable *et al*, 1986] Constable, Robert L., Allen, Stuart F., Bromley, H.M., Cleaveland, W.R., Cremer, J.F., Harper, R.W., Howe, Douglas J., Knoblock, T.B., Mendler, N.P., Panangaden, P., Sasaki, James T. and Smith, Scott F. (1986). *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall, NJ.

[Dahn *et al*, 1997] Dahn, Bernd Ingo, Gehne, J, Honigmann, Th. and Wolf, A. (1997). Integration of automated and interactive theorem proving in ILF. In McCune, W., (ed.), *Proceedings of the 14^{th} International Conference on Automated Deduction (CADE)*, LNAI 1249, pages 57–60, Townsville, North Queensland, Australia. Springer.

[De Bruijn, 1973a] De Bruijn, Nicolaas Govert. (1973a). AUTOMATH - Ein Projekt zur Kontrolle von Mathematik. In Braffort, P., (ed.), *Proceedings of the symposium APLASM*, volume I, Orsay, France. Talk given at Innsbrucker Mathematikertag, 1974. German translation of "The AUTOMATH Mathematics Checking Project".

[De Bruijn, 1973b] De Bruijn, Nicolaas Govert, (1973b). AUTOMATH, A Language for Mathematics. Séminaire de Mathématiques Superieures 52, Département de Mathématiques, Université de Montréal, Montréal, Canada.

[De Bruijn, 1980] De Bruijn, Nicolaas Govert. (1980). A survey of the project AUTOMATH. In Seldin, J. P. and Hindley, J. R., (eds.), *To H. B. Curry - Essays on the Combinatory Logic, Calculus and Formalism*, pages 579–606. Academic Press, London, UK.

[Dowek, 2000] Dowek, Gilles. (2000). Automated theorem proving in first-order logic modulo: on the difference between type theory and set theory. In Cafferra, R and Salzer, G., (eds.), *Automated Deduction in Classical and Non-Classical Logics*, number 1761 in LNAI, pages 1–22. Springer-Verlag.

[Dowek *et al*, 1998] Dowek, Gilles, Hardin, Thérèse and Kirchner, Claude. (April 1998). Theorem proving modulo. Rapport de Recherche 3400, Institut National de Recherche en Informatique et en Automatique.

[Duncan & Lowe, 1997] Duncan, David and Lowe, Helen. (1997). Xbarnacle: Making theorem provers more accessible. In McCune, W., (ed.), *Proceedings of the 14th International Conference on Automated Deduction (CADE)*, LNAI 1249, Townsville, North Queensland, Australia. Springer.

[Eisinger & Ohlbach, 1986] Eisinger, Norbert and Ohlbach, Hans-Jürgen. (1986). The Markgraf Karl Refutation procedure (MKRP). In Siekmann, Jörg, (ed.), *Proceedings of the 8th International Conference on Automated Deduction (CADE)*, LNCS , pages 681–682. Springer.

[Elliott & Pfenning, 1991] Elliott, Conal and Pfenning, Frank. (1991). A semi-functional implementation of a higher-order logic programming language. In Lee, Peter, (ed.), *Topics in Advanced Language Implementation*, pages 289–325. MIT Press.

[Fiedler, 2001] Fiedler, Armin. (2001). *User-adaptive Proof Explanation*. Phd thesis, Naturwissenschaftlich-Technische Fakultät I, Universität des Saarlandes, Saarbrücken, Germany.

[Fitting, 1972] Fitting, Melvin. (1972). Tableau methods of proof for modal logics. *Notre Dame Journal of Formal Logic*, XIII:237–247.

[Fraenkel, 1922] Fraenkel, Adolf Abraham. (1922). Zu den Grundlagen der Cantor-Zermeloschen Mengenlehre. *Mathematische Annalen*, 86:230–237.

[Franke & Kohlhase, 1999] Franke, Andreas and Kohlhase, Michael. (1999). Mbase: Representing mathematical knowledge in a relational data base. In *CALCULEMUS 99, Systems for Integrated Computation and Deduction*. Elsevier.

[Frege, 1879] Frege, Gottlob, (1879). Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens, Halle, Germany, Reprint in: Begriffsschrift und andere Aufsätze, J. Angelelli, editor, Hildesheim. See also in Logiktexte, Karel Berka, Lothar Kreiser, editors, pages 82-112.

[Ganzinger & Waldmann, 1996] Ganzinger, Harald and Waldmann, Uwe. (1996). Theorem proving in cancellative abelian monoids. In McRobbie, M. A. and

Slaney, J. K., (eds.), *Proceedings of the 13th International Conference on Automated Deduction (CADE)*, volume 1104 of *LNCS* , pages 388–402, New Brunswick, N. Y. Springer.

[Gentzen, 1969] Gentzen, Gerhard. (1969). *The Collected Papers of Gerhard Gentzen (1934-1938)*. Edited by Szabo, M. E., North Holland, Amsterdam.

[Giunchiglia & Villafiorita, 1996] Giunchiglia, Fausto and Villafiorita, Adolfo. (1996). ABSFOL: A proof checker with abstraction. In McRobbie, M. A. and Slaney, J. K., (eds.), *Proceedings of the 13th International Conference on Automated Deduction (CADE)*, volume 1104 of *LNCS* , pages 136–140, New Brunswick, N. Y. Springer.

[Gödel, 1930] Gödel, Kurt. (1930). Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematik*, 37:349–360.

[Gödel, 1931] Gödel, Kurt. (1931). Über formal entscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematic und Physik*, 38:173–198.

[Gödel, 1940] Gödel, Kurt. (1940). The Consistency of the Axiom of Choice and of the Generalized Continuum-Hypothesis with the Axioms of Set Theory. *Annals of Mathematics Studies*, 3.

[Gordon *et al*, 1979] Gordon, M. J., Milner, A. J. and Wadsworth, C. P. (1979). *Edinburgh LCF – A mechanised logic of computation*. Springer Verlag, LNCS 78.

[Graham, 1994] Graham, P. (1994). *On Lisp – Advanced Techniques for Common Lisp*. Prentice Hall.

[Grundy, 1991] Grundy, Jim. (1991). Window inference in the HOL system. In *Proceedings of the International Workshop on the HOL Theorem Proving System and its Applications*.

[Grundy, 1992] Grundy, Jim. (1992). A window inference tool for refinement. In *Proceedings of the Fifth Refinement Workshop, Workshop in Computer Science*, pages 230–254. Springer Verlag.

[Harper *et al*, 1987] Harper, Robert, Honsell, Furio and Plotkin, Gordon. (June 22-25 1987). A framework for defining logics. In *Proceedings of the Symposium on Logic in Computer Science (LICS '87)*, pages 194–204, Ithaca, New York, USA. IEEE Computer Society Press.

[Heisel *et al*, 1991] Heisel, M., Reif, W. and Stephan, W. (1991). Formal software development in the KIV system. In Lowry, M. R. and McCartney, R. D., (eds.), *Automating Software Design*, pages 547–574. AAAI Press, Menlo Park, CA.

[Henkin, 1950] Henkin, Leon. (1950). Completeness in the theory of types. *The Journal of Symbolic Logic*, 15:81–91.

[Herbrand, 1930] Herbrand, Jacques. (1930). Recherches sur la théorie de la démonstration. *Sci. Lett. Varsovie, Classes III sci. math. phys.*, 33.

[Heyting, 1956] Heyting, Arend. (1956). *Intuitionism*. North-Holland Publishing Company, Amsterdam, Netherlands, (1971) third edition.

[Hilbert, 1930] Hilbert, David. (1930). Probleme der Grundlegung der Mathematik. *Mathematische Annalen*, 102:1–9.

[Hindley & Seldin, 1986] Hindley, J. Roger and Seldin, Jonathan P. (1986). *Introduction to Combinators and λ-Calculus*, volume 1 of *London Mathematical Society Student Texts*. Cambridge University Press.

[Horacek, 1999] Horacek, Helmut. (1999). Presenting proofs in a human-oriented way. In Ganzinger, H., (ed.), *Proceedings of the 16th International Conference on Automated Deduction (CADE)*, LNAI 1632, pages 142–156, Trento, Italy. Springer.

[Huang, 1994] Huang, Xiaorong. (1994). *Human Oriented Proof Presentation: A Reconstructive Approach*. Phd thesis, FB 14 Informatik, Saarland University.

[Huang, 1996] Huang, Xiaorong. (1996). *Human Oriented Proof Presentation: A Reconstructive Approach*. Number 112 in DISKI. Infix, Sankt Augustin, Germany, Also published as [Huang, 1994].

[Hübner, 2003] Hübner, Malte. (2003). Supporting interactive theorem proving in CORE. Diploma thesis, FR 6.2 Informatik, Saarland University.

[Hughes & Cresswell, 1996] Hughes, G. E. and Cresswell, M. J. (1996). *A New Introduction to Modal Logic*. Routledge, 11 New Fetter Lane, London, EC4P 4EE.

[Hutter & Sengler, 1996] Hutter, Dieter and Sengler, Claus. (1996). INKA - The Next Generation. In McRobbie, M. A. and Slaney, J. K., (eds.), *Proceedings of the 13th International Conference on Automated Deduction (CADE)*, volume 1104 of *LNCS* , New Brunswick, N. Y. Springer.

[Hutter, 1990] Hutter, Dieter. (July 1990). Guiding induction proofs. In Stickel, Mark E., (ed.), *Proceedings 10th International Conference on Automated Deduction (CADE)*, volume 449 of *LNAI* . Springer Verlag.

[Hutter, 1994] Hutter, Dieter. (1994). Synthesizing induction orderings for existence proofs. In Bundy, Alan, (ed.), *Proceedings of the 12th International Conference on Automated Deduction (CADE)*, LNAI , pages 29–41, Nancy, France. Springer.

[Hutter, 1997a] Hutter, Dieter. (1997a). Colouring terms to control equational reasoning. *Journal of Automated Reasoning*, 18:399–442.

[Hutter, 1997b] Hutter, Dieter. (1997b). Equalizing terms by difference reduction techniques. In Kirchner, H. and Gramlich, B., (eds.), *Workshop on Strategies in Automated Deduction, Townsville, Australia, 14th International Conference on Automated Deduction, CADE-14*.

[Hutter, 2000a] Hutter, Dieter. (2000a). Annotated reasoning. *Annals of Mathematics and Artificial Intelligence (AMAI), Special Issue on Strategies in Automated Deduction*.

[Hutter, 2000b] Hutter, Dieter. (2000b). Management of change in structured verification. In *Proceedings of Automated Software Engineering, ASE-2000*. IEEE.

[Issar, 1990] Issar, Sunil. (1990). Path-focused duplication: A search procedure for general matings. In The American Association for Artificial Intelligence (AAAI), (ed.), *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI 90), Vol. 1, July 29 - August 3, 1990: Proceedings 5V Vol. 1*, pages 221–226, Menlo Park - Cambridge - London. AAAI Press / MIT Press.

[Jamnik *et al*, 1997] Jamnik, Mateja, Bundy, Alan and Green, Ian. (1997). Automation of diagrammatic reasoning. In Pollack, M. E., (ed.), *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI), August*, volume 1, pages 528–533, San Mateo, CA. Morgan Kaufmann Publisher.

[Jamnik *et al*, 1999] Jamnik, Mateja, Bundy, Alan and Green, Ian. (1999). On automating diagrammatic proofs of arithmetic arguments. *Journal of Logic, Language and Information*, 8(3):297–321.

[Kaplan, 1978] Kaplan, D. (1978). Logic of demonstratives. *Journal of Philosophical Logic*, 8.

[Kerber, 1992] Kerber, Manfred. (1992). *On the Representation of Mathematical Concepts and their Translation into First Order Logic*. Phd thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany.

[Kerber *et al*, 1998] Kerber, Manfred, Kohlhase, Michael and Sorge, Volker. (1998). Integrating computer algebra into proof planning. *Journal of Automated Reasoning*, 21(3):327–355.

[Kohlhase, 2000] Kohlhase, Michael. (2000). OMDOC: Towards an Internet Standard for the Administration, Distribution and Teaching of mathematical Knowledge. In Campbell, John A. and Roanes-Lozano, Eugenio, (eds.), *Proceedings of Artificial intelligence and symbolic computation (AISC-00)*, volume 1930 of *LNCS*. Springer.

[Letz & Stenz, 1999] Letz, Reinhold and Stenz, Gernot. (1999). Model elimination and connection tableau procedures. In Robinson, A. and Voronkov,

A., (eds.), *Handbook of Automated Reasoning*, chapter 28, pages 2015–2114. Elsevier.

[Lüth *et al*, 1999] Lüth, Christoph, Tej, H, Kolyang and Krieg-Brückner, Bernd. (1999). TAS and IsaWin: Tools for transformational program development and theorem proving. In *Proceedings of the European Joint Conference on Theory and Practice of Software (ETAPS'99)*, number 1577 in LNCS. Springer.

[Mantel & Kreitz, 1998] Mantel, Heiko and Kreitz, Christoph. (October 1998). A Matrix Characterization for MELL. In Dix, J., del Cerro, L. Farinas and Furbach, U., (eds.), *Proceedings of Logics in Artificial Intelligence, European Workshop, JELIA '98*, LNAI 1489, pages 169–183, Dagstuhl, Germany. Springer.

[McCarthy, 1993] McCarthy, Jon. (August 28 - September 3 1993). Notes on formalizing context. In *Artificial intelligence (IJCAI-93) : 13th International Joint Conference on Artificial Intelligence*, volume 2, Chambéry, France. Morgan Kaufman.

[McCune, 1990] McCune, William. (July 1990). OTTER 2.0. In Stickel, Mark E., (ed.), *Proceedings 10^{th} International Conference on Automated Deduction (CADE)*, volume 449 of *LNAI* , pages 663–664. Springer Verlag.

[Meier, 2000] Meier, Andreas. (2000). System Description: TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. volume 1831 of *Lecture notes in computer science*, pages 460–464. Springer.

[Meier, 2003] Meier, Andreas. (2003). *Proof Planning with Multiple Strategies*. Phd thesis, FR 6.2 Informatik, Saarland University, forthcoming.

[Melis & Meier, 2000] Melis, Erica and Meier, Andreas. (2000). Proof Planning with Multiple Strategies. In Loyd, J., Dahl, V., Furbach, U., Kerber, M., Lau, K., Palamidessi, C., Pereira, L.M. and Stuckey, Y. Sagivand P., (eds.), *First International Conference on Computational Logic (CL-2000)*, volume 1861 of *LNAI*, pages 644–659, London, UK. Springer-Verlag.

[Melis *et al*, 2001] Melis, Erica, Andrès, Eric, Büdenberger, Jochen, Frischauf, Adrian, Goguadze, George, Libbrecht, Paul, Pollet, Martin and Ullrich, Carsten. (2001). Activemath: A generic and adaptive web-based learning environment. *Artifical Intelligence in Education*, 12(4).

[Miller, 1983] Miller, Dale A. (1983). *Proofs in Higher-Order Logic*. Phd thesis, Carnegie Mellon University.

[Murray & Rosenthal, 1987a] Murray, Neil V. and Rosenthal, Erik. (April 1987). Inference with path resolution and semantic graphs. *Journal of the Association of Computing Machinery*, 34(2):225–254.

[Murray & Rosenthal, 1987b] Murray, Neil V. and Rosenthal, Erik. (July 12-17 1987). Path dissolution: A strongly complete inference rule. In *Proceedings of the* 6^{th} *National Conference on Artificial Intelligence*, pages 161–166, Seattle, WA.

[Nelson & Oppen, 1977] Nelson, Greg and Oppen, Derek C. (October 1977). Fast decision algorithms based on union and find. In *Proceedings of the* 18^{th} *Annual Symposium on Foundations of Computer Science*, pages 114–119. American Mathematical Society.

[Newell *et al*, 1957] Newell, Allen, Shaw, Cliff and Simon, Herbert. (1957). Empirical explorations with the logic theory machine: A case study in heuristics. In *Proceedings of the 1957 Western Joint Computer Conference*, New York, USA. McGraw-Hill. reprinted in Computer and Thoughts, Edward A. Feigenbaum, Julian Eldman, editors, new York, USA, 1963.

[Nipkow, 1991] Nipkow, Tobias. (1991). Higher-order critical pairs. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, pages 342–349. IEEE Computer Society Press.

[Nonnengart, 1995] Nonnengart, Andreas. (1995). *A Resolution-Based Calculus for Temporal Logics*. Phd thesis, Computer Science Department, Saarland University.

[Norvig, 1992] Norvig, Peter. (1992). *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann.

[Otten & Kreitz, 1996] Otten, Jens and Kreitz, Christoph. (1996). T-string unification: Unifying prefixes in non-classical proof methods. In Miglioli, P., Moscato, U., Mundici, D. and Ornaghi, M., (eds.), *Proceedings of* 5^{th} *Workshop on theorem Proving with analytic tableaux and related methods*, LNAI 1071, pages 244–260. Springer Verlag.

[Paulson, 1989] Paulson, Lawrence C. (1989). The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5:363–397.

[Pfenning & Elliott, 1988] Pfenning, Frank and Elliott, Conal. (1988). Higher order abstract syntax. In *Proceedings of the ACM SIGPLAN '88 Symposium on Language Design and Implementation*, pages 199–208.

[Pfenning & Schürmann, 1999] Pfenning, Frank and Schürmann, Carsten. (1999). System description: Twelf - a meta-logical framework for deductive systems. In Ganzinger, H., (ed.), *Proceedings of the* 16^{th} *International Conference on Automated Deduction (CADE)*, LNAI 1632, pages 202–206, Trento, Italy. Springer.

[Pfenning, 1987] Pfenning, Frank. (1987). *Proof Transformation in Higher-Order Logic*. Phd thesis, Carnegie Mellon University.

[Pfenning, 1996] Pfenning, Frank. (April 22-24 1996). The practice of logical frameworks. In *Trees in Algebra and Programming - CAAP'96, 21st International Colloquium*, volume 1059 of *LNCS*, pages 119–134, Linköping, Sweden.

[Plaisted, 1981] Plaisted, David. (1981). Theorem Proving with Abstractions. *Journal of Artificial Intelligence*, 16:47–108.

[Prehofer, 1994] Prehofer, Christian. (1994). Higher-order narrowing. In *Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science*, pages 507–516, Paris, France. IEEE Computer Society Press.

[Reynolds, 1993] Reynolds, J. C. (1993). The discoveries of continuations. *Lisp and Symbolic Computation*, 6.

[Riazanov & Voronkov, 2001] Riazanov, Alexander and Voronkov, Andrei. (2001). Vampire 1.1 (system description). In Goré, Rajeev, Leitsch, Alexander and Nipkow, Tobias, (eds.), *Automated Reasoning*, volume 2083 of *LNAI*, pages 376–380.

[Robinson & Staples, 1993] Robinson, Peter D. and Staples, John. (1993). Formalizing a hierarchical structure of practical mathematical reasoning. *Journal of Logic and Computation*, 3(1):47–61.

[Robinson, 1965] Robinson, John Alan. (1965). A machine oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41.

[Schairer *et al*, 2001] Schairer, Axel, Autexier, Serge and Hutter, Dieter. (June 2001). A pragmatic approach to reuse in tactical theorem proving. In Bonacina, Maria-Paola and Gramlich, Bernhard, (eds.), *Proceedings of the 4^{th} Workshop on Strategies in Automated Deduction (STRATEGIES'01)*, volume TR DII 10/01, pages 75–86. Universitá degli studi di Siena.

[Schütte, 1977] Schütte, Kurt. (1977). *Proof Theory. (Originaltitel: Beweistheorie)*, volume 255 of *Die Grundlehren der mathematischen Wissenschaften*. Springer, Berlin;Heidelberg;New York.

[Siekmann, 1987] Siekmann, Jörg. (1987). Unification Theory. *Journal of Symbolic Computation*.

[Siekmann *et al*, 1999] Siekmann, Jörg, Hess, Stephan, Benzmüller, Christoph, Cheikhrouhou, Lassaad, Fiedler, Armin, Horacek, Helmut, Kohlhase, Michael, Konrad, Karsten, Meier, Andreas, Melis, Erica, Pollet, Martin and Sorge, Volker. (1999). LΩUI: Lovely Ωmega user interface. *Formal Aspects of Computing*, 11:326–342.

[Siekmann *et al*, 2002a] Siekmann, Jörg, Benzmüller, Christoph, Brezhnev, Vladimir, Cheikhrouhou, Lassaad, Fiedler, Armin, Franke, Andreas, Horacek, Helmut, Kohlhase, Michael, Meier, Andreas, Melis, Erica, Moschner, Markus, Normann, Immanuel, Pollet, Martin, Sorge, Volker, Ullrich, Carsten, Wirth, Claus-Peter and Zimmer, Jürgen. (2002a). Proof development with OMEGA. In Voronkov, Andrei, (ed.), *Proceedings of the 19th International Conference on Automated Deduction (CADE-19)*, number 2392 in LNAI, pages 144–149, Copenhagen, Denmark. Springer.

[Siekmann *et al*, 2002b] Siekmann, Jörg, Benzmüller, Christoph, Fiedler, Armin, Meier, Andreas and Pollet, Martin. (2002b). Proof development with OMEGA: $\sqrt{2}$ is irrational. In Baaz, Matthias and Voronkov, Andrei, (eds.), *Logic for Programming, Artificial Intelligence, and Reasoning, 9th International Conference, LPAR 2002*, number 2514 in LNAI, pages 367–387. Springer.

[Smullyan, 1968] Smullyan, R. M. (1968). *First-Order Logic*, volume 43 of *Ergebnisse der Mathematik*. Springer-Verlag, Berlin.

[Snyder & Gallier, 1989] Snyder, Wayne and Gallier, Jean. (July/August 1989). Higher-order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation*, 8(2):101–140.

[Staples, 1995] Staples, Mark. (September 1995). Window inference in isabelle. In Paulson, Larry, (ed.), *Proceedings of the First Isabelle Users Workshop, Cambridge, UK*.

[Stuber, 1996] Stuber, Jürgen. (1996). Superposition theorem proving for abelian groups represented as integer modules. In Ganzinger, Harald, (ed.), *Rewriting techniques and applications: Intern. conference (RTA-7): New Brunswick, NJ, USA, July 27-30, 1996; proceedings*, volume LNCS 1103, pages 33–47S., Berlin. Springer.

[Tarski, 1936] Tarski, Alfred. (1936). Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophia*, 1:261–405.

[Turing, 1937] Turing, Alan. (1937). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265. **43**:544-546.

[Von Neumann, 1928] Von Neumann, John. (1928). Die Axiomatisierung der Megenlehre. *Mathematische Zeitschrift*, 27:669–752.

[Wallen, 1990] Wallen, Lincoln. (1990). *Automated proof search in non-classical logics: efficient matrix proof methods for modal and intuitionistic logics*. MIT Press series in artificial intelligence.

[Walther, 1987] Walther, Christoph. (1987). *A many-sorted calculus based on resolution and paramodulation*. Research notes in artificial intelligence. Morgan Kaufmann.

[Weidenbach, 1999] Weidenbach, Christoph. (1999). SPASS: Combining Superposition, Sorts and Splitting. In Robinson, A. and Voronkov, A., (eds.), *Handbook of Automated Reasoning*. Elsevier.

[Whitehead & Russell, 1910] Whitehead, Alfred North and Russell, Bertrand. (1910). *Principia Mathematica*, volume I. Cambridge University Press, Cambridge, Great Britain, second edition.

[Wolfram, 1993] Wolfram, David A. (1993). *The Clausal Theory of Types*, volume 2 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

[Zermelo, 1908] Zermelo, Ernst. (1908). Untersuchungen über die Grundlagen der Mengenlehre. *Mathematische Annalen*, 65:261–281.

Appendix

Appendix A

Completeness Proof

Theorem 5.4.1 (Completeness) Let φ be an $\mathcal{L}$-formula, Q an initial indexed formula tree for φ^+, let R be an initial FVIF-tree for Q, and Id the empty substitution. If φ is $\mathcal{L}$-valid then there is a CoRe derivation

$$[Q, Id \rhd_{\mathcal{L}} R] \longmapsto^* [Q', \sigma \rhd_{\mathcal{L}} \stackrel{+}{\_} True]$$

■

Proof. *The completeness proof relies on the soundness and completeness results of Theorem 4.12.1 which is due to [Wallen, 1990, Andrews, 1989, Pfenning, 1987]. The proof sketch is as follows: from Theorem 4.12.1 we assume that we have guessed the right multiplicities for γ- and ν-type nodes, the right combined substitution σ, the necessary introductions of Leibniz' equality, extensionality introductions, boolean ζ-expansions, cut, and have moved any $\Box$ and $\Diamond$-quantifier in front of literal nodes using the structural modal permutation rule. All paths in the resulting FVIF-tree R_P are (propositionally) $\mathcal{L}$-unsatisfiable. That is from $[Q, Id \rhd_{\mathcal{L}} R]$ we can derive a proof state $[Q_P, \sigma \rhd_{\mathcal{L}} R_P]$. In a second phase we have to prove that from $[Q_P, \sigma \rhd_{\mathcal{L}} R_P]$ we can derive $[Q_P, \sigma \rhd_{\mathcal{L}} \stackrel{+}{\_} True]$. The problem $[Q_P, \sigma \rhd_{\mathcal{L}} R_P]$ is essentially propositional, since all necessary substitutions have already been applied. In this second phase we prove that the combination of the contraction rule, resolution replacement rule application, and simplification allows us to simulate the* path resolution *rule from [Murray & Rosenthal, 1987a]. Since* path resolution *is complete the* CoRe *calculus is also complete. However, while path resolution derives an empty subgraph, we show that in* CoRe *we obtain the final proof state $[Q_P, \sigma \rhd_{\mathcal{L}} \stackrel{+}{\_} True]$.*

The first part of the proof is straightforward since the CoRe *calculus provides all necessary rules*

- *to eliminate positive equivalences and equalities on booleans using the boolean ζ-expansion rule,*
- *to increase the multiplicities, apply substitutions, introduce Leibniz' equalities and extensionalities like in [Pfenning, 1987] by combining the* CoRe *extensionality introduction rule with the cut rule (cf. Lemma 4.9.1), and*
- *to safely group all modal quantifiers that occur in the FVIF-tree around the literal nodes using the structural modal permutation rule (Definition 5.3.9).*

Thus, we can derive $[Q_P, \sigma \rhd_{\mathcal{L}} R_P]$ and it remains to prove in the second phase that the combination of the contraction rule, resolution replacement rule application, and simplification allows us to simulate path resolution. In order to motivate this we compare how path resolution transforms the

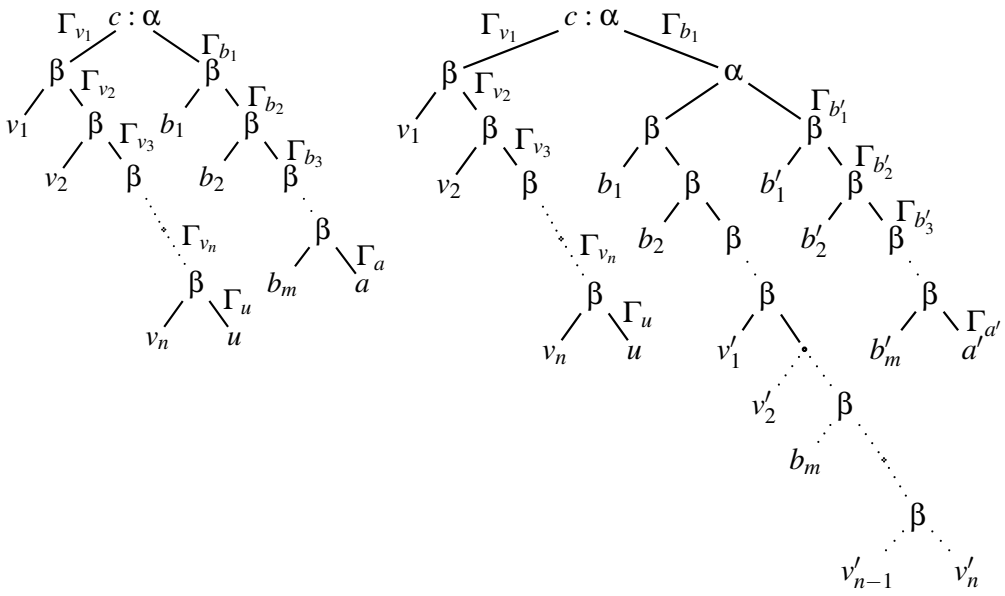
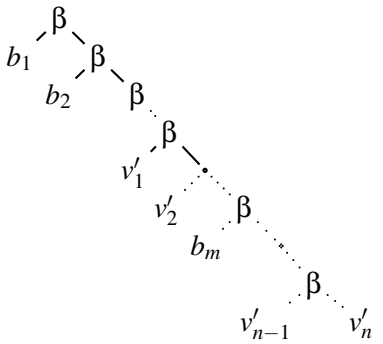


Figure A.1: Subtree before and after path resolution on the connectable nodes u and a.

FVIF-tree to how the CoRe *calculus rule transform the FVIF-tree. Consider the subtree of the actual overall FVIF-tree in which a replacement rule shall be applied. This situation is shown on the left hand side of Figure A.1.*

In this subtree u and a are connectable subtrees. The path resolution step derives the resolvent



which is then α-inserted on the top-node c, which results in the new subtree viewed on the right hand side of Figure A.1.

In Figure 5.5 on page 76 the respective replacement rule is applied in the same situation. Comparing the resulting trees it is obvious that the subtree containing a is transformed into the resolvent obtained during path resolution, except for three things:firstly, there is an additional subtree Proved. Secondly, the α-related Γ_{b_i} are preserved, while in the resolvent of path resolution they are removed. Thirdly, the old version of this subtree is missing in comparison to the situation after path resolution.

We now show that the FVIF-tree obtained by path resolution can be obtained by replacement rule application in combination with weakening, contraction, and simplification.

– Removing Context: *The Γ parts in the subtree are the α-related parts to the b_i that remain in*

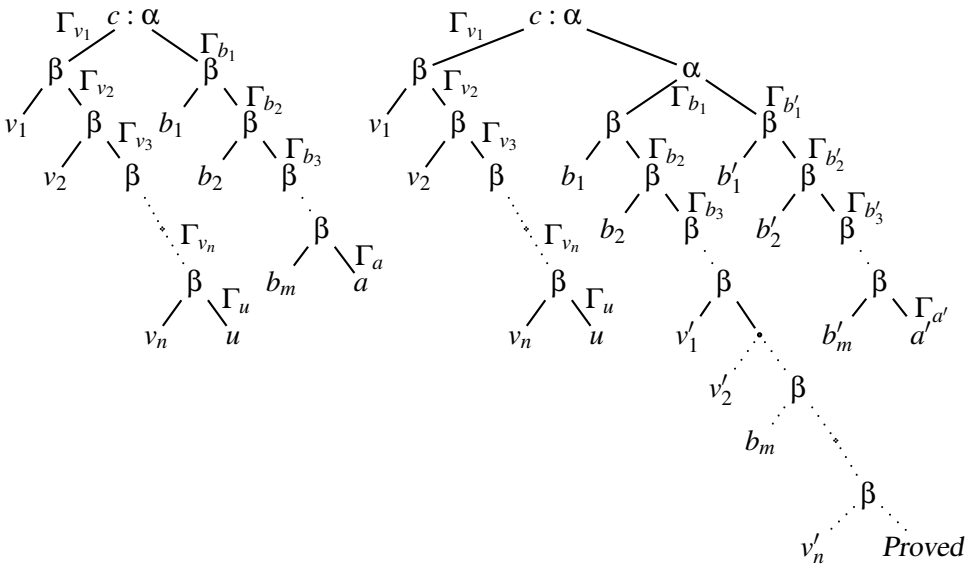


Figure A.2: Structures of the FVIF-tree before and after copying and the rule application.

---

place during the replacement rule application. Those can be removed by applying the weakening rule (cf. Definition 5.3.6) on the connecting α*-type nodes.*

– Removing *Proved*: *The subtree Proved can be deleted by applying the simplification rule on the right-hand side of the last* β*-type node that governs Proved. If there is no such* β*-type node, then the whole subtree rooted on c can be simplified to Proved by the simplification rule.*

– Missing subtree: *The subtree containing a can be preserved by applying the contraction rule on its root node (immediately below c), and applying the replacement rule afterwards. This results in the subtree shown on the right-hand side of Figure A.2.*

The rest of the proof consists on proving formally that indeed the resolvent obtained by path resolution has the shape as sketched before.

In [Murray & Rosenthal, 1987a] the resolvent of a path resolution step is given by a function WS computing the weak split graph. *This function takes a subgraph H that contains only the connection partners and the whole graph G. The definition of the function* $WS(H,G)$ *is given in Figure A.3 (p. 212).*

Translating the notation used in [Murray & Rosenthal, 1987a] to our setting, the so-called c-arcs $(X,Y)_c$ *correspond to nodes of primary type* α *and the so-called* d-arcs $(X,Y)_d$ *to nodes of primary type* β*. Finally,* H_X *denotes the restriction of H to the nodes contained in X. Furthermore, the disjunction* ∨ *used to connect the graphs corresponds to nodes of primary type* β*. Exploiting this relationship allows to define WS for FVIF-trees. We call this function* WS_t *and it takes as argument the set C of nodes that form the connections and a FVIF-tree R. Note that for sake of completeness we also define it for nodes of primary type* ν *and* π*.*

$$
\begin{array}{rcll}
WS(\emptyset, G) & = & G & \\
WS(G, G) & = & \emptyset & \\
WS(H, G) & = & WS(H_X, X) \vee WS(H_Y, Y) & \text{if } G = (X,Y)_d \\
WS(H, G) & = & WS(H_X, X) \vee WS(H_Y, Y) & \text{if } G = (X,Y)_c \text{ and } \\
 & & & H \text{ meets both } X \text{ and } Y \\
WS(H, G) & = & WS(H, X) & \text{if } G = (X,Y)_c \text{ and } \\
 & & & H \text{ is contained in } X
\end{array}
$$

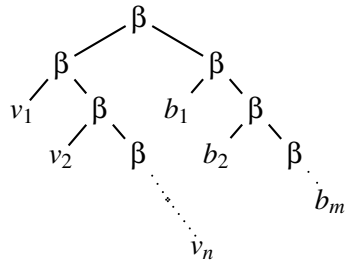
Figure A.3: The Function *WS* that computes the *weak split graph*.

$$
\begin{array}{rcll}
WS_t(\emptyset, R) & = & R & \\
WS_t(\{R\}, R^p) & = & \mathit{Proved}^p & \\
WS_t(C, \beta^p(R_1, R_2)) & = & \underline{\beta^p}(WS_t(C_{R_1}, R_1), WS_t(C_{R_2}, R_2)) & \\
WS_t(C, \alpha^p(R_1, R_2)) & = & \underline{\beta^p}(WS_t(C_{R_1}, R_1), WS_t(C_{R_2}, R_2)) & \textit{if } C \textit{ meets} \\
 & & & \textit{both } R_1 \textit{ and } R_2 \\
WS_t(C, \alpha^p(R_1, R_2))) & = & WS_t(C, R_1) & \textit{if } C \textit{ is cont-} \\
 & & & \textit{ained in } R_1 \\
WS_t(C, \nu^p(R)) & = & \underline{\nu^p}(WS_t(C, R)) & \\
WS_t(C, \pi^p(R)) & = & \underline{\pi^p}(WS_t(C, R)) &
\end{array}
$$

where $\underline{\nu^p}(R) = \mathit{Proved}$ *if* $R = \mathit{Proved}$*, and otherwise* $\underline{\nu^p}(R) = \nu^p(R)$*, and analogously for* $\underline{\pi^p}$*. Furthermore* $\underline{\beta^p}(R, R')$ *is defined as follows:*

$$
\underline{\beta^p}(R, R') = \begin{cases}
\mathit{Proved}^p & \textit{if } R = R' = \mathit{Proved} \\
R & \textit{if } R' = \mathit{Proved} \\
R' & \textit{if } R = \mathit{Proved} \\
\beta^p(R, R') & \textit{otherwise}
\end{cases}
$$

It is easy to see that applying WS_t *on the connecting* α*-type node* c *in the original FVIF-tree in Figure A.1 with respect to the connection* u *and* a *results in the following FVIF-tree:*



Obviously this resolvent computed by WS_t *is equivalent to the resolvent presented before, which concludes the proof.* □

Appendix B

Sample CoRe Window Proofs

B.1 Proof of $\mathbf{p}_{o\to o}(\mathbf{a}_o \wedge \mathbf{b}_o) \Rightarrow \mathbf{p}(\mathbf{b} \wedge \mathbf{a})$

The initial proof node for the positive formula $(p_{o\to o}(a_o \wedge b_o) \Rightarrow p(b \wedge a))^+$ is

Label	WPState	Window	Justification/Abstraction/Refinement
L_0	$WPS_1 \vdash$	$p_{o\to o}(a_o \wedge b_o) \Rightarrow p(b \wedge a)$	

In order to ease readability, we agree to omit in the presentation of that proof the necessary *Adapt-Window-Proof-State*-justifications. The idea for the proof of that theorem is to use the formula $a \wedge b = b \wedge a$ as a lemma to reduce $p(b \wedge a)$ to $p(a \wedge b)$, and subsequently use the negative occurrence of that formula to finish the proof.

The lemma is introduced by window cut, which transforms the FVIF-tree into

$$(\underbrace{((a \wedge b) = (b \wedge a))^+}_{L_{10}} \vee \underbrace{(p(a \wedge b) \Rightarrow p(b \wedge a)^+)}_{L_{20}}) \wedge (\underbrace{((a \wedge b) = (b \wedge a))^-}_{L_{30}} \Rightarrow \underbrace{(p(a \wedge b) \Rightarrow p(b \wedge a)^+)}_{L_{40}})$$

Thereby the window structure is inherited to the new occurrences of the old goal formula (L_{20}, L_{40}), and new windows are introduced for the occurrences of the lemma, on the one hand where it is a new assumption (L_{30}), and on the other hand where it is an additional alternative proof obligation (L_{10}). This results in the window proof nodes

Label	WPState	Window	Justification/Abstraction/Refinement
L_{10}	$WPS_2 \vdash$	$((a \wedge b) = (b \wedge a))^+$	
L_{20}	$WPS_2 \vdash$	$p(a \wedge b) \Rightarrow p(b \wedge a)^+$	
L_{30}	$WPS_2 \vdash$	$((a \wedge b) = (b \wedge a))^-$	
L_{40}	$WPS_2 \vdash$	$p(a \wedge b) \Rightarrow p(b \wedge a)^+$	

We now consider the two cases $((a \wedge b) = (b \wedge a) \vee p(a \wedge b) \Rightarrow p(b \wedge a))^+$ (L_{10} and L_{20}) and $((a \wedge b) = (b \wedge a) \Rightarrow p(a \wedge b) \Rightarrow p(b \wedge a))^+$ (L_{30} and L_{40}):

1. L_{10} and L_{20}: first we remove the proof node L_{20} by weakening. On L_{10} we proceed by applying the ζ-expansion rule, which results in

Label	WPState	Window	Justification/Abstraction/Refinement
L_{11}	$WPS_3 \vdash$	$((a \wedge b) \Rightarrow \underbrace{(b \wedge a)}_{(A)}) \wedge ((b \wedge a) \Rightarrow \underbrace{(a \wedge b)}_{(B)})^+$	

There we obtain for both (A) and (B) the replacement rules $a^- \rightarrow \langle True^+ \rangle$ and $b^- \rightarrow \langle True^+ \rangle$ from the logical context of those subformulas. The successive application of these rules results in

Label	WPState	Window	Justification/Abstraction/Refinement
L_{15}	$WPS_4 \vdash$	$(a \wedge b) \Rightarrow (True \wedge True) \wedge ((b \wedge a) \Rightarrow (True \wedge True))^+$	

which can be simplified to

Label	WPState	Window	Justification/Abstraction/Refinement
L_{26}	$WPS_4 \vdash$	$True$	

2. L_{30} and L_{40}: we use the rewriting replacement rule $a \wedge b \rightarrow \langle b \wedge a \rangle$ from L_{30} on L_{40} to obtain

Label	WPState	Window	Justification/Abstraction/Refinement
L_{41}	$WPS \vdash$	$p(a \wedge b) \Rightarrow p(a \wedge b)^+$	

For the positive subformula $p(a \wedge b)$ we obtain the expected resolution replacement rule $p(a \wedge b)^- \rightarrow \langle True^+ \rangle$, whose application reduces L_{41} to

Label	WPState	Window	Justification/Abstraction/Refinement
L_{42}	$WPS \vdash$	$p(a \wedge b) \Rightarrow True^+$	

After closing the subwindows and a final simplification we obtain

Label	WPState	Window	Justification/Abstraction/Refinement
L_6	$WPS_{14} \vdash$	$True^+$	

Using the representation of CORE window proofs from Chapter 8 the complete CORE window proof is then:

Label	WPState	Window	Justification/Abstraction/Refinement
L_0	$WPS_1 \vdash$	$p_{o \rightarrow o}(a_o \wedge b_o) \Rightarrow p(b \wedge a)$	$\text{Cut}(a \wedge b = b \wedge a) : L_{10}, L_{20}, L_{30}, L_{40}$

We use $(a \wedge b) = (b \wedge a)$ as a lemma which we introduce by Cut.

1. Subgoal $((a \wedge b) = (b \wedge a) \vee p(a \wedge b) \Rightarrow p(b \wedge a))^+$

L_{10}	$WPS_2 \vdash$	$(a \wedge b) = (b \wedge a)^+$	ζ-Expansion : L_{11}
L_{11}	$WPS_3 \vdash$	$((a \wedge b) \Rightarrow (b \wedge a)) \wedge ((b \wedge a) \Rightarrow (a \wedge b))^+$	Apply $(b \to \langle True \rangle)$: L_{12}
L_{12}	$WPS_4 \vdash$	$(a \wedge b) \Rightarrow (True \wedge a) \wedge ((b \wedge a) \Rightarrow (a \wedge b))^+$	Apply $(a \to \langle True \rangle)$: L_{13}
L_{13}	$WPS_4 \vdash$	$(a \wedge b) \Rightarrow (True \wedge True) \wedge ((b \wedge a) \Rightarrow (a \wedge b))^+$	Apply $(a \to \langle True \rangle)$: L_{14}
L_{14}	$WPS_4 \vdash$	$(a \wedge b) \Rightarrow (True \wedge True) \wedge ((b \wedge a) \Rightarrow (True \wedge b))^+$	Apply $(b \to \langle True \rangle)$: L_{15}
L_{15}	$WPS_4 \vdash$	$(a \wedge b) \Rightarrow (True \wedge True) \wedge ((b \wedge a) \Rightarrow (True \wedge True))^+$	Simplify : L_{26}
L_{26}	$WPS_4 \vdash$	$True$	Close : L_5
L_{20}	$WPS \vdash$	$p(a \wedge b) \Rightarrow p(b \wedge a)^+$	Weakening

2. Subgoal $((a \wedge b) = (b \wedge a) \Rightarrow p(a \wedge b) \Rightarrow p(b \wedge a))^+$

L_{30}	$WPS_2 \vdash$	$((a \wedge b) = (b \wedge a))^-$	Close : L_5
L_{40}	$WPS_2 \vdash$	$p(a \wedge b) \Rightarrow p(b \wedge a)^+$	Apply $(a \wedge b \to \langle b \wedge a \rangle)$: L_{41}
L_{41}	$WPS \vdash$	$p(a \wedge b) \Rightarrow p(a \wedge b)^+$	Apply $(p(a \wedge b) \to \langle True \rangle)$: L_{42}
L_{42}	$WPS \vdash$	$p(a \wedge b) \Rightarrow True^+$	Simplify : L_{43}
L_{43}	$WPS \vdash$	$True^+$	Close : L_5
L_5	$WPS \vdash$	$True \wedge ((a \wedge b) = (b \wedge a)) \Rightarrow True^+$	Simplify : L_6
L_6	$WPS_{14} \vdash$	$True^+$	Axiom

B.2 Proof of $\forall \mathbf{p}_{o\to o} . \lambda \mathbf{x} . \mathbf{p}(\mathbf{p}(\mathbf{p}(\mathbf{x}))) = \lambda \mathbf{x} . \mathbf{p}(\mathbf{x})$

The proof of the theorem $\forall p_{o\to o} . \lambda x . p(p(p(x))) = \lambda x . p(x)$ is performed by case analysis over $x = True$, and the values of $p(True)$ and $p(False)$. The axioms usd in that proof are given on p. 169. The initial window proof node is

Label	WPState	Window	Justification/Abstraction/Refinement
L_0	$\vdash$	$\lambda x . p(p(p(x))) = \lambda x . p(x)^+$	

By using the lemma (11.4) we obtain

Label	WPState	Window	Justification/Abstraction/Refinement
L_1	$\vdash$	$p(p(p(x))) = p(x)^+$	

We perform the first case analysis over $x = True$ by using the cut rule which results in

Label	WPState	Window	Justification/Abstraction/Refinement
L_{20}	$\vdash$	$x = True^-$	
L_{30}	$\vdash$	$p(p(p(x))) = p(x)^+$	
L_{40}	$\vdash$	$\neg(x = True)^-$	
L_{50}	$\vdash$	$p(p(p(x))) = p(x)^+$	

Thereby L_{20} and L_{30} occur in the same logical context as well as L_{40} and L_{50}.

– Case L_{20}, L_{30}: to L_{30} we apply the rewriting replacement rule $x \rightarrow \langle True \rangle$ obtained from L_{20} and obtain

Label	WPState	Window	Justification/Abstraction/Refinement
L_{31}	$\vdash$	$p(p(p(True))) = p(True)^+$	

On that proof node we perform a case analysis over $p(True) = True$ using the cut rule, which results in

Label	WPState	Window	Justification/Abstraction/Refinement
L_{320}	$\vdash$	$p(True) = True^-$	
L_{330}	$\vdash$	$p(p(p(True))) = p(True)^+$	
L_{340}	$\vdash$	$\neg(p(True) = True)^-$	
L_{350}	$\vdash$	$p(p(p(True))) = p(True)^+$	

As before, on the one hand L_{320} and L_{330} occur in a same logical context, and on the other hand L_{340} and L_{350}. Note for both subgoals L_{20} is also in the same logical context:

1. Case L_{320} and L_{330}: to L_{330} we apply four times the rewriting replacement rule from L_{320}, which reduces L_{330} to

Label	WPState	Window	Justification/Abstraction/Refinement
L_{331}	$\vdash$	$True = True^+$	

2. Case L_{340} and L_{350}: to L_{340} we apply the resolution replacement rule resulting from (11.3) to obtain

Label	WPState	Window	Justification/Abstraction/Refinement
L_{341}	$\vdash$	$p(True) = False^-$	

From that proof node we obtain the rewriting replacement rule $p(True) \rightarrow \langle False \rangle$, which we apply to L_{350} and obtain:

Label	WPState	Window	Justification/Abstraction/Refinement
L_{351}	$\vdash$	$p(p(False)) = False$	

For that proof node we perform the final case analysis over $p(False) = True$ using the cut rule, which results in

Label	WPState	Window	Justification/Abstraction/Refinement
L_{3520}	$\vdash$	$p(False) = True^-$	
L_{3530}	$\vdash$	$p(p(False)) = False$	
L_{3531}	$\vdash$	$False = False^+$	
L_{3532}	$\vdash$	$True^+$	

(a) Case L_{3520}, L_{3530}: by application of the rewriting replacement rules from L_{3520} and L_{341} on L_{3530} we obtain

Label	WPState	Window	Justification/Abstraction/Refinement
L_{3531}	$\vdash$	$False = False^{+}$	

which is trivially provable by simplification.

(b) Case L_{3540}, L_{3550}: by application of (11.3) on L_{3540} we obtain

Label	WPState	Window	Justification/Abstraction/Refinement
L_{3541}	$\vdash$	$p(False) = False^{-}$	

To L_{3550} we apply twice the rewriting replacement rule from L_{3541} to obtain

Label	WPState	Window	Justification/Abstraction/Refinement
L_{3551}	$\vdash$	$False = False^{+}$	

which is also trivially provable by simplification.

– Case L_{40}, L_{50}: the proof of this case is analogously to the other by using the resolution replacement rule from (11.3), and case analysis over $p(False) = True$ and $p(True) = True$.

The complete CoRE window proof for that theorem is presented below:

Label	WPState	Window	Justification/Abstraction/Refinement
L_0	$\vdash$	$\lambda x \,.\, p(p(p(x))) = \lambda x \,.\, p(x)^{+}$	Apply $(11.4) : L_1$
L_1	$\vdash$	$p(p(p(x))) = p(x)^{+}$	Cut$(x = True) : L_{20}, L_{30}, L_{40}, L_{50}$

1. Subgoal $(x = \boldsymbol{True} \Rightarrow p(p(p(x))) = p(x))^{+}$

L_{20}	$\vdash$	$x = True^{-}$	
L_{30}	$\vdash$	$p(p(p(x))) = p(x)^{+}$	Apply $(L_{20}) : L_{31}$
L_{31}	$\vdash$	$p(p(p(True))) = p(True)^{+}$	Cut$(p(True) = True) :$ $L_{320}, L_{330}, L_{340}, L_{350}$

A. Subgoal $(p(\boldsymbol{True}) = \boldsymbol{True} \Rightarrow p(p(p(\boldsymbol{True}))) = p(\boldsymbol{True}))^{+}$

L_{320}	$\vdash$	$p(True) = True^{-}$	Close : L_{36}
L_{330}	$\vdash$	$p(p(p(True))) = p(True)^{+}$	$4 \times$ Apply $(L_{320}) : L_{331}$
L_{331}	$\vdash$	$True = True^{+}$	Simplify : L_{332}
L_{332}	$\vdash$	$True^{+}$	Close : L_{36}

B. Subgoal $(\neg(p(\boldsymbol{True}) = \boldsymbol{True}) \Rightarrow p(p(p(\boldsymbol{True}))) = p(\boldsymbol{True}))^{+}$

L_{340}	$\vdash$	$\neg(p(True) = True)^{-}$	Apply $(11.3) : L_{341}$
L_{341}	$\vdash$	$p(True) = False^{-}$	Close : L_{36}
L_{350}	$\vdash$	$p(p(p(True))) = p(True)^{+}$	Apply $(L_{341}) : L_{351}$
L_{351}	$\vdash$	$p(p(False)) = False$	Cut$(p(False) = True) :$ $L_{3520}, L_{3530}, L_{3540}, L_{3550}$

i. Subgoal $(p(\textbf{\textit{False}}) = \textbf{\textit{True}} \Rightarrow p(p(\textbf{\textit{False}})) = \textbf{\textit{False}})^+$

L_{3520}	$\vdash$	$p(False) = True^-$	
L_{3530}	$\vdash$	$p(p(False)) = False^+$	Apply $(L_{3520}, L_{341}) : L_{3531}$
L_{3531}	$\vdash$	$False = False^+$	Simplify : L_{3532}
L_{3532}	$\vdash$	$True^+$	

ii. Subgoal $(\neg(p(\textbf{\textit{False}}) = \textbf{\textit{True}}) \Rightarrow p(p(\textbf{\textit{False}})) = \textbf{\textit{False}})^+$

L_{3540}	$\vdash$	$\neg(p(False) = True)^-$	Apply $(11.3) : L_{3541}$
L_{3541}	$\vdash$	$p(False) = False^-$	
L_{3550}	$\vdash$	$p(p(False)) = False$	$2 \times$ Apply $(L_{3541}) : L_{3551}$
L_{3551}	$\vdash$	$False = False^+$	Simplify : L_{3552}
L_{3552}	$\vdash$	$True^+$	
L_{356}	$\vdash$	$(p(False) = True \Rightarrow True) \wedge (p(False) = False \Rightarrow True)^+$	Simplify : L_{357}
L_{357}	$\vdash$	$True^+$	Close : L_{36}
L_{36}	$\vdash$	$(p(True) = True \Rightarrow True) \wedge (p(True) = False \Rightarrow True)^+$	Simplify : L_{37}
L_{37}	$\vdash$	$True^+$	

2. Subgoal $(x = \textbf{\textit{True}} \vee p(p(p(x))) = p(x))^+$

L_{40}	$\vdash$	$\neg(x = True)^-$	Apply $(11.3) : L_{41}$
L_{41}	$\vdash$	$x = False^-$	
L_{50}	$\vdash$	$p(p(p(x))) = p(x)^+$	Apply $(L_{41}) : L_{51}$
L_{51}	$\vdash$	$p(p(p(False))) = p(False)^+$	Cut$(p(False) = True)$: $L_{520}, L_{530}, L_{540}, L_{550}$

A. Subgoal $(p(\textbf{\textit{False}}) = \textbf{\textit{True}} \Rightarrow p(p(p(\textbf{\textit{False}}))) = p(\textbf{\textit{False}}))^+$

L_{520}	$\vdash$	$\neg(p(False) = True)^-$	Apply $(11.3) : L_{521}$
L_{521}	$\vdash$	$p(False) = False^-$	Close : L_{56}
L_{530}	$\vdash$	$p(p(p(False))) = p(False)^+$	$4 \times$ Apply $(L_{520}) : L_{531}$
L_{531}	$\vdash$	$False = False^+$	Simplify : L_{532}
L_{532}	$\vdash$	$True^+$	Close : L_{56}

B. Subgoal $(p(\textbf{\textit{False}}) = \textbf{\textit{True}} \Rightarrow p(p(p(\textbf{\textit{False}}))) = p(\textbf{\textit{False}}))^+$

L_{540}	$\vdash$	$p(False) = True$	Close : L_{56}
L_{550}	$\vdash$	$p(p(p(False))) = p(False)^+$	Apply $(L_{540}) : L_{551}$
L_{551}	$\vdash$	$p(p(True)) = True^+$	Cut$(p(True) = True)$: $L_{5520}, L_{5530}, L_{5540}, L_{5550}$

i. Subgoal $(p(\textbf{\textit{True}}) = \textbf{\textit{True}} \Rightarrow p(p(\textbf{\textit{True}})) = \textbf{\textit{True}})^+$

L_{5520}	$\vdash$	$p(True) = True^-$	Close : L_{556}
L_{5530}	$\vdash$	$p(p(True)) = True$	$2 \times$ Apply $(L_{5520}) : L_{5531}$
L_{5531}	$\vdash$	$True = True^+$	Simplify : L_{5532}
L_{5532}	$\vdash$	$True^+$	Close : L_{556}

ii. Subgoal $(\neg(p(\textbf{\textit{True}}) = \textbf{\textit{True}}) \Rightarrow p(p(\textbf{\textit{True}})) = \textbf{\textit{True}}^+$

L_{5540}	$\vdash$	$\neg(p(True) = True)^-$	Apply $(11.3) : L_{5541}$
L_{5541}	$\vdash$	$p(True) = False^-$	Close : L_{556}
L_{5550}	$\vdash$	$p(p(True)) = True$	Apply $(L_{5541}, L_{540}) : L_{5551}$
L_{5551}	$\vdash$	$True = True^+$	Simplify : L_{5552}
L_{5552}	$\vdash$	$True^+$	Close : L_{556}
L_{556}	$\vdash$	$(p(True) = True \Rightarrow True) \wedge (p(True) = False \Rightarrow True)^+$	Simplify : L_{557}
L_{557}	$\vdash$	$True^+$	Close : L_{56}
L_{56}	$\vdash$	$(p(False) = False \Rightarrow True) \wedge (p(False) = True \Rightarrow True)^+$	Simplify : L_{57}
L_{57}	$\vdash$	$True^+$	
L_6	$\vdash$	$(x = True \Rightarrow True) \wedge (x = False) \Rightarrow True)^+$	Simplify : $L7$
L_7	$\vdash$	$True^+$	Axiom

B.3 Proof of the Irrationality of Square Root of 2

The theorem is $\neg(rat(\sqrt{2}))$ and the initial window proof node for the positive formula $\neg(rat(\sqrt{2}))^+$ is

Label	WPState	Window	Justification/Abstraction/Refinement
L_0	$\vdash$	$\neg(rat(\sqrt{2}))^+$	

The axioms used in that proof are given on p. 171. By focusing on $rat(\sqrt{2})$ in the initial window and subsequent application of (11.23) we obtain the following 4 windows, which are all in the same logical context:

Label	WPState	Window	Justification/Abstraction/Refinement
L_{20}	$\vdash$	$nat(n)^-$	
L_{30}	$\vdash$	$nat(m)^-$	
L_{40}	$\vdash$	$m \times \sqrt{2} = n^-$	
L_{50}	$\vdash$	$nat(D) \wedge cd(n,m,D)^+$	

To L_{40} we apply (11.27), (11.26), and (11.28) which results in the two window proof nodes:

Label	WPState	Window	Justification/Abstraction/Refinement
L_{430}	$\vdash$	$2 \geq 0^+$	
L_{440}	$\vdash$	$(m^2 \times 2 = n^2)^-$	

The proof of L_{430} is achieved by using (11.22), (11.15), and (11.16). To L_{440} we apply the contraction rule to duplicate that formula which results in

Label	WPState	Window	Justification/Abstraction/Refinement
L_{450}	$\vdash$	$(m^2 \times 2 = n^2)^-$	

$L_{460} \quad \vdash \quad (m^2 \times 2 = n^2)^-$

To L_{450} we apply (11.30), which results in the subgoals

Label	WPState	Window	Justification/Abstraction/Refinement
L_{451}	$\vdash$	$nat(n^2)^+$	
L_{452}	$\vdash$	$nat(m^2)^+$	
L_{453}	$\vdash$	$even(n^2)^-$	

The proofs for L_{451} and L_{452} are trivial using L_{20}, L_{30}, and (11.19). To L_{453} we apply (11.31) and subsequently (11.30) to obtain

Label	WPState	Window	Justification/Abstraction/Refinement
L_{4550}	$\vdash$	$nat(n)^+$	
L_{4560}	$\vdash$	$nat(m')^-$	
L_{4570}	$\vdash$	$n = m' \times 2^-$	

Again, L_{4550} and L_{4560} are trivially proven by L_{20} and L_{30}. Then we can apply L_{4570} to L_{60} which results in

Label	WPState	Window	Justification/Abstraction/Refinement
L_{461}	$\vdash$	$(m^2 \times 2 = (m' \times 2)^2)^-$	

After application of (11.26), (11.25), and (11.24) this proof node is reduced to

Label	WPState	Window	Justification/Abstraction/Refinement
L_{4640}	$\vdash$	$\neg(2 = 0)^+$	
L_{4650}	$\vdash$	$(m^2 = m'^2 \times 2)^-$	

The proof node L_{4640} is trivially proven by (11.22), (11.21), and (11.18). To L_{4650} we apply (11.30), to obtain

Label	WPState	Window	Justification/Abstraction/Refinement
L_{4660}	$\vdash$	$nat(m^2)^+$	
L_{4670}	$\vdash$	$nat(m'^2)^+$	
L_{4680}	$\vdash$	$even(m^2)^-$	

After application of (11.31) on L_{4680} and further application of (11.30) we obtain

Label	WPState	Window	Justification/Abstraction/Refinement
L_{4660}	$\vdash$	$nat(m^2)^+$	
L_{4670}	$\vdash$	$nat(m'^2)^+$	
L_{4682}	$\vdash$	$nat(m)^+$	
L_{4683}	$\vdash$	$nat(m'')^+$	
L_{4684}	$\vdash$	$m = m'' \times 2^-$	

Again, the proof nodes L_{4660}, L_{4670}, L_{4682}, and L_{4683} are trivially provable. The complete CoRe window proof for the irrationality of $\sqrt{2}$ is presented below. Note that, analogously to the detailed presentation above, we have omitted the detailed proofs for the trivial subgoals that arise during replacement rule application.

Label	WPState	Window	Justification/Abstraction/Refinement
L_0	$\vdash$	$\neg(rat(\sqrt{2}))^+$	Subwindow : L_1
L_1	$\vdash$	$rat(\sqrt{2})^-$	Apply (11.23) : $L_{20}, L_{30}, L_{40}, L_{50}$
L_{20}	$\vdash$	$nat(n)^-$	
L_{30}	$\vdash$	$nat(m)^-$	
A. Alternatives $nat(n)^-, nat(m)^-, m \times \sqrt{2} = n^-$			
L_{40}	$\vdash$	$m \times \sqrt{2} = n^-$	Apply (11.27) : L_{41}
L_{41}	$\vdash$	$((m \times \sqrt{2})^2 = n^2)^-$	Apply (11.26) : L_{42}
L_{42}	$\vdash$	$(m^2 \times \sqrt{2}^2 = n^2)^-$	Apply (11.28) : L_{430}, L_{440}
L_{430}	$\vdash$	$2 \geq 0^+$	...
L_{440}	$\vdash$	$(m^2 \times 2 = n^2)^-$	Contraction : L_{450}, L_{460}
i. First alternative $(m^2 \times 2 = n^2)^-$			
L_{450}	$\vdash$	$(m^2 \times 2 = n^2)^-$	Apply (11.30) : $L_{451}, L_{452}, L_{453}$
L_{451}	$\vdash$	$nat(n^2)^+$	
L_{452}	$\vdash$	$nat(m^2)^+$	
L_{453}	$\vdash$	$even(n^2)^-$	Apply (11.31) : L_{454}
L_{454}	$\vdash$	$even(n)^-$	Apply (11.30) : $L_{4550}, L_{4560}, L_{4570}$
L_{4550}	$\vdash$	$nat(n)^+$	Apply (L_{20}) :
L_{4560}	$\vdash$	$nat(m')^-$	
L_{4570}	$\vdash$	$n = m' \times 2^-$	
ii. Second alternative $(m^2 \times 2 = n^2)^-$			
L_{460}	$\vdash$	$(m^2 \times 2 = n^2)^-$	Apply (L_{4570}) : L_{461}
L_{461}	$\vdash$	$(m^2 \times 2 = (m' \times 2)^2)^-$	Apply (11.26) : L_{462}
L_{462}	$\vdash$	$(m^2 \times 2 = m'^2 \times 2^2)^-$	Apply (11.25) : L_{463}
L_{463}	$\vdash$	$(m^2 \times 2 = (m'^2 \times 2) \times 2)^-$	Apply (11.24) : L_{4640}, L_{4650}
L_{4640}	$\vdash$	$\neg(2 = 0)^-$	
L_{4650}	$\vdash$	$(m^2 = m'^2 \times 2)^-$	Apply (11.30) : $L_{4660}, L_{4670}, L_{4680}$
L_{4660}	$\vdash$	$nat(m^2)^+$	
L_{4670}	$\vdash$	$nat(m'^2)^+$	
L_{4680}	$\vdash$	$even(m^2)^-$	Apply (11.31) : L_{4681}
L_{4681}	$\vdash$	$even(m)^-$	Apply (11.30) : L_{4682}, L_{4683}
L_{4682}	$\vdash$	$nat(m)^+$	
L_{4683}	$\vdash$	$nat(m'')^+$	
L_{4684}	$\vdash$	$m = m'' \times 2^-$	
B. Alternative $nat(D) \wedge cd(n,m,D)^+$			
L_{50}	$\vdash$	$nat(D) \wedge cd(n,m,D)^+$	Subwindow : L_{51}, L_{52}

L_{51}	$\vdash$	$nat(D)^+$	Apply $([m'/Q_1]) : L_{510}$
L_{510}	$\vdash$	$nat(2)^+$	$Apply(11.22), 2 \times (11.18), (11.17)$
L_{52}	$\vdash$	$cd(n,m,D)^+$	Apply $(11.29) : L_{53}$
L_{53}	$\vdash$	$(nat(n) \wedge nat(m) \wedge nat(D) \wedge nat(Q_1) \wedge nat(Q_2) \wedge n = Q_1 \times D \wedge m = Q_2 \times D)^+$	Apply $([m'/Q_1, 2/D], L_{4670})$
L_{54}	$\vdash$	$(nat(n) \wedge nat(m) \wedge nat(2) \wedge nat(m') \wedge nat(Q_2) \wedge True \wedge m = Q_2 \times 2)^+$	Apply $([m''/Q_2], L_{4684}) : L_{55}$
L_{55}	$\vdash$	$(nat(n) \wedge nat(m) \wedge nat(2) \wedge nat(m') \wedge nat(m'') \wedge True \wedge True)^+$	Apply $(L_{20}, L_{30}, L_{4660}, L_{4683}) : L_{56}$
L_{56}	$\vdash$	$(True \wedge True \wedge nat(2) \wedge True \wedge True \wedge True \wedge True)^+$	Simplify : L_{57}
L_{57}	$\vdash$	$nat(2)^+$	$Apply(11.22), 2 \times (11.18), (11.17)$

B.4 Proof of $\exists \mathbf{x} \,.\, \Box(\Box\varphi(\mathbf{x}) \vee \psi(\mathbf{y})) \Leftrightarrow \Box\exists \mathbf{x} \,.\, (\Box\varphi(\mathbf{x}) \vee \psi(\mathbf{y}))$

The initial window proof state for the theorem is

Label	WPState	Window	Justification/Abstraction/Refinement
L_0	$\vdash$	$\exists x \,.\, \Box(\Box\varphi(x) \vee \psi(y)) \Leftrightarrow \Box\exists x' \,.\, (\Box\varphi(x') \vee \psi(y))$	

Note that the quantifiers are still inside the formula, as they are below the equivalence and thus cannot be eliminated. However, the expansion of the positive equivalence by the ζ-expansion rule removes the quantifiers. The complete CORE window proof is then:

Label	WPState	Window	Justification/Abstraction/Refinement
L_0	$\vdash$	$\exists x \,.\, \Box(\Box\varphi(x) \vee \psi(y)) \Leftrightarrow \Box\exists x' \,.\, (\Box\varphi(x') \vee \psi(y))$	ζ-Expansion : L_{01}
L_{01}	$\vdash$	$\Box(\Box\varphi(x) \vee \psi(y)) \Rightarrow \Box(\Box\varphi(X') \vee \psi(y))$ $\Box(\Box\varphi(x') \vee \psi(y)) \Rightarrow \Box(\Box\varphi(X) \vee \psi(y))$	Subwindow : L_1, L_2

A. Subgoal $\Box_P(\Box_{P'}\varphi(x) \vee \psi(y)) \Rightarrow \Box_c(\Box_{c'}\varphi(X') \vee \psi(y))$

L_1	$\vdash$	$\Box_P(\Box_{P'}\varphi(x) \vee \psi(y)) \Rightarrow \Box_c(\Box_{c'}\varphi(X') \vee \psi(y))$	Instantiate$([c/P, c'/P', x/X']) : L_{11}$
L_{11}	$\vdash$	$\Box_c(\Box_{c'}\varphi(x) \vee \psi(y)) \Rightarrow \Box_c(\Box_{c'}\varphi(x) \vee \psi(y))$	$\varphi(x) \rightarrow \langle True \rangle : L_{12}$
L_{12}	$\vdash$	$\Box_c(\Box_{c'}\varphi(x) \vee \psi(y)) \Rightarrow \Box_c(\Box_{c'}True \vee \psi(y))$	$\psi(y) \rightarrow \langle True \rangle : L_{13}$
L_{13}	$\vdash$	$\Box_c(\Box_{c'}\varphi(x) \vee \psi(y)) \Rightarrow \Box_c(\Box_{c'}True \vee True)$	Simplify : L_{14}
L_{14}	$\vdash$	$True$	Close : L_3

B. Subgoal $\Box_{P''}(\Box_{P'''}\varphi(x') \vee \psi(y)) \Rightarrow \Box_{c''}(\Box_{c'''}\varphi(x') \vee \psi(y))$

L_2	$\vdash$	$\Box_{P''}(\Box_{P'''}\varphi(x') \vee \psi(y)) \Rightarrow \Box_{c''}(\Box_{c'''}\varphi(X) \vee \psi(y))$	Instantiate$([c''/P'', c'''/P''', x'/X]) : L_{21}$
L_{21}	$\vdash$	$\Box_{c''}(\Box_{c'''}\varphi(x') \vee \psi(y)) \Rightarrow \Box_{c''}(\Box_{c'''}\varphi(x') \vee \psi(y))$	$\varphi(x') \rightarrow \langle True \rangle : L_{22}$

L_{22}	$\vdash$	$\Box_{c''}(\Box_{c'''}\varphi(x') \vee \psi(y))$ $\Box_{c''}(\Box_{c'''}\mathit{True} \vee \psi(y))$	$\Rightarrow$	$\psi(y) \rightarrow \langle \mathit{True} \rangle : L_{23}$
L_{23}	$\vdash$	$\Box_{c''}(\Box_{c'''}\varphi(x') \vee \psi(y))$ $\Box_{c''}(\Box_{c'''}\mathit{True} \vee \mathit{True})$	$\Rightarrow$	Simplify : L_{24}
L_{24}	$\vdash$	True		Close : L_3
L_3	$\vdash$	$\mathit{True} \wedge \mathit{True}$		Simplify : L_4
L_4	$\vdash$	True		Axiom

Index