Certification of First-order proofs in classical and intuitionistic logics [1]

Zakaria Chihani

August 21, 2015

[1]This work has been funded by the ERC Advanced Grant ProofCert.

Abstract

The field of automated reasoning contains a plethora of methods and tools, each with its own language and its community, often evolving separately. These tools express their proofs, or some proof evidence, in different formats such as resolution refutations, proof scripts, natural deductions, expansion trees, equational rewritings and many others. The disparity in formats reduces communication and trust between the different communities. Related efforts were deployed to fill the gaps in communication including libraries and languages bridging two or more tools.

This thesis proposes a novel approach at filling this gap for first-order classical and intuitionistic logics. Rather than translating proofs written in various languages to proofs written in one chosen language, this thesis introduces a framework for describing the semantics of a wide range of proof evidence languages through a relational specification, called Foundational Proof Certification (FPC). The description of the semantics of a language can then be appended to any proof evidence written in that language, forming a proof certificate, which allows a small kernel checker to verify them independently from the tools that created them. The use of semantics description for one language rather than proof translation from one language to another relieves one from the need to radically change the notion of proof.

Proof evidence, unlike complete proof, does not have to contain all details. Using proof reconstruction, a kernel checker can rebuild missing parts of the proof, allowing for compression and gain in storage space. Other desired aspects such as non-determinism, parallelism and flexibility in the description of a given proof evidence language are offered by the FPC framework.

Building on well-established proof theoretical foundations, namely focused proof systems, this thesis describes how proof certificates use a communication protocol to guide a kernel during proof checking. This protocol prevents the kernel from accepting false certificates, thereby ensuring soundness. Multiple examples, for classical and intuitionistic logics, are also presented for some proof formats as well as decision procedures.

This thesis is at the center of the multi-year ERC funded project ProofCert.

Le domaine du raisonnement automatique contient une multitude de mthodes et outils, chacun rassemblant une communaut autour d'un mme langage et voluant le plus souvent sparment, en utilisant d'autres langages. Ces outils expriment des preuves ou des lments de preuves, dans des formats varis tels que les rfutations par rsolution, les scripts de preuves, les dductions naturelles, les arbres d'expansion, la rcriture quationnelle, et beaucoup d'autres. L'existence de tant de moyens d'expression rduit la communication et la confiance entre les diffrentes communauts et de rcents efforts ont t dploys pour pallier ce manque de communication au moyen de bibliothques ou de traductions entre deux outils ou plus.

Cette thse propose une nouvelle approche pour combler ce manque de communication en ce qui concerne les logiques classique et intuitionniste du premier ordre. Au lieu de traduire les formats d'expression des diffrentes communauts en un seul langage choisi, cette thse introduit un cadre pour dcrire la smantique de ces diffrents langages travers des spcifications relationnelles, appel Foundational Proof Certification (FPC). La description de la smantique d'un certain langage peut alors tre accole des lments de preuves crits dans ce langage formant des certificats de preuve et permettant leur vrification au moyen d'un noyau unique, indpendamment des outils qui les ont gnrs. L'utilisation de ces descriptions de smantique au lieu de traductions des preuves d'un langage un autre permet de garder les notions originelles de preuve, connue des communauts respectives.

Les lments de preuves, contrairement aux preuves compltes, peuvent omettre des dtails. Le noyau de vrification tant capable de reconstruction de preuve, une grande compression et un gain en espace de stockage sont ainsi possibles. D'autres aspects, tels que le non-dterminisme, le paralllisme et une flexibilit dans la description de langage sont aussi permis dans le cadre de FPC.

En prenant pour base des fondements bien tudis de la thorie des preuves, en particulier les systmes de preuves focaliss, cette thse dcrit un protocole de communication que les certificats de preuves utilisent pour guider le noyau pendant la vrification. Ce protocole garantit la correction en s'assurant que le noyau n'accepte aucune formule invalide propose par un certificat. De multiples descriptions de formats et des procdures de dcision, en logiques classique et intuitionniste, sont prsentes en exemple.

Cette thse est au centre du projet ERC pluriannuel ProofCert.

Acknowledgement

I thank Gilles Dowek and Carsten Schuermann for kindly agreeing to serve as my rapporteurs. I also thank the reviewers for their patience and very useful comments. Dale Miller, Michael Filhol, Ali Assaf, Taus Brock-Nannestad, Rob Blanco, Kaustuv Chaudhuri, Danko Ilik and Ulysse Gérard.

Contents

1 Introduction **7**
1.1 Situating the thesis . . . 7
1.2 Scope of the thesis . . . 15

2 Preliminary notions on focusing **18**
2.1 Sequent calculi: From Gentzen to Girard . . . 18
2.1.1 Classical sequent calculus . . . 20
2.1.2 Intuitionistic sequent calculus . . . 21
2.2 Focused sequent calculi . . . 22
2.2.1 The tenets of focusing . . . 23
2.2.2 Focused classical sequent calculus *LKF* . . . 27
2.2.3 Intuitionistic sequent calculus LJF . . . 29

3 Global architecture **32**
3.1 Desired properties . . . 33
3.1.1 Poincaré Principle . . . 33
3.1.2 Modular integration . . . 33
3.1.3 Abstractions and typing . . . 34
3.1.4 Declarative and relational . . . 34
3.1.5 Proof reconstruction tools . . . 35
3.1.6 Parallelism . . . 35
3.1.7 Additional features . . . 35
3.2 Tailoring the framework . . . 36
3.3 Components of the framework . . . 36
3.3.1 The client's side . . . 39
3.3.2 The kernel checker's side . . . 39
3.3.3 The middle man: the semantics . . . 40
3.4 The augmented sequent calculus LKF^a . . . 40

3.4.1 Experts . . . 42
3.4.2 Clerks . . . 44
3.4.3 The complete LKF^a system . . . 45
3.5 Augmenting the *LJF* system . . . 47

4 Foundational Proof Certification 50
4.1 Descriptive semantics . . . 50
4.1.1 Polarity assignment . . . 51
4.1.2 Region delimitation . . . 53
4.1.3 Indexing . . . 54
4.1.4 Clerks & Experts . . . 54
4.2 Programmable semantics . . . 55
4.2.1 Type signature . . . 56
4.2.2 Predicate definitions . . . 60
4.3 Default teams of agents . . . 61
4.3.1 The `done` team . . . 61
4.3.2 The `oneOf` team . . . 62
4.3.3 The `witness` case . . . 62
4.3.4 The `tag` case . . . 63

5 FPC for classical logic 64
5.1 The *CNF* decision procedure . . . 64
5.1.1 Preliminaries . . . 65
5.1.2 *CNFdec* in the *FPC* framework . . . 65
5.2 Mating . . . 68
5.2.1 Mating's `p.r.i.c.e.` . . . 69
5.3 Resolution refutations . . . 72
5.3.1 Semantics of refutation sequences . . . 74
5.3.2 Semantics of a binary resolution step . . . 80
5.3.3 Interpreting a hyperresolution step . . . 89
5.4 Expansion Trees . . . 96
5.4.1 Expansion trees in *LK* . . . 98
5.4.2 Sequentialization of expansion trees to LKF^a . . . 100
5.4.3 Indexing . . . 102
5.4.4 Region delimitation . . . 102
5.4.5 Clerks and experts . . . 103

6 FPC for intuitionistic logic **106**
6.1 Mimic . . . 106
6.1.1 Initial in LJF^a . . . 106
6.1.2 Negative/positive alternation . . . 107
6.1.3 A `p.r.i.c.e.` for mimic . . . 110
6.1.4 Extending to first order . . . 112
6.2 λ-calculus . . . 113
6.2.1 `p.r.i.c.e.` for simply-typed η-long β-normal form λ-terms . . . 116
6.2.2 Simply-typed β-normal form λ-terms evidence . . . 117
6.2.3 Dependently-typed β-normal form λ-terms evidence . . . 119

7 Reasoning with equality **125**
7.1 Introduction . . . 125
7.2 Formalizing equality reasoning . . . 126
7.2.1 Smallest common unit . . . 127
7.2.2 Proof evidence and system's properties . . . 128
7.2.3 A `p.r.i.c.e.` definition for one-step rewrite . . . 129
7.3 Paramodulation . . . 130
7.3.1 Describing a paramodulation in LJF^a . . . 131
7.3.2 A `p.r.i.c.e.` for a paramodulation step . . . 133

8 Satellite interests **135**
8.1 Hosting kernels . . . 135
8.1.1 Introduction . . . 136
8.1.2 Mapping LKF sequents to LJF sequents . . . 137
8.1.3 Mapping LKF^a checking to LJF^a checking . . . 141
8.2 Transducing proof certificates . . . 143
8.2.1 Elaborating transducer . . . 144
8.2.2 Forgetful transducer . . . 146
8.3 Cooperating proof certificates . . . 149

9 Conclusion, related and future work **151**
9.1 Future work . . . 152
9.2 Related work . . . 155

Chapter 1

Introduction

Study the past if you would define the future.

- Confucius

What is the lifespan of a mathematical proof? The rhetorical character of this question is mostly caused by its lack of precision. If one asks users of a mechanized prover A: "what is the lifespan of a mathematical proof output by prover A?", their answer may very well be "until next version of A". If you ask users of a prover B the very same question, they may say "we have no way of knowing, B cannot read the output of A".

This thesis proposes a way to overcome this limitation of computer proofs, bringing them a few steps closer to the same immortality as that of a traditional mathematical proof. The first section of this introduction places the problematic of this thesis in a historical context by following the evolution of the notion of mathematical proofs and the birth of computer proofs. The second section determines the scope and goals of the thesis.

1.1 Situating the thesis

The word "proof" comes from the Latin *proba*, giving the verb *probare* with two meanings: *to demonstrate*, and *to test* or *to examine*. Today, both of these meanings are still accepted, along with many others, but the criteria for what constitutes a proof are different and depend on the context. A proof in a medical context is different from a proof in other sciences. In a court of law, a proof is understood yet another way: it is a demonstration "beyond reasonable doubt". It is unfortunate that the

words *proved* and *probable* stem from the same etymology because, in mathematics, demonstrate "beyond reasonable doubt" or "with a high probability" is not enough.

In all the above interpretations, a proof is an argument used to convince a third party of the validity of a statement. However, adding the word "mathematical" to the word "proof" places a higher burden *on* proof. To paraphrase Peter Andrews, a mathematical proof *establishes* truth [Andrews, 2002]; it does not merely indicate it, or strongly suggest it. A *correct* mathematical proof will survive the tides of time.

Several sciences, built on rigorous foundations with no room for doubt or acceptable error, gave birth to less strict counterparts dubbed *experimental*. Escaping the rigors of the original discipline, scientists in those experimental branches chose to embrace a small margin of error in exchange for ever growing knowledge.

Some people assert that mathematics is experiencing a similar divergence [Zeilberger and Andrews, 1994]. Indeed, there are members of the mathematical community that understand a proof as what is beyond reasonable doubt, be it for pragmatic reasons (see, for example Rabin [1976]) or because they are unable to do otherwise [Barwise, 1989].

The trade-off between fast answers with a low margin of error and slow perfectly sound answers must be considered in any real-world application. However, this thesis is grounded in the rigorous mathematics and proposes a new approach at giving to computer proofs the same immortality as the traditional ones. That immortality gave mathematics its rightful place among sciences and crowned mathematical reasoning above all others, because "proof – the traditional, Euclid-inspired, tightly knit chain of logical reasoning leading inexorably to a precise conclusion – is immortal", it is a "bulletproof means of verifying and confirming an assertion" [Krantz, 2011], and so should be a mechanized proof.

Evolution of Mathematics: a case for formal proofs

Celebrated by some, criticized by others, *formal proofs* never were, and will probably never be, unanimously accepted among mathematicians. This thesis takes no part in a century-old debate and adopts formal proofs as its starting point.

The basis of logical reasoning as inference, premise and conclusion was independently laid in many places. Only the Greek logic, however, permeated the centuries and came to dominate the western and Muslim worlds.

A graduate student today absorbs, in the span of a few years, logical notions that took millennia to evolve into their current body. Aristotle's syllogisms is consumed in the first few hours. Only years later can the student reflect on those few hours, on those first formalisms, and take them for what they were: the stuttering of yet

another child of philosophy, slowly maturing into science.

Aristotle's edifice, including the logic heritage, stood for centuries and extended to many areas of science. Its foundations were eventually shaken, in the 17th century, when the geocentric model, that he defended throughout his life, was shown to be erroneous. Francis Bacon vividly criticized Aristotle's philosophy in his Great Instauration saying :

> " After the sciences had been in several parts perhaps cultivated and handled diligently, there has risen up some man [Aristotle] of bold disposition, and famous for methods and short ways which people like, who has in appearance reduced them to an art, while he has in fact only spoiled all that the others had done. And yet this is what posterity likes, because it makes the work short and easy, and saves further inquiry, of which they are weary and impatient."

Several modern science areas were born from this revolution, but Logic as a science only came to being in the 19th century. And, while George Boole's Algebra of Logic was a stepping stone, it didn't denounce Aristotle's logic but rather attempted to formalize it and widen its applications. The real departure from Aristotle's logic was Gottlob Frege's Begriffsschrift (Concept Script), which was the birth of modern Logic. While the notions brought forth by Frege may seem mundane today, his work was the kite tempting the lightning and the apple falling from the tree.

Prominent figures of Logic followed, built on top of his work, discovered cracks in its foundations and repaired them, most famously Whitehead and Russell, that recognize their "chief debt" to Frege "in all questions of logical analysis" in the introduction of their monumental Principia Mathematica. It would be a lost cause to attempt a full description of the intricate phylogenetics of Logic, and to give credit everywhere credit is due. This brief historical overview is given only to spark interest and curiosity, to take a step back and wonder how, in the many centuries that separate Aristotle and Frege, no one thought of concepts as mundane as function-argument analysis, or as natural as unambiguous quantification, concepts that today seem so straightforward to an undergraduate student that their origin is not even stated. From that step back, we can imagine what would be the next simple idea, lurking in the shadows of the unknown, ready to bring about a new revolution.

Expansion of Mathematics: a case for mechanized proofs

At the dawn of mathematical reasoning, a proof had the "democratic virtue" that anyone could follow it. In Meno's dialogue, Aristotle was able to bring a boy with

no prior knowledge of mathematics to understand a mathematical argument. However, modern days have seen the complexity of certain mathematical proofs increase beyond the understanding of a single student.

Barendregt and Wiedijk [2005] put in correspondence known theorems with how their verification was carried out. Ranging from the proof of the irrationality of $\sqrt{2}$, verifiable by students, to Fermat's Last Theorem for which one has to be a specialist, to Kepler's Conjecture where the computer was needed. To answer the growing complexity of proofs, mechanized proofs are becoming an indispensable tool, both for academics and for industry. By expanding the realm of what is tractable, "the computer has already started doing to mathematics what the telescope and microscope did to astronomy and biology" [Zeilberger and Andrews, 1994].

But this evolution is also controversial. One of the most violent disagreements concerning mechanized proofs followed the paper by De Millo et al. [1979]. This paper was intended as a criticism of formal verification of programs, but also offered arguments against the general use of computers in mathematics. To their insistence on the "need for the mathematical consensus" when deciding the validity of a theorem, Leslie Lamport famously replied "I dont believe that the correctness of a theorem is to be decided by a general election", in a letter to the editor [Ashenhurst, 1979] that was mistakenly interpreted as a rebuttal to De Millo et al. [1979].

This sharp disagreement, covered and analyzed numerous times in literature (see [Asperti, 2012] for a survey or the book [MacKenzie, 2001]), is still poignant today. Some mathematicians, however, are bridging the great divide, some by learning how to employ computers to mechanize their proofs, most notably Thomas Hales and his team, and some by tentatively speaking up in favor of using computers in the mathematical field. In a blog post, Fields Medalist Timothy Gowers commented on the solution to the Erdos discrepancy problem using SAT solvers [Konev and Lisitsa, 2014] saying that he was "relaxed about huge computer proofs". Vladimir Voevodsky is another Fields Medalist on the side of computer assisted proofs, although he rejects complete automation of theorem proving. In a remarkable and recent paper, Martin [2015] describes a panel discussion at the 2014 ceremonies for the Breakthrough Prize where the winners discussed the usage of computer in proofs. These leading figures in today's mathematical world did not reject the use of computers in their work, quite the opposite. They perceived the computer as a natural ally in the pursuit of mathematical accomplishment.

In light of this change in position from leading mathematicians, one can hope that the great divide is closing and that the computer can finally be allowed to push mathematics towards the next generation of proofs.

Between art and necessity: a case for machine-generated proofs

Asperti [2012] discusses "the essence and purpose of proofs" in terms of what he adequately calls "message" and "certificate", closely related to the two meanings of the Latin root of "proof" given in the beginning of this introduction. His paper, is a call to "prevent the divorce between these two epistemological functions", and it is shared by the majority of mathematicians, including some of those that embrace computer-aided proofs. In other words, a difference is made between a computer *assisting* a mathematician in finding a proof and a computer actually *finding* a proof (via exhaustive search, heuristics or other techniques). Many critics focus on the lack of artistry in computer proofs, arguing that the beauty and simplicity of a proof are central. Defenders of computer *assistants* dispute the lack of beauty or artistry of a mechanized proof.

Advocating for *automated* theorem provers, at the 2015 Proof eXchange for Theorem Proving workshop's panel discussion on the subject of proof certificates, members of the panel expressed personal experiences where, after looking closely at a complicated resolution proof, they were able to find its key steps and transform it into a beautiful proof. A machine generated proof, they argued, can also be beautiful.

But is it outside the realm of possibilities that a machine-generated proof may be irrevocably inartistic? And if that proof plays a role in ensuring, say, that planes do not collide or that space rockets do not explode, would that *necessity* for it not be enough to make it *interesting*? Was the beauty of a proof at a discussion on proof *certificates* not a conflation of the "message" and "certificate" aspects of a proof?

To someone concerned with the art of advancing mathematical knowledge, a correct but unsurprising and inelegant proof of a predictable conjecture may be *less valuable*, in some ways, than an incorrect proof. If an incorrect proof brings new valid insights, widens understanding of the community and prompts the study of intermediary theorems that are reusable in other proofs, it can be considered valuable. Here, the "message" side of the proof is essential. To someone concerned with stability of auto-pilots on planes or safety of nuclear facilities, artistry is secondary, *i.e.,*the "certificate" side of the proof is essential.

It is an important aim to prevent the "divorce" of the two aspects of proof [Asperti, 2012], but preserving that marriage should not hinder the study of one aspect over the other. As Harvey Friedman points out: "just because a proof is explanatory doesn't mean it's certain, just because it is certain doesn't mean it's explanatory. They are two separate dimensions"[Mackenzie, 2005]

The work presented here does not discriminate on the basis of beauty, simplicity or any aspect of the "message" in a proof. The only states of interest a proof can be in are "checked" or "unchecked". The scope is, thus, restricted to machine-generated,

machine-checkable proofs. Whether a human can learn from the proof, or even read it, plays close to no role.

Henceforth, the terms "prover" and "tool" denote any computational logic system, automatic or interactive.

Dramatic consequences of small margins: a case for documented proofs

What is the lifespan of a machine-generated mathematical proof? This question is no longer rhetorical but its current answer is unsatisfactory. Indeed, mechanized proofs are often heavily dependent on the tool that produces them. What is then the value of proofs, generated with one tool, that hold no meaning in another? And that is far from being the worse scenario, as some tools are unable to replay a proof generated by a former version of themselves. Others do not even give a document justifying that they found a proof (let alone an independently checkable one), perhaps because it is too time consuming or because it would take too much space. Consider that there is no doubt that Fermat was a genius, but brilliant as he was, to what extent can one have faith in the claim "I have a truly marvelous demonstration of this proposition which this margin is too narrow to contain"? What is the value of such a claim compared to the value of the actual proof of that claim, 358 years later, even if producing that proof was both time consuming and took a lot of space?

A mathematical proof is, first and foremost, a proof in the original sense: a demonstration, destined to be probed, tested, examined and this thesis only considers provers that output such proofs.

Collaboration: a case for communicating systems

Mechanized provers have helped mathematicians cross the t's and dot the i's at a scale previously unattainable. Furthermore, a social process took place when large teams of mathematicians from different continents worked together on proofs of the same theorem. Such interactions, unprecedented in the history of mathematics, weakens the argument of lack of social process often used by those opposed to computer mathematics. One can simply subscribe to one of the many prover-related mailing-list to witness a thriving collaboration. However, with this great advancement came the fragmentation of the community, relative to a prover.

People often use a particular prover "simply because it was the dominant system at their particular location", as MacKenzie [2001, p.314] suggests. But even if the choice of a prover over another depends on an accident of academic birth, such a specialization can be a richness, provided the communication is restored.

One approach to do so is to translate outputs from one system to the other. Another approach is to translate all outputs to one language, an Esperanto for proofs, and check them in an independent checker. Efforts following this approach are described in the related works section.

A comparatively radical approach to bringing the community together is to have one language, one framework, one tool for all. QED [Mathematicians, 1994] is such a bold endeavor with the advantage of being close to the traditional mathematical language, thus preserving the idea of beauty of the proofs and the ability to learn from them. Unfortunately, it has also the inconvenience of not having materialized. Wiedijk [2007] offers an enlightening dissection of the (current) failure of QED, in which he states that he does not "believe that the QED system will consist of many different systems living peacefully together. One of the systems – hopefully the best one – will kill the others. That is how evolution works", provocatively calling the effort to improve communication between different systems a "lost energy".

It is fair to note that natural selection, in the sense of the theory of evolution, is more complex than that. The fittest rarely survives by killing other members of his species, or of any species. The ill-adapted members fail to evolve and simply halt. But even if it were the case in evolution, does the metaphor provide enough incentive to make QED the bottleneck of the provers' population? One is allowed to hope that all those interested in computer proofs will gladly abandon their preferred prover and converge around one single tool, but that hope seems rather Utopian. One can compare that claim to the following: "there will come a time when a programming language will kill all other languages, and all the programmers will use that one single language". Whether or not this will be the case remains to be seen. But even if it were the case, is that a desirable thing, when one thinks of the many technology transfers that occurred from one language to others *precisely* because that language was different from the others.

The diversification of proof systems resulted in numerous tools with different areas of expertise, sometimes very distant from one another, and the colonization of more and more mathematical terrain. And that "speciation" is arguably important: for example, the techniques used for algebraic topology proofs and for cache coherence protocols are different. Seeing QED as a genetic bottleneck that would sacrifice all the systems on the altar of conformity may result in the desertion of areas of mathematics (for lack of an adapted tool) or worse, the non discovery of currently unknown worlds. A rich ecosystem of evolving provers, bringing fresh ideas and preserving diversity, may be a better path to a higher level of mathematical knowledge.

From that point of view, striving to improve communication between systems is energy well spent.

Universality: a case for machine-checked proofs

Many automated and interactive theorem provers have achieved considerable levels of trust and gathered around them communities of regular users and developers. But what is the social process whereby a proof is examined and reviewed? Because of the size of the proofs, the reviewing akin to that of the traditional mathematical world is unpractical. That did not prevent the mechanized proving community from seeking its own ways of ensuring a level of trust in the correctness of the "certificate" side of mechanized proofs.

MacKenzie [2001, p.313] asks "What is necessary to support the claim that a mechanized prover has performed a formal proof? Ought it to produce a full, formal proof object which can be submitted to an independent proof-checking program? Ought one seek to verify formally that a theorem-proving program has itself been implemented correctly?".

To trust the tool he or she uses, a researcher can be content with the way it was built or with the countless times it has been used. But to restore the social process where proofs are communicated outside the community of one prover, checking should be independent from that prover. Paraphrasing Socrates on his death bed, urging his friends to think only of the truth and not of himself, one can make the following request to his peers: "I would ask you to be thinking of the proof, and not of the prover".

As said above, there are already a number of investigations in bridging the gap between the provers through translations of the proofs. This thesis explores another path towards the same goal. Instead of translating the proofs from a source language to a target language, this thesis focuses on describing the semantics of the source language so that any proof written in that language can be checked without being translated.

By lifting the need for trust from the prover to the proof object, people can feel unshackled by the proof of correctness of their tools and implement bold ideas. In the words of Voevodsky [Martin, 2015], this can give rise to a "flowering of collaboration as it enables trust between participants, who can rely on the machine to check each other's work, and hence enables participants to take more risks".

Independent checking allows a liberty similar to Polymath's[1] third guideline: "it's OK" for a mechanized prover to be "tentative, incomplete, or even incorrect".

[1] http://polymathprojects.org/

1.2 Scope of the thesis

This thesis sits in the broader ERC Advanced Grant ProofCert[2] project [Miller, 2011], and is centered around first-order classical and intuitionistic logics.

As recommended in a recent paper by Plaisted [2015], investigations into first-order logic may uncover "significant advances yet to be made. In addition, one can expect that methods that are good for first-order logic will also help to design higher order logic provers."

The aim of this thesis is to establish a framework, called *Foundational Proof Certification*, or FPC, where the semantics of a multitude of proof languages or formats can be described. This is not the first time a unified framework is sought in the computer community, there are encouraging precedents. For example, finite state machines are used for lexical analysis. Formal grammars, originally introduced to describe natural language structures, are extensively used for formal language. One can also consider HTML as a useful standard to uniformly describe text such that any compatible browser can display it on screen. Perhaps the most resemblant effort is structural operational semantics [Plotkin, 1981], or SOS. The Foundational Proof Certification framework can define the semantics of many proof languages and checking is done according to the semantics. Similarly, SOS can define the semantics of many programming languages and compilation is done according to that semantics.

The thesis is centered around the same desiderata as the ProofCert project:

1. Simplicity: Barendregt and Wiedijk [2005] define a desirable characteristic of mathematical assistant (that can be generalized to all provers) in the following exact quote: "A Mathematical Assistant satisfying the possibility of independent checking by a small program is said to satisfy the de Bruijn criterion". However, the term "de Bruijn criterion" was used somewhat loosely to also describe systems that rely on an integrated (therefore not independent) kernel checker.

Additionally, they stipulate: a "small program" is one that "can be inspected in the usual way by a mathematician or logician", a requirement that very few small kernels of provers satisfy. Indeed, because no clear quantitative measure is given, kernels ranging from less than a thousand lines of code to tens of thousands of lines claim to satisfy the de Bruijn criterion. Others use the term "simple" instead of "small".

One can provide a less vague and more realistic definition by considering the size of the checker not in terms of its lines of code but in terms of the knowledge base that is required to reimplement it.

[2] team.inria.fr/parsifal/proofcert

Trust in such a checker will not only come from reading the code written by others but also from the ease in which one can reimplement one's own version of the checker.

2. Broadness: The checker should be able to accommodate a wide range of proof formats, or proof languages. One of the obstacles to communication is the cost, in time and energy, of learning to use new tools and/or establishing a translation of proofs between tools. This thesis tries to bypass this obstacle by focusing on describing the semantics of the languages instead of translating the proofs. Modularity in the definitions by employing relational specifications help reuse semantics descriptions when needed. The goal is to demand as little work as possible from the prover's side while minimizing the complexity of the semantics description.

3. Structure: The checked proof objects should certify proofs in the sense of structural proof theory, *i.e.,*proofs in which restricting to analytic proofs (*e.g.,* cut-free sequent proofs or normal natural deductions) preserves completeness. More details can be found in the following references on structural proof theory: [Gentzen, 1935, Prawitz, 1965, Troelstra and Schwichtenberg, 2000, Negri and von Plato, 2001]. The proof objects themselves, however, need not be a formal proof.

4. Trade-offs: One must acknowledge that proofs can be huge objects. This thesis addresses this obstacle by considering "proof evidence" instead of "proof". Proof evidence is any proof object indicating, with varying amount of details, that a proof has been found. To make this possible, the checker is granted proof reconstruction capabilities. Searching for missing parts in the proof may take time, however, and a trade-off appears between storage space and checking time. Of course, such a search should be known (by the designer of the proof certificate) to be a simple and bounded process.

For example, while a proof contains all instantiations of quantifiers, proof evidence can omit the instantiating terms. Using logic variables and relying on unification, the specific values of these terms can eventually be determined. Of course, the checker and the prover may not arrive at the same terms, but the checker only accepts valid instantiations, *i.e.,*ones that complete a correct proof. Similarly computation traces can be left out of the proof object.

In what follows, chapter 2 describes the bedrock of decades of research into logic and proof theory on which this thesis is built. Chapter 3 details the global architecture of the Foundational Proof Certification framework. Chapter 4 focuses on the process of semantics description. Chapter 5 and 6 present some case studies

respectively in classical and intuitionistic logics. Chapter 7 investigates term-level rewriting in reasoning with equality. Chapter 8 discusses some further characteristics of the Foundational Proof Certification framework. Finally, chapter 9 concludes.

Chapter 2

Preliminary notions on focusing

You can't depend on your eyes when your imagination is out of focus.

- Mark Twain, *A Yankee at the Court of King Arthur*

This chapter introduces the foundations on which the rest of this thesis is based. It starts with the classical and intuitionistic sequent calculi of Gentzen then presents their focused counterparts.

2.1 Sequent calculi: From Gentzen to Girard

When Gentzen embarked in a quest to find a "formalization of logical deduction", it was with the "specific task to find a proof of the consistency of logical deduction in arithmetic"[1]. That formalism of logical deduction was called Natural Deduction (ND). It differs from prior systems, like Hilbert-Frege systems, in the number of axioms as opposed to inference rules. In ND, there are a multitude of inference rules and close to no axioms, whereas in Hilbert-Frege systems, emphasis is put on axioms. The rules of ND are syntax directed, they deal with the top level connective of a formula, either eliminating it or introducing it (reading the rules top-down).

Definition 2.1.1. Function symbols *have fixed arity and a first-order* term *is built from function symbols and variables. The letters x, y, z are used for the variables and $f, g, h \ldots$ for the functions. If a function symbol has arity 0 then it may also be called a* constant *and be written as $a, b, c \ldots$. A first-order logic formula is inductively defined as follows. If B and C are formulas*[2]*, then so are:*

[1]Gentzen's words in a letter to his old teacher Hellmuth Kneser.

[2]This plural is preferred to *formulae* when not describing a mathematical formulae

- *The atomic formula obtained by the application of an n-ary ($n \geq 0$) predicate to a list of n first-order terms*
- $\neg B$, *denoting negation*
- $B \wedge C$, *denoting conjunction*
- t *denoting truth (which is the unit of conjunction)*
- $B \vee C$, *denoting disjunction*
- f *denoting falsehood (which is the unit of disjunction)*
- $B \supset C$, *denoting implication*
- $\forall x.B$, *denoting universal quantification*
- $\exists x.B$, *denoting existential quantification*

with the decreasing order of precedence being $\neg, \wedge, \vee, \supset$, *and the quantifiers binding until the end of the formula or the enclosing parentheses. For instance, the formula*

$$\forall x.A \vee B \wedge C \supset D \vee \exists y.E \vee F$$

is taken to be the fully parenthesized formula

$$\forall x.((A \vee (B \wedge C)) \supset (D \vee (\exists y.(E \vee F))))$$

•

Definition 2.1.2. *A* sequent *is a conditional assertion written* $A_1, \cdots, A_n \vdash B_1, \cdots, B_m$, *where* B_i *and* A_j *are formulas and* n *and* m *non-negative integers. Formulas on the left-hand side of* turnstile *(*$\vdash$*) are taken conjunctively and those on the right-hand side are taken disjunctively. The sequent above "has exactly the same informal meaning" [Gentzen, 1935] as the formula*

$$A_1 \wedge \cdots \wedge A_n \supset B_1 \vee \cdots \vee B_m$$

A sequent is said to be one-sided *when it has no formulas on the left.* •

The difference between natural deduction and sequent calculus is that the former contains elimination rules and introduction rules, whereas the latter contains introduction rules on the left or on the right hand-side of a *turnstile* ($\vdash$).

Definition 2.1.3. *A* sequent calculus *is a system where sequents are derived from other sequents using a set of inference rules. A rule of inference is usually read bottom-up,* i.e.,*is said to be* applied *to the conclusion sequent (and, sometimes, applied to a formula in the conclusion sequent) to yield the premise sequents. The rules of a sequent calculus system can be divided in two groups. The first group contains the so called* introduction *rules, each introducing the top level connective of a formula, on the right-hand side or the left-hand side of the turnstile. The second group contains so called* structural *and* identity *rules. These rules do not introduce any connective but can move or copy a formula to restructure the shape of a sequent. For all rules of both these groups, the formula in the conclusion sequent on which the rule is applied is called the* principal formula. •

Definition 2.1.4. *A system is said to be* sound *if it only proves true statements, with respect to a given logic. It is said to be* complete *if it proves all true statements of a given logic.* •

2.1.1 Classical sequent calculus

This section introduces the one-sided version of the classical sequent calculus *LK*. Restricting the *LK* system as originally presented by Gentzen to one-sided sequents is possible because of the left/right symmetry in the introduction rules.

For example, the left rule of the conjunction:

$$\frac{\Gamma, B_i \vdash \Delta}{\Gamma, B_1 \wedge B_2 \vdash \Delta}$$

and the right rule of the disjunction:

$$\frac{\Gamma \vdash B_i, \Delta}{\Gamma \vdash B_1 \vee B_2, \Delta}$$

can be seen as mirror of each other. Thanks to this symmetry, the number of rules of the classical sequent calculus can be reduced by half and yield what is known as a one-sided sequent calculus, shown in figure 2.1. The sequents are of the form $\vdash \Gamma$ where Γ is a multiset of formulas, and where an eigenvariable is defined as follows:

Definition 2.1.5. *An eigenvariable is a constant symbol that satisfies the freshness condition,* i.e.,*it does not appear anywhere in the lower sequents of the rules that introduce it.* •

Introduction rules

$$\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \wedge B} \wedge \qquad \frac{}{\vdash \Gamma, t}\, t$$

$$\frac{\vdash \Gamma, [y/x]B}{\vdash \Gamma, \forall x.B} \forall \qquad \frac{\vdash \Gamma, [s/x]B}{\vdash \Gamma, \exists x.B} \exists \qquad \frac{\vdash \Gamma, A_i}{\vdash \Gamma, A_1 \vee A_2} \vee$$

Structural and identity rules

$$\frac{}{\vdash \Gamma, \neg A, A}\, I \qquad \frac{\vdash \Gamma, F \qquad \vdash \Gamma, \neg F}{\vdash \Gamma}\, Cut \qquad \frac{\vdash \Gamma, F, F}{\vdash \Gamma, F}\, Contraction$$

Figure 2.1: The classical one-sided sequent calculus LK, where s is a first-order term and y is an eigenvariable (not free in the conclusion).

Definition 2.1.6. *A formula with no implication $\supset$ is in* negation normal form *if all negations have atomic scope.* •

Any formula can be put in its classically equivalent negation normal form following these (classically valid) formula transformations:

1. the material implication interpretation allows replacing all implications $B \supset C$ by the equivalent $\neg B \vee C$
2. the double-negation elimination replaces all formulas of the form $\neg\neg B$ with B
3. the extended de Morgan Laws replace all formulas $\neg(B \wedge C)$ with $\neg B \vee \neg C$, all formulas $\neg(B \vee C)$ with $\neg B \wedge \neg C$, all formulas $\neg\exists x.B$ with $\forall x.\neg B$, all formulas $\neg\forall x.B$ with $\exists x.\neg B$, all $\neg t$ with f and all $\neg f$ with t

These transformations push the negations downwards until negations have only atomic scope. Henceforth, the negation symbol in $\neg B$ stands for the negation normal form of the negation of B.

Theorem 2.1.1. *The LK system (one or two sided) is sound and complete with respect to classical logic.*

2.1.2 Intuitionistic sequent calculus

The difference between intuitionistic logic and classical logic causes one of the deepest cleavages of the community. It started with what is known as the Foundational Crisis

Right introduction rules

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B} \supset_r \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_r \qquad \frac{}{\Gamma \vdash t}\, t_r$$

$$\frac{\Gamma \vdash [y/x]B}{\Gamma \vdash \forall x.B} \forall_r \qquad \frac{\Gamma \vdash [s/x]B}{\Gamma \vdash \exists x.B} \exists_r \qquad \frac{\Gamma \vdash A_i}{\Gamma \vdash A_1 \vee A_2} \vee_r$$

Left introduction rules

$$\frac{\Gamma \vdash A \quad \Gamma, B \vdash R}{\Gamma, A \supset B \vdash R} \supset_l \qquad \frac{\Gamma, A_i \vdash R}{\Gamma, A_1 \wedge A_2 \vdash R} \wedge_l \qquad \frac{\Gamma \vdash R}{\Gamma, t \vdash R}\, t_l$$

$$\frac{\Gamma, [s/x]B \vdash R}{\Gamma, \forall x.B \vdash R} \forall_l \qquad \frac{\Gamma, [y/x]B \vdash R}{\Gamma, \exists x.B \vdash R} \exists_l \qquad \frac{\Gamma, A \vdash R \quad \Gamma, B \vdash R}{\Gamma, A \vee B \vdash R} \vee_l \qquad \frac{}{\Gamma, f \vdash R}\, f_l$$

Structural and identity rules

$$\frac{}{\Gamma, A \vdash A}\, I_r \qquad \frac{\Gamma \vdash F \qquad \Gamma, F \vdash R}{\Gamma \vdash R}\, Cut \qquad \frac{\Gamma, F, F \vdash R}{\Gamma, F \vdash R}\, Contraction$$

Figure 2.2: The intuitionistic sequent calculus LJ, where s is a first-order term and y is an eigenvariable.

in mathematics and, while carried out by many mathematicians, no other duel is remembered as much as the one between Hilbert and Brouwer. This project takes no sides in this debate but rather aims at flexibly accommodating both intuitionistic and classical proof evidence.

Amazingly, simply restricting the right-hand side of the turnstile to contain at most one formula is enough to have an intuitionistic sequent calculus. The resulting rules are given in figure 2.2. The *LJ* sequent is of the form $\Gamma \vdash B$ where Γ is a multiset of formulas and B is a formula.

Theorem 2.1.2. *The LJ system is sound and complete with respect to intuitionistic logic.*

2.2 Focused sequent calculi

When Gentzen described his sequent calculi, he evidently did not have machine generated proofs in mind. While sound and complete, there is a hint of chaos and a certain ambiguity in the LK and LJ systems that make them ill-suited for proof

$$\frac{}{\vdash \Delta, A, \neg A}\ init \qquad \frac{\vdash \Delta, B, C}{\vdash \Delta, B \vee C}\ \vee \qquad \frac{\vdash \Delta, B \quad \vdash \Delta, C}{\vdash \Delta, B \wedge C}\ \wedge$$

Figure 2.3: One-sided propositional classical sequent calculus

automation. In order to use sequent calculus as the basis of automated deduction, much more structure within proofs needs to be established. Uniform Proofs [Miller et al., 1991] are one of many systems of computational logic that restrict sequent calculus in soundness and completeness preserving ways. A Uniform Proof is organized as an alternation of two phases, that the authors call *goal-directed search* and *backchaining*. Andreoli, in [Andreoli, 1992], applied this same organization to linear logic and discovered *focusing*. Later on, LKF and LJF appeared in [Liang and Miller, 2009] as the focused versions of LK and LJ systems, of which many others focused classical and intuitionistic systems are special cases.

While some may consider them to be too low-level to be of serious utility in proof automation, the use of sequent calculi in mechanized proofs is not new. Several efforts are known, from the precursor work of [Wang, 1960] to the recent and widely known method of analytic tableaux [Fitting, 1990].

2.2.1 The tenets of focusing

This section introduces focusing notions that are more general than the *LK* and *LJ* sequent calculi.

Invertibility and contraction

Consider the contraction-free, one-sided sequent calculus for propositional classical logic on formulas in negation normal form, consisting of the three rules in figure 2.3 where A is atomic. It should be noted that the $\vee$ rule is different from that in figure 2.1, but both rules are accepted in classical sequent calculus through the admissibility of the contraction rule.

The lack of contraction impedes neither soundness nor completeness. If a classical propositional formula in negation normal form is provable then a proof exists in the above system. Moreover, because there is no contraction, a syntax directed proof-search procedure with the rules of this system necessarily terminates, which is a good step towards automation.

The reason why contraction is superfluous appears with a closer look at the introduction rules. In definition 2.1.2, the sequence of formulas on the right-hand side

of a turnstile was said to be equivalent to a disjunction of those formulas. Therefore, the $\vee$ rule only "lifts" the disjunction to the (essentially same) *meta*-disjunction. Similarly, the $\wedge$ rule has two premises, both of which have to be provable. They can thus be seen as being linked with a *meta*-conjunction.

These rules, where the conclusion and the premise are equiprovable, are called *invertible*. Because no information is lost, going from conclusion to premise, one can always permute any of the two rules over any of the others. The following derivations, for example, are equivalent:

$$\dfrac{\dfrac{\vdash B, C, D, \Delta \quad \vdash B, C, F, \Delta}{\vdash B, C, D \wedge F, \Delta}\ \wedge}{\vdash B \vee C, D \wedge F, \Delta}\ \vee \qquad \dfrac{\dfrac{\vdash B, C, D, \Delta}{\vdash B \vee C, D, \Delta}\ \vee \quad \dfrac{\vdash B, C, F, \Delta}{\vdash B \vee C, F, \Delta}\ \vee}{\vdash B \vee C, D \wedge F, \Delta}\ \wedge$$

Remark 2.2.1. Consider the process of taking a sequent $\vdash \Gamma$ and eagerly applying the conjunction and disjunction rules whenever, forming a sequence of rules S and obtaining as premises the sequents $\vdash \Gamma_1 \cdots \vdash \Gamma_n$. A crucial side-effect of invertibility is that, for that sequent $\vdash \Gamma$, any permutation of the sequence S will always yield the same multiset of premises $\vdash \Gamma_1 \cdots \vdash \Gamma_n$ as output. One can then see the process of applying these invertible rules as a deterministic function.

There is no loss in completeness or soundness, then, in choosing an arbitrary order (left-most, for example) in which to apply these rules.

If one wants to extend the previous sequent calculus from propositional to first order, one can add the following rules.

$$\dfrac{\vdash \Delta, [y/x]B}{\vdash \Delta, \forall x.B}\ \forall \qquad \dfrac{\vdash \Delta, [t/x]B}{\vdash \Delta, \exists x.B}\ \exists \qquad \dfrac{\vdash \Delta, B, B}{\vdash \Delta, B}\ contr$$

The $\forall$ rule is also invertible, thus if only this rule is added, contraction would still be superfluous. However, contraction is needed when adding the $\exists$ connective and associated rule; the Drinker's paradox $(\exists x.\neg P(x) \vee \forall y.P(y))$, classically valid, cannot be proved without contraction. Furthermore, the $\exists$ rule is the only rule that is not invertible, so one can restrict contraction to be only on $\exists$, *i.e.,*require that B be an existential. More generally, contraction can be restricted to those formulas the introduction rule of which is not invertible.

Focusing

With this distinction between invertible and non-invertible rules, one can take a further step towards making sequent calculus suitable for proof automation by organizing the proofs into two phases: in one phase, all invertible rules are applied

consecutively (reading rules bottom-up, *i.e.,*as being applied to the conclusion to yield the premises) until no more invertible rules can be applied.

In the other phase, contraction and non-invertible rules are applied. In focused proof systems this phase is structured as follows. One starts by deciding on a single formula on which to place the focus. When under focus, a formula is the principal formula of a non-invertible rule, and the resulting subformulas are, themselves, the principal formulas of the following non-invertible rules. The focus is maintained until an initial or a unit rule is applied to end the derivation or until the formula under focus appears with a top-level connective whose rule is invertible. In the latter case, the focus is released and an invertible phase begins again.

Polarity

Polarities are simple annotations given to connectives to indicate, at the syntactic level, in which phase they are introduced. If a connective is of negative polarity, its right-introduction rule is invertible and its left-introduction rule (if any) is focused. If a connective is of positive polarity, its left-introduction rule (if any) is invertible and its right-introduction rule is focused. A connective is said to *belong* in an invertible phase if it is the positive top level connective of a formula that appears on the left-hand side of the turnstile or if it is the negative top level connective of a formula that appears on the right-hand side of the turnstile. A connective is said to *belong* in a focused phase if it is the positive top level connective of a formula that appears on the right of the turnstile or if it is the negative top level connective of a formula that appears on the left-hand side of the turnstile. Contraction can be restricted to those formulas that belong in a focused phase.

The conjunction and disjunction and their units, t and f, are said to be *ambiguous* because they come in two versions, both a negative and a positive one, resulting in the following connectives: $\vee^+, \wedge^-, \vee^-, \wedge^+$ and their units, respectively f^+, t^-, f^-, t^+.

The implication $\supset$ and universal $\forall$ are *unambiguously* negative while the $\exists$ is unambiguously positive. The atomic formulas are also given a negative or positive polarity.

A formula is positive (resp. negative) if its top level connective is positive (resp. negative), it is *strictly* positive (resp. *strictly* negative) if all its connectives and atoms are positive (resp. negative).

While the provability of a formula is not compromised by the polarities given to its connectives and atoms, the structure of the proofs of that formula can differ greatly from one polarity assignment to another. Polarities also make it possible to replace negative statements such as "apply invertible rules until no more can be

applied" by positive statements such as "apply invertible rules until all remaining formulas are positive".

Delays

The focusing behavior adopted for this work is somewhat aggressive because the focused phases and the invertible phases are maximal, *i.e.,*every connective belonging and appearing in a phase is introduced.

However, in some cases one might want more control over the focusing by breaking a focused phase prematurely or by postponing the introduction of a connective. For this, we have unary connectives called positive and negative *delays*, $\partial^{-}(\cdot)$ and $\partial^{+}(\cdot)$. The idea is that $\partial^{-}(B)$ is always negative and $\partial^{+}(B)$ is always positive no matter what the polarity of B is. These delay operators are easily defined using polarized logical connectives: we can take the official definitions to be $\partial^{-}(B) = \forall x.B$ and $\partial^{+}(B) = \exists x.B$ (provided that x is not free in B). Alternatively, they can be defined to be the unary versions of the binary $\wedge^{-}$ and $\wedge^{+}$ connectives, respectively.

Structural and identity rules

Similar to the original sequent calculi, there are structural rules.

There are three structural rules in a focused system:

- Store: When the sequent is in an invertible phase, all formulas that are encountered are *stored* if they do not belong in an invertible phase.
- Decide: At the end of an invertible phase (when no connective can be introduced), a previously stored formula is chosen if it belongs in a focused phase. It becomes the formula under focus.
- Release: When the sequent is in a focused phase and the formula under focus does not belong in a focused phase, this phase ends with a *release* and the sequent reverts to an invertible phase.

Definition 2.2.1. *A* bipole *is a cut-free derivation consisting of a focused phase (starting with a decide rule) followed by an invertible phase.* •

A related notion is the following:

Definition 2.2.2. *A* decide depth *is the number of decide rules of a given branch of a proof derivation. The maximal decide depth of the depths of all the branches in a proof tree is the decide depth of that tree.* •

The decide depth is often taken as a measure of the size of proofs that is less fine-grained than the traditional number of rules. A bounded decide depth implies a terminating (cut-free) proof search.

Identity rules

This last group of rules in a focused system contains the *cut* and *initial* rules. The *initial* rules can finish a proof and they are applied on an atom of the adequate polarity at the end of a focused phase (*i.e.,*there is no initial rule in an invertible phase). While multiple cut rules are possible (see [Liang and Miller, 2009]), the cut rule used in this thesis only appears at the end of an invertible phase.

2.2.2 Focused classical sequent calculus *LKF*

Sequents of *LKF* come in two kinds, unfocused, or up-arrow sequents: $\vdash \Theta \Uparrow \Gamma$ where Θ is a multiset and Γ is a list, on which the invertible rules are applied from left to right, and focused, or down-arrow sequents: $\vdash \Theta \Downarrow P$ where Θ is a multiset and P is a formula. The left-hand side of the up or down arrow is called *storage* while the right-hand side of the up or down arrow is called *workbench*. It is an invariant that Θ only contains positive formulas or negative atoms. The *LKF* system recognizes all connectives given above except for the implication $\supset$, that is, *LKF* formulas follow this grammar:

$$\begin{aligned} B, C &::= \quad B \wedge^- C \mid B \wedge^+ C \mid B \vee^+ C \mid B \vee^- C \mid \forall x.B \mid \exists x.B \\ &\qquad \mid A \mid \neg A \mid t^+ \mid t^- \mid f^+ \mid f^- \\ A &::= \quad atm^+ \mid atm^- \end{aligned}$$

where *atm* is an atomic formula. The superscripts $+, -$ on the atoms are omitted when clear from context.

The negation normal form of a polarized formula is very similar to that of an unpolarized formula. The two differences are:

- the de Morgan dual flips the polarity, *e.g.,* $\neg(B \vee^+ C)$ gives $(\neg B) \wedge^- (\neg C)$
- the smallest formula is not a literal (atom or negation of an atom) but a polarized atom, *i.e.,*the negation symbols does not appear even at the atomic level
- when written in the context of *LKF*, the negation $\neg B$ stands for the negation normal form of the negation of B.

Remark 2.2.2. In *LKF*, atoms have positive and negative polarities and there is no negation connective in *LKF*. For this reason, the term *atoms* is preferred when talking about *LKF*'s basic (without connectives) formulas whereas the term *literals* is preferred when talking about the unpolarized classical formulas. But because atoms are also literals, the latter is sometimes used to designate *LKF*'s basic formulas. Henceforth, the terms "literal" and "atoms" will only be differentiated when unclear from the context.

Remark 2.2.3. There is no correlation between a *negative* atom and a *negated* atom. The former is an atom of a negative polarity in a focused system, the latter is an atom in a non-focused system with the negation symbol in front of it. In particular, when choosing a polarity, an atom can be mapped to either polarity say p, then each non-negated occurrence of it will yield an atom polarized with p while each negated occurrence will yield an atom with the opposite polarity. These two polarized atoms are said to be *complementary*. Writing $\neg a$ for a polarized atom a indicates its complement, *not* its negation (the negation symbol in front of it).

Example 2.2.1. *To illustrate the difference in structure of proof resulting from a polarity assignment, consider the focused proof of the formula* $\neg p \vee^{+} (C \vee^{+} p)$ *where* C *is a formula with a large number of conjunctions and disjunctions:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \neg p \vee^{+} (C \vee^{+} p), \neg p \Downarrow p}}{\vdash \neg p \vee^{+} (C \vee^{+} p), \neg p \Downarrow C \vee^{+} p}}{\vdash \neg p \vee^{+} (C \vee^{+} p), \neg p \Downarrow \neg p \vee^{+} (C \vee^{+} p)}}{\vdash \neg p \vee^{+} (C \vee^{+} p), \neg p \Uparrow \cdot}}{\vdash \neg p \vee^{+} (C \vee^{+} p) \Uparrow \neg p}}{\vdash \neg p \vee^{+} (C \vee^{+} p) \Downarrow \neg p}}{\vdash \neg p \vee^{+} (C \vee^{+} p) \Downarrow \neg p \vee^{+} (C \vee^{+} p)}}{\vdash \neg p \vee^{+} (C \vee^{+} p) \Uparrow \cdot}}{\vdash \cdot \Uparrow \neg p \vee^{+} (C \vee^{+} p)}$$

The proof of the provably equivalent formula $\neg p \vee^{-} (C \vee^{-} p)$ *will be exponentially bigger (albeit smaller in decide depth). Indeed, every conjunction rule will essentially copy the surrounding sequent in each of its premises, and it only stops once the formula is decomposed into atoms. These atoms are then stored and in each of the (potentially many) premises, the pair* $\langle p, \neg p \rangle$ *will be present and can end the proof after a decide rule and an initial rule.* $\odot$

Theorem 2.2.1. *The focusing system LKF is sound and complete* w.r.t.*the original system LK. More formally, let B be an unpolarized first-order formula in negation*

Invertible introduction rules

$$\frac{}{\vdash \Theta \Uparrow t^-, \Gamma} \quad \frac{\vdash \Theta \Uparrow A, \Gamma \quad \vdash \Theta \Uparrow B, \Gamma}{\vdash \Theta \Uparrow A \wedge^- B, \Gamma} \quad \frac{\vdash \Theta \Uparrow \Gamma}{\vdash \Theta \Uparrow f^-, \Gamma} \quad \frac{\vdash \Theta \Uparrow A, B, \Gamma}{\vdash \Theta \Uparrow A \vee^- B, \Gamma} \quad \frac{\vdash \Theta \Uparrow [y/x]B, \Gamma}{\vdash \Theta \Uparrow \forall x.B, \Gamma}$$

Focused introduction rules

$$\frac{}{\vdash \Theta \Downarrow t^+} \quad \frac{\vdash \Theta \Downarrow B_1 \quad \vdash \Theta \Downarrow B_2}{\vdash \Theta \Downarrow B_1 \wedge^+ B_2} \quad \frac{\vdash \Theta \Downarrow B_i}{\vdash \Theta \Downarrow B_1 \vee^+ B_2}\ i \in \{1,2\} \quad \frac{\vdash \Theta \Downarrow [s/x]B}{\vdash \Theta \Downarrow \exists x.B}$$

Identity rules

$$\frac{}{\vdash \neg P_a, \Theta \Downarrow P_a}\ init \quad \frac{\vdash \Theta \Uparrow B \quad \vdash \Theta \Uparrow \neg B}{\vdash \Theta \Uparrow \cdot}\ cut$$

Structural rules

$$\frac{\vdash \Theta, C \Uparrow \Gamma}{\vdash \Theta \Uparrow C, \Gamma}\ store \qquad \frac{\vdash \Theta \Uparrow N}{\vdash \Theta \Downarrow N}\ release \qquad \frac{\vdash P, \Theta \Downarrow P}{\vdash P, \Theta \Uparrow \cdot}\ decide$$

Here, P is a positive formula; N a negative formula; P_a a positive literal; C a positive formula or negative literal; s is a first-order term; y is an eigenvariable.

Figure 2.4: *LKF*: a focused proof system for classical logic

normal form without implications. Let $\hat{B}$ be a polarized formula that results from (i) choosing either a negative or a positive version for each occurrence of the ambiguous connectives (ii) picking some global polarization of atomic formulas, and (iii) inserting any number of delays anywhere into the formulas. Then, $\vdash B \iff \vdash \cdot \Uparrow \hat{B}$.

One can see in figure 2.5 that polarity is simply a notation and that, by removing the up and down arrows and the polarity superscripts, one finds the usual one-sided *LK* rules.

2.2.3 Intuitionistic sequent calculus LJF

Sequents in *LJF*, presented in figure 2.6, come in three kinds, one unfocused and two focused sequents. The unfocused sequent: $\Gamma \Uparrow \Theta \vdash \Delta_1 \Uparrow \Delta_2$ where the union of Δ_1 and Δ_2 contains exactly one element, Γ is a multiset and Θ is a list. The invertible rules are applied on Θ from left to right, and on Δ_1 only if Θ is empty. Δ stands

Rules from one-sided *LKF*

$$\frac{\vdash \Theta \Uparrow A, \Gamma \quad \vdash \Theta \Uparrow B, \Gamma}{\vdash \Theta \Uparrow A \wedge^- B, \Gamma} \quad \frac{\vdash \Theta \Uparrow [y/x]B, \Gamma}{\vdash \Theta \Uparrow \forall x.B, \Gamma} \quad \frac{\vdash \Theta \Uparrow N}{\vdash \Theta \Downarrow N}\ release \quad \frac{\vdash P, \Theta \Downarrow P}{\vdash P, \Theta \Uparrow \cdot}\ decide$$

Rules from one-sided *LK*

$$\frac{\vdash \Theta A, \Gamma \quad \vdash \Theta B, \Gamma}{\vdash \Theta A \wedge B, \Gamma} \quad \frac{\vdash \Theta[y/x]B, \Gamma}{\vdash \Theta \forall x.B, \Gamma} \quad \frac{\vdash \Theta, N}{\vdash \Theta, N}\ null \quad \frac{\vdash P, \Theta P}{\vdash P, \Theta \cdot}\ contraction$$

Figure 2.5: Focused and unfocused rules

for $\Delta_1 \Uparrow \Delta_2$. The sequent has four zones, the middle zones (left-hand side of the up-arrow on the right-hand side of the turnstile and the right-hand side of the up or down arrow on the left-hand side of the turnstile) are called (left and right) storage while the other two zones are called (left and right) workbenches.

The focused sequents are the left-focused sequent $\Gamma \Downarrow N \vdash R$ and the right-focused sequent $\Gamma \vdash P \Downarrow$. In both these sequents, R and Γ are also called storage while P and N are in the workbench.

It is an invariant that Γ only contains negative formulas or positive atoms. The *LJF* system recognizes all connectives given above except for the negative disjunction, $\vee^-$, and its unit f^-.

Theorem 2.2.2. *The focusing system LJF is sound and complete* w.r.t.*the original system LJ. More formally, let C be an unpolarized first-order formula. Let $\hat{C}$ be a polarized formula that results from (i) choosing either a negative or a positive version for each occurrence of the conjunction and its unit t and giving the positive polarity to the disjunction, (ii) picking some polarization globally for atomic formulas, and (iii) inserting any number of delays anywhere into the formulas. Then, $\cdot \vdash C \iff \cdot \Uparrow \cdot \vdash \hat{C} \Uparrow \cdot$*

Remark 2.2.4. By removing all notation from focused systems (up and down arrows, polarities) one obtains the usual sequent calculi of Gentzen (albeit in verbose versions). In particular the contraction corresponds to decide rules (only left-decide rule for *LJF*).

Invertible Rules

$$\frac{\Gamma \Uparrow A \vdash B \Uparrow}{\Gamma \Uparrow \;\vdash A \supset B \Uparrow} \supset_r \qquad \frac{\Gamma \Uparrow \;\vdash A \Uparrow \quad \Gamma \Uparrow \;\vdash B \Uparrow}{\Gamma \Uparrow \;\vdash A \wedge^- B \Uparrow} \wedge_r^- \qquad \frac{}{\Gamma \Uparrow \;\vdash t^- \Uparrow} t_r^-$$

$$\frac{\Gamma \Uparrow \;\vdash [y/x]B \Uparrow}{\Gamma \Uparrow \;\vdash \forall x.B \Uparrow} \forall_r \qquad \frac{\Gamma \Uparrow [y/x]B, \Theta \vdash \Delta}{\Gamma \Uparrow \exists x.B, \Theta \vdash \Delta} \exists_l \qquad \frac{}{\Gamma \Uparrow f^+, \Theta \vdash \Delta} f_l^+$$

$$\frac{\Gamma \Uparrow A, B, \Theta \vdash \Delta}{\Gamma \Uparrow A \wedge^+ B, \Theta \vdash \Delta} \wedge_l^+ \qquad \frac{\Gamma \Uparrow \Theta \vdash \Delta}{\Gamma \Uparrow t^+, \Theta \vdash \Delta} t_l^+ \qquad \frac{\Gamma \Uparrow A, \Theta \vdash \Delta \quad \Gamma \Uparrow B, \Theta \vdash \Delta}{\Gamma \Uparrow A \vee B, \Theta \vdash \Delta} \vee_l$$

Focused Rules

$$\frac{\Gamma \vdash A \Downarrow \quad \Gamma \Downarrow B \vdash D}{\Gamma \Downarrow A \supset B \vdash D} \supset_l \qquad \frac{\Gamma \vdash A_i \Downarrow}{\Gamma \vdash A_1 \vee A_2 \Downarrow} \vee_r \qquad \frac{\Gamma \Downarrow A_i \vdash D}{\Gamma \Downarrow A_1 \wedge^- A_2 \vdash D} \wedge_l^-$$

$$\frac{\Gamma \Downarrow [s/x]B \vdash D}{\Gamma \Downarrow \forall x.B \vdash D} \forall_l \qquad \frac{\Gamma \vdash A \Downarrow \quad \Gamma \vdash B \Downarrow}{\Gamma \vdash A \wedge^+ B \Downarrow} \wedge_r^+ \qquad \frac{}{\Gamma \vdash t^+ \Downarrow} t_r^+ \qquad \frac{\Gamma \vdash [s/x]B \Downarrow}{\Gamma \vdash \exists x.B \Downarrow} \exists_r$$

Identity rules

$$\frac{N_a \text{ atomic}}{\Gamma \Downarrow N_a \vdash N_a} \mathbb{I}_l \qquad \frac{P_a \text{ atomic}}{\Gamma, P_a \vdash P_a \Downarrow} \mathbb{I}_r \qquad \frac{\Gamma \Uparrow \;\vdash F \Uparrow \qquad \Gamma \Uparrow F \vdash \;\Uparrow D}{\Gamma \Uparrow \;\vdash \;\Uparrow D} \mathbb{C}$$

Structural rules

$$\frac{\Gamma, N \Downarrow N \vdash D}{\Gamma, N \Uparrow \;\vdash \;\Uparrow D} \mathbb{D}_l \qquad \frac{\Gamma \vdash P \Downarrow}{\Gamma \Uparrow \;\vdash \;\Uparrow P} \mathbb{D}_r \qquad \frac{\Gamma \Uparrow P \vdash \;\Uparrow D}{\Gamma \Downarrow P \vdash D} \mathbb{R}_l \qquad \frac{\Gamma \Uparrow \;\vdash N \Uparrow}{\Gamma \vdash N \Downarrow} \mathbb{R}_r$$

$$\frac{C, \Gamma \Uparrow \Theta \vdash \Delta}{\Gamma \Uparrow C, \Theta \vdash \Delta} \mathbb{S}_l \qquad \frac{\Gamma \Uparrow \;\vdash \;\Uparrow D}{\Gamma \Uparrow \;\vdash D \Uparrow} \mathbb{S}_r$$

Here, P is positive; N is negative; C is a negative formula or positive atom; and D a positive formula or negative atom; Other formulas are arbitrary; s is a first-order term; y is an eigenvariable.

Figure 2.6: The intuitionistic sequent calculus *LJF*.

Chapter 3

Global architecture

A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools.

- Douglas Adams, *Mostly Harmless*

The first chapter introduced the habitat and aims of this thesis. The second chapter presented sequent calculi and notions of focusing which are the theoretical foundations necessary for the understanding of this research effort.

This chapter explains how these foundations answer the objectives and desiderata set in the introduction. Section 3.1 explains desired characteristics of the framework and the advantages of this approach. Section 3.2 discusses how the properties given to the framework may be adjusted to fit the needs of specific checking tasks. Section 3.3 informally describes the three components around which the Foundational Proof Certification framework is articulated and, in particular, details the workload associated with (and the knowledge base needed for) each component. Finally, section 3.4 and 3.5 introduce the kernels used in this thesis, similar to the one presented by Chihani et al. [2013a]. These kernels are versions of the previously seen *LKF* and *LJF* systems that are modified to allow guidance of the kernel in a completely sound fashion.

These last sections explain *how* the kernel can be guided to the intended proof of a proposed theorem, *not* how this guidance is actually defined. The definition of the guidance is related to the language semantics description and is the subject of next chapter. In other words, the focus is put here on how some signals can influence the kernel, or what is the protocol with which it communicates with the proof certificate, *not* what constitutes those signals.

3.1 Desired properties

In the introduction, several desiderata were presented to structure this thesis and impose boundaries and guiding principles. This section proposes ways of fulfilling the desiderata.

3.1.1 Poincaré Principle

The first desired property is the ability to separate computation from deduction, answering the 3rd desideratum. This desideratum is somewhat more general than the so-called Poincaré Principle, coined in [Barendregt, 1997]. Barendregt's notion of the Poincaré principle follows the remark, made by Henri Poincaré in his book Science and Hypothesis, that an argument showing $2 + 2 = 4$ is not a proof but a verification that can be carried out by a mechanical algorithm. The traces of these computations can be left out of a proof to reduce its size.

The notion of computation, however, is not limited to mathematic calculation. As noted in remark 2.2.1, an invertible phase of a focused system will always give the same (set of premises as) output when presented with the same (unfocused sequent as) input. This operation is therefore no less determined than an argument for $2 + 2 = 4$. This determinacy allows leaving out more information from the proof evidence and is one of the perks of focusing. Building the kernel on top of a focused logic is then a desired property and, while this framework is presented through *LKF* and *LJF*, the same ideas apply by using other focused sequent calculi such as *LKU* [Liang and Miller, 2011], $\mu MALLF$ [Baelde, 2012] and others [Laurent, 2002, Simmons, 2014].

3.1.2 Modular integration

In addition to the proof compression obtained by a focusing setting, one can further reduce the amount of information in the proof evidence if parts of the proof can be reconstructed using a decision procedure (G4ip[Dyckhoff, 1992] for example). Then the reduction in size can also benefit from modularly calling such previously defined procedures on those parts. Another desired property is thus support for modular integration.

The use of this property can be broadened to combining multiple proof evidence semantics definitions to check proof evidence that involves more than one proof language (*e.g.,* integrating semantics for equational reasoning with that of dependently typed λ-calculus to check $\lambda\Pi$-modulo proofs).

However, support for such integration possibilities must provide security guarantees.

3.1.3 Abstractions and typing

Ensuring security properties is paramount if one wants to seal the checker from outside tampering and have stronger confidence in the correctness of the implementations. Types, in particular *abstract data types*, as well as the ability to hide parts of modules when integrated are often used for such a goal in mind. The LCF approach achieves high levels of trust by relying on clever usage of types: theorems have a particular abstract data type `thm`, and the language is comprised of axioms of that same type, along with functions, denoting inference rules, to build `thm` objects from `thm` objects. One is assured, then, that if a theorem (of type `thm`) has been proven, it is either the result of an axiom or of sound repeated applications of inference rules to previously derived formulas. The Foundational Proof Certification framework offers the same guarantees.

Furthermore, by means of a careful separation, an erroneous (or even malicious) proof certificate will never cause the checking mechanism to validate a falsehood.

3.1.4 Declarative and relational

The framework relies on a relational setting. In [Wiedijk, 2007], the author provocatively states that the focus in formal mathematics should be on declarative systems (*e.g.,* Isabelle/HOL and Mizar) over procedural systems (*e.g.,* Coq), arguing both that this is the recognizable, traditional way of doing mathematics and that a declarative setting is more robust to changes in foundations.

The Foundational Proof Certification is concerned with machine-generated and machine-checked documents to be communicated between machines. In this environment, a human has little role and therefore being "recognizable and traditional" is not a concern when it comes to proof evidence. However, when describing the semantics of a language using the Foundational Proof Certification framework, readability and concision are valuable attributes that promote trust.

The latter argument of robustness also applies, especially in making the framework benefit from modularity and flexibly accommodate a wide range of proof evidence languages, which is the second desideratum.

While the declarative paradigm includes both relational and functional settings, a relational setting is preferred because a function is a special case of a relation. This generality is sought also to satisfy the second desideratum.

3.1.5 Proof reconstruction tools

Another flexibility that this project aims at is the possibility of proof reconstruction. Unification and backtracking search play a vital role in this setting. Indeed, non-determinism, far from being a liability, is a useful resource. The order in which unification can be done depends on the scope of the checker, it can range from no unification to first-order unification to higher-order pattern unification.

A trade-off appears between search time, reduced by greater details, and storage space, reduced by fewer details.

3.1.6 Parallelism

With the same goal of generality in mind, and because sequentiality is a special case of parallelism, the FPC framework supports parallelism. If part of the proof certificate instructs that some portions of the proof are obtained independently, the framework should check this parallelism. This is achieved, for example, through the use of multi-cuts and multi-focus.

This thesis, limited to first-order classical and intuitionistic logics, does not investigate these possibilities further. However, previous efforts showed how parallelism in a focused setting can be used to obtain canonicity of sequent proofs [Chaudhuri et al., 2008] as well as an isomorphism for expansion proofs [Chaudhuri et al., 2014]. Future parts of the ProofCert project include treatment of modal logics, to which these ideas also apply.

3.1.7 Additional features

Together with the above properties, other features of arguably less importance can be useful.

- λ-tree syntax is a logically supported mechanism particularly useful when checking proofs involving bindings (substitutions, α-conversions), be it at the term level or at the formula level. This can be of great use when dealing with quantifier-related challenges (eigenvariable generation, quantifier instantiation). Deeper understanding of the utility of the λ-tree syntax can be found in [Miller, 2000].

- The possibility for hypothetical reasoning (*i.e.,*the presence of implications in the body of clauses) can greatly facilitate certain aspects of this framework. The relational setting offered by, say, Horn clauses [Horn, 1951], which doesn't

allow hypothetical reasoning, is weaker than hereditary Rasiowa-Harrop Formulas [Troelstra and Schwichtenberg, 2000].

3.2 Tailoring the framework

Any implementation of this framework should preferably satisfy these desired properties. However, the generality of the framework can be tailored to specific needs. For example, if one is only interested in propositional logic there is no need to implement higher-order unification. Similarly, if the proof evidence contains the full details of a proof, there might be reduced need for backtracking search. If the proof evidence is given in full (including all instantiations of all quantifiers), then the checker may well be seen as a functional deterministic system instead of a relational non-deterministic system. In fact, section 8.2 shows how one can use this framework to fill in all the missing details of a proof, allowing the resulting proof certificate to be checked on a functional implementation of the framework.

It should be noted that these requirements have strong foundations in logic and have been sufficiently studied and well understood, to the point that one can simply take an off-the-shelf unification algorithm, for example, and implement it in one's language of preference. One can also use failure and success continuations (which are arguably simple to use for a seasoned functional programmer) to implement backtracking in a functional programming language.

By minimizing the amount of theoretical knowledge needed to implement the presented framework, the hope is to have multiple coexisting tools, validating or contradicting each other, all the while augmenting trust and familiarity with Foundational Proof Certification. If one is not convinced of an implementation, a reasonable amount of time should be sufficient to reimplement one's own version of this work. The ultimate goal is that prover implementers will one day see as many benefits in presenting output with precise semantics as programming languages designers see in presenting the semantics of their language in a standard setting such as SOS [Plotkin, 1981].

3.3 Components of the framework

The global architecture is summarized in figure 3.1. The starting point of the proof checking process is a prover outputting a document. This document is called *proof evidence* and is written in the language chosen by the designer of the prover. This language is called the *proof evidence language* or the *proof evidence format* (if unam-

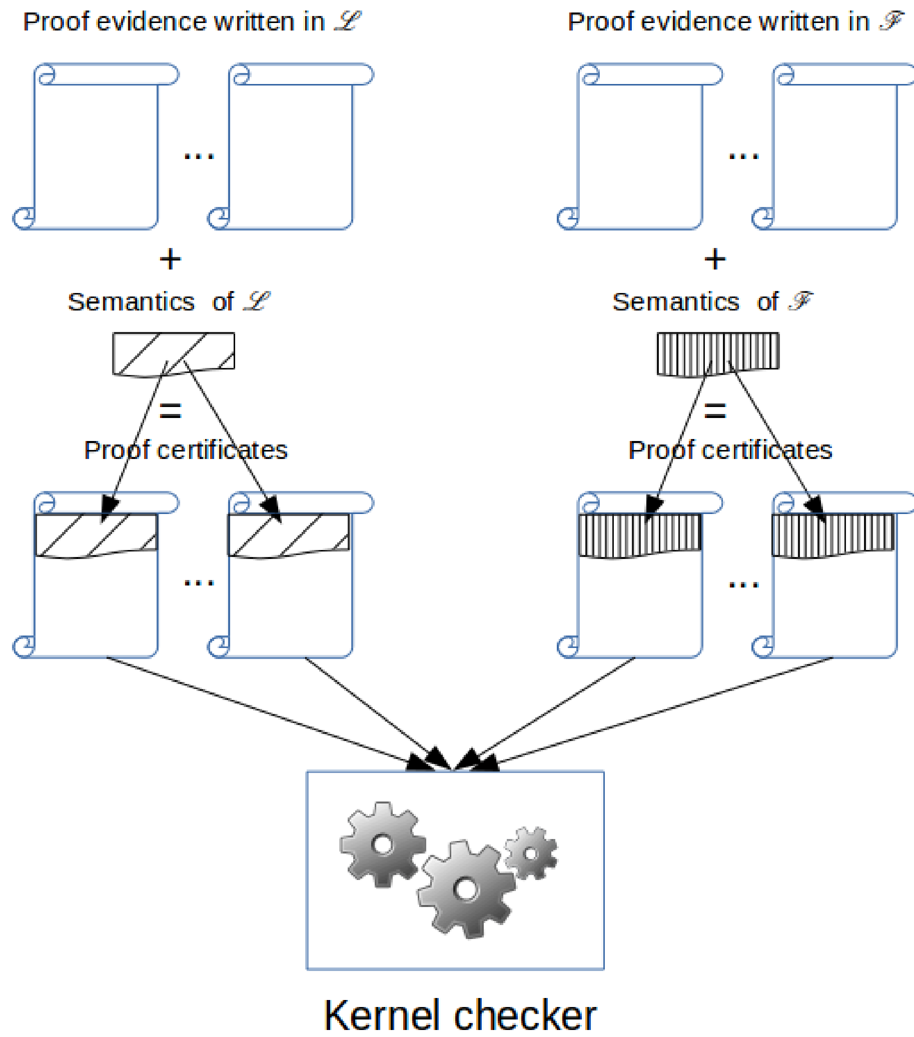


Figure 3.1: Architecture of the framework

biguous from the context, it will be called simply *language* or *format*). The kernel checker has no *a priori* knowledge of it and, therefore, some process is obviously necessary to establish a link between the two.

Instead of *translating* the proof evidence from its original language to one understood by the kernel checker, the Foundational Proof Certification framework gives a way to *describe* the language, or to *define* its *semantics*, by means of a relational specification. Once the semantics of the proof evidence format has been defined, the kernel can then *perform* the proof evidence, much like an interpreter does for a program.

Joining the semantics of a proof evidence language with a piece of proof evidence expressed in that language yields a *proof certificate*.

Remark 3.3.1. The proof evidence, the proof evidence format (or simply: format), the proof evidence format semantics definition (or simply: semantics) and the proof certificate (or simply: certificate) are then four related but distinct notions.

Like any other research effort, the present framework can mature and grow if multiple actors work together towards the same goal of increased communication and improved trust. "Standards can emerge as the consequence of consensus, the imposition of authority, or a combination of both"[Russell, 2014]. The starting belief of this work is that the theorem proving community is ready, if not for a consensus, at least for a serious discussion and for the elaboration of different and competing ideas to address the lack of communication.

The main obstacle to this discussion is an unwillingness to compromise. Proposing a standard that is based on one logic, on one system, or on one notion of proof reduces the space for discussion. On the other extreme, starting from scratch and hoping that the community will find its way towards a natural and all-satisfying standard is Utopian. This project proposes a middle ground by establishing a few guidelines and goals (section 3.1) that constitute an inclusive philosophy and present many opportunities for cooperations, improvements and original ideas.

Another obstacle to the discussion is the unavailability of researchers. Members of the theorem proving community are involved in their own research questions and have little time to participate. And finally, they might not all have (nor have time to learn) the right formal background to understand what is to be done.

This section offers a possible, mildly optimistic division of labor and of required knowledge base between the actors involved in this discussion.

3.3.1 The client's side

The client's workload is the lightest. The client has created a theorem prover and shares the views of de Bruijn in that her theorem prover *does* output some object that indicate some parts of the proving process. This object is what is meant by *proof evidence*.

The client has a specific notion of what the proof evidence should be and would like not to be required to change it too much, typically by having to embed it in another language. She is open to discussion and may agree to small changes (*e.g.,* pretty printing) if she can be convinced that these changes will not take too much of her time and if they do not change the overall aspect of a proof.

For example, a paramodulation-based prover may use clever heuristics or intermediate steps, part of which may be mentioned in the proof evidence. When participating in the Foundational Proof Certification, the client may agree to go so far as to write the proof evidence as paramodulation steps as seen in the original paramodulation paper by Robinson [Robinson and Wos, 1983]. It is an arguably reasonable request for a paramodulation-based prover to output a paramodulation proof. One of the goals of Foundational Proof Certification is to accommodate various formats without requesting radical changes in the user's notion of proof, *e.g.,* translating a paramodulation proof into, say, a natural deduction derivation.

Finally, the client is not required to know anything about any other logic area than her own, but will gladly answer any questions relative to that area.

3.3.2 The kernel checker's side

The Foundational Proof Certification framework is deeply rooted in the theoretical concept of focusing. Designing a kernel checker consists in adding a communication protocol to a focused system (which implies knowledge of that system) in a soundness-preserving way.

Designing the kernel and implementing it are two separate tasks and may be carried out by two actors with separate knowledge bases. The former needs knowledge of focusing, the latter needs experience with implementing an inference based system (not necessarily a focused sequent calculus).

The simpler the framework, the easier it is to implement. This can encourage more researchers to code their own version of the framework.

3.3.3 The middle man: the semantics

The link between a proof evidence language $\mathscr{L}$ and the kernel checker is made through the semantics definition of that language. This semantics is written in terms of relational specifications without particular abilities in a programming language. Establishing this semantics requires general knowledge of both the language $\mathscr{L}$ and the focusing paradigm. In particular, the semantics should be a stand-alone object interpreted by many different and compatible kernels as long as they are based on the relevant logic. Therefore, knowledge of the exact focusing system the kernel is based on should not be required. For example, a proof certificate for linear logic can be checked both by a kernel based on the *MALLF* focused system or on the MALLF fragment of the hybrid *LKU* focused system. A proof certificate for classical logics should be checkable by kernels based on *LKF*, on *LJF* through some double-negation-like translation, or on the *LKF* fragment of the *LKU* system.

Similarly to that of the kernel checker, the definer and implementer of the semantics need not be the same person. However, links between the implementation of the semantics and the implementation of the kernel are necessary. In particular, the interface between the proof certificate and the kernel must be known to both.

3.4 The augmented sequent calculus $\boldsymbol{LKF}^a$

The notions presented here are relative to the last stage of the Foundational Proof Certification process: the actual checking of a proof certificate. It is this proof certificate that supplies the kernel with guiding points so as to perform a correct proof. Because a focused system alternates between invertible phases and focused phases, information guiding the checker is only needed in certain locations. This flow of information must follow a soundness-preserving protocol which is the object of this section.

Definition 3.4.1.

1. *A* case *is a collection of data structures that can be supplied with portions of the proof evidence. Similar to variables in a program, a case can be defined in a number of ways.*

2. *An* agent *is a predicate with a certain number of arguments, some of which are cases.*

3. *An* index *is a label. It can also be defined in a number of ways and is not restricted to being an integer.*

4. *An* indexed formula *is a pair $\langle I, F \rangle$ of formula F and index I*

5. *An* indexed multiset *is a multiset of indexed formulas.*

6. *An* augmented sequent *is the pair of a sequent and a case where the storage is an indexed multiset. (*e.g., *if Θ is an indexed multiset, the (unfocused) sequent $\Xi_1 \vdash \Theta \Uparrow \Gamma$ and the (focused) sequent $\Xi_2 \vdash \Theta \Downarrow P$ are augmented).*

7. *An* augmented inference rule *is an inference rule with an agent as an extra premise. In addition, all sequents appearing in the conclusion and premises are augmented. The agent predicate has at least as many case arguments as the number of sequents in the inference rule (conclusion and premises). These arguments will be the cases associated with these sequents. For example, an inference rule*

$$\frac{premise_1 \quad premise_2}{conclusion}$$

is augmented to yield

$$\frac{\Xi_1\, premise_1 \quad \Xi_2\, premise_2 \quad agent(\Xi_0, \Xi_1, \Xi_2, \cdots)}{\Xi_0\, conclusion}$$

In addition to the case arguments, there can be different numbers of guidance arguments, depending on the agent. Intuitively, the agent examines the "input"[1] *case (here Ξ_0), communicates information, if any, through the rest of its arguments and returns "output" cases (here Ξ_1 and Ξ_2).*

-

Remark 3.4.1. For space saving, when giving an example derivation, augmented inference rules are sometimes written

$$\frac{\Xi_1\, premise_1 \quad \Xi_2\, premise_2}{\Xi_0\, conclusion}\ agent(\Xi_0, \Xi_1, \Xi_2, \cdots)$$

Also for space saving, indexed formulas in the storage, $\langle I, F \rangle$, are sometimes written F_I.

[1]Since predicates are relations, there is no intrinsic input/output distinction. In Logic Programming, however, there is a notion of such data flow.

3.4.1 Experts

During the focused phase the checker requests guiding information. These pieces of information are handed in by agents called *experts*. These agents are in charge of *exploring* and *investigating* the portion of the proof evidence inside the input case in search for guiding information. The name of an expert is the name of its rule with a subscript “e”. The most important experts are given in this section.

Disjunction expert

The positive disjunction rule picks one of the disjuncts and discards the other. The agent of this rule is called the positive disjunction expert, or simply $\vee_e^+$. It can pass a choice in its guidance argument, the augmented $\vee^+$ rule can then be guided to choose the right or the left disjunct.

$$\frac{\Xi_1 \vdash \Theta \Downarrow B_i \qquad \vee_e^+(\Xi_0, \Xi_1, i)}{\Xi_0 \vdash \Theta \Downarrow B_1 \vee^+ B_2}\; i = 1, 2$$

In the presence of backtracking search, this information can be left out of the proof certificate.

Existential expert

The $\exists$ rule instantiates an existential formula with a first-order term s as a witness. This term can be supplied by the existential expert, called $\exists_e$, through the following augmentation of the $\exists$ rule.

$$\frac{\Xi_1 \vdash \Theta \Downarrow [s/x]B \quad \exists_e(\Xi_0, \Xi_1, s)}{\Xi_0 \vdash \Theta \Downarrow \exists x.B}$$

In the presence of a unification mechanism, this guiding information can be left incomplete in various levels of detail:

- the witness s is plainly given to the kernel. In this case it is chosen as a witness and the proof checking will fail if the witness is inadequate
- the proof certificate holds a set of witnesses ω eligible for instantiating this particular existential. The agent succeeds with all of them and, in the presence of backtracking, the kernel checker is given members ω until it reaches one that completes the proof or until all of them fail

- the agent might have no information at all, in which case it succeeds with an unconstrained logic variable. In the presence of unification, this variable is given a satisfactory instantiation, if any exists.

Cut expert

While cut elimination is usually essential for the soundness and completeness of any proof system, the proof checker has little benefit from eliminating the cuts. On the contrary, since many proofs rely on previously proved lemmas, the cut rule is a valuable tool for proof checking. The cut rule is augmented with a cut expert noted $\mathscr{C}_e$:

$$\frac{\Xi_1 \vdash \Theta \Uparrow B \quad \Xi_2 \vdash \Theta \Uparrow \neg B \quad \mathscr{C}_e(\Xi_0, \Xi_1, \Xi_2, B)}{\Xi_0 \vdash \Theta \Uparrow \cdot}$$

Decide expert

The decide rule is augmented with a decide expert, noted $\mathscr{D}_e$, yielding the following rule:

$$\frac{\Xi_1 \vdash \langle l, P \rangle, \Theta \Downarrow P \quad \mathscr{D}_e(\Xi_0, \Xi_1, l)}{\Xi_0 \vdash \langle l, P \rangle, \Theta \Uparrow \cdot} \; decide$$

Because, in a focused system, contraction is the only source of non-termination in (cut-free) proof search, no proof certificate should let the decide rule be applied unrestrained. The guiding information (the index l) can come in different flavors, some of which are given here:

- the proof certificate can give the exact index of the formula on which to decide
- the proof certificate can have a set of indexes on which to decide. As for the existential, backtracking allows trying each of them until one succeeds in finishing the proof
- the kernel receives no guidance, that is the index is an unconstrained logic variable. Then the decide rule can be applied on any formula in the context.

Initial expert

One has to distinguish between checking a proof certificate and checking the provability of a theorem. For the latter, finding one proof is enough. The concern of this

project, however, is checking the proof certificate. An analogy can be made with a school exercise: if a student gives a bad solution to a solvable problem, he should not be rewarded. Similarly, a proof certificate that leads to an incorrect proof should not be accepted, even if a proof actually exists.

Thus the initial rule is also augmented with the initial expert, noted $\mathscr{I}_e$:

$$\frac{\langle l, \neg P_a \rangle \in \Theta \quad \mathscr{I}_e(\Xi_0, l)}{\Xi_0 \vdash \Theta \Downarrow P_a} \; init$$

If a proof certificate gives an index (here l) for the negative atom (here $\neg P_a$) that should be paired with the positive atom under focus (here P_a), this information must be used. It is not sufficient that the initial rule succeeds, it must succeed with the given index.

The information given by $\mathscr{I}_e$ through the index can vary in the same level of details as the information given by the decide expert $\mathscr{D}_e$.

3.4.2 Clerks

During the invertible phase, no guidance is asked (there are no choice points) but the information can flow in the opposite direction, *i.e.,*from the proof-performing kernel to the proof certificate. These opposite-flowing pieces of information are recorded by agents called *clerks* and saved in the case if need be. These agents are in charge of *bookkeeping* steps taken by the kernel checker.

Store clerk

The store rule is augmented with the store clerk, noted $\mathscr{S}_c$:

$$\frac{\Xi_1 \vdash \Theta, \langle l, C \rangle \Uparrow \Gamma \quad \mathscr{S}_c(\Xi_0, \Xi_1, l)}{\Xi_0 \vdash \Theta \Uparrow C, \Gamma} \; store$$

The store clerk is responsible for filing and indexing the formulas entering the context. To do so, it supplies the kernel with an index l.

Universal clerk

Perhaps the best example of the inverse flow of information is the universal clerk, augmenting the $\forall$-rule as follows:

$$\frac{([y/x]\Xi_1) \vdash \Theta \Uparrow [y/x]B, \Gamma \quad \forall_c(\Xi_0, \Xi_1)}{\Xi_0 \vdash \Theta \Uparrow \forall x.B, \Gamma}$$

Notice that the output case and the kernel use the same eigenvariable y.

3.4.3 The complete LKF^a system

Once these *clerks* and *experts* are added as premises to most rules, one obtains the proof system presented in figure 3.2. Notice that by removing all augmentation (in blue), one obtains the *LKF* system. Thus soundness is immediate, because a proof obtained in LKF^a is a proof in *LKF*. More formally:

Definition 3.4.2. *The erasure function* $\|\cdot\|$ *is defined on sequents and formulas as follows.*

$$\|B \circ^{\pm} C\| = \|B\| \circ \|C\| \quad \|\diamond x.B\| = \diamond x.\|B\| \quad \|\blacktriangle^{\pm}\| = \blacktriangle$$

where $\circ \in \{\vee, \wedge\}$, $\pm \in \{+, -\}$, $\diamond \in \{\forall, \exists\}$, $\blacktriangle \in \{f, t, a\}$ *and* a *is an atom.*

$$\|\Xi \vdash \langle l_1, B_1\rangle, \ldots\langle l_n, B_n\rangle \Uparrow C_1, \ldots C_m\| = \ \vdash \|B_1\|, \ldots\|B_n\| \Uparrow \|C_1\|, \ldots\|C_m\|$$

$$\|\Xi \vdash \langle l_1, B_1\rangle, \ldots\langle l_n, B_n\rangle \Downarrow C\| = \ \vdash \|B_1\|, \ldots\|B_n\| \Downarrow \|C\|$$

•

Theorem 3.4.1 (Erasure)**.** *The* LKF^a *system is sound with respect to the LK system.*

Proof. Let π be the proof derivation of the augmented sequent $\Xi \vdash \Gamma \Uparrow \Delta$. Let $\|\pi\|$ be the proof obtained by applying the erasing function to all the sequents in the derivation π. Then $\|\pi\|$ is an *LKF* proof.

It follows that LKF^a is sound with respect to *LKF*, and by the soundness of *LKF*, it is also sound with respect to *LK*. □

The office analogy

Naming the agents *clerks* and *experts* (as opposed to simply *agents*) emphasizes the separation between their respective roles. But the difference between these roles is more than philosophical. Clerks are the instruments of the invertible phase. The phase is sometimes called *asynchronous* because it does not communicate with the outside world, thus clerks do not guide the kernel. However, the clerks can record (hence their name) what happens in that phase as well as perform computations and bookkeep their results in the case object.

The experts, on the other hand, are the instruments of guidance, they directly influence the behavior of the kernel by offering non jeopardizing directions.

Certain agents may seem confusing, in particular, the store agent is a clerk, not an expert, notwithstanding it does provide an index which can be seen as supplying the kernel with information. However, the index holds no meaning to the *kernel*, it only holds meaning to *other agents* (namely the decide experts), as opposed to the existential expert which actually supplies the kernel with the witness. Another way to view this is: by removing all augmentations, the indexes disappear but the witnesses are still part of the proof.

The following analogy might convey the spirit of clerks and experts. Imagine an accounting office that is given a collection of financial documents and tasked, by its client, to check if these documents are in accordance with a certain tax code. The tax office staff is divided into two groups. The first group of workers, called experts, are given the responsibility of looking into the mound and extracting information: they must *decide* into which series of transactions to dig and they need to know when to *release* their findings for later consideration and eventual storage. The second group of workers, called clerks, are responsible for taking information released by the experts and performing various computations on them, including their *indexing* and *storing*. The justification of this division of effort between clerks and experts comes from the structure of focused sequent proof systems: experts operate during the focused phase of proof construction while clerks operate during the invertible phase.

To get an intuitive understanding of the dynamics of this communication, one can imagine a room with a desk on which documents are laid, representing a *case* (as in a piece of work). This case is the *active case*, other cases may be piled on shelves. At each call to a clerk or expert, the requested agent enters the room and approaches the desk. He is not able to communicate with any other agent directly, but he is free to modify the information on the desk for subsequent agents to read. These modifications include adding new information, removing obsolete information, clearing the desk from the former case and taking a different case from a shelf, thus making it the active case.

If an expert is asked a certain guidance by the kernel, he may communicate only to the kernel and only through restricted signals (such as a direction for a disjunction or an index for a decision). If an agent is called into the room and finds a case unknown to him, he declares failure, provoking the failure of the kernel's current investigation. This does not mean failure of the overall checking, however, as the kernel may backtrack to prior choice points and start a new investigation.

3.5 Augmenting the *LJF* system

The LJF^a system (figure 3.3) is an augmented version of *LJF* in the same way LKF^a is an augmented version of *LKF*. As said before, similar augmentations are possible for other focused sequent calculi.

Invertible phase's rules

$$\frac{\Xi_1 \vdash \Theta \Uparrow \Gamma \quad f_c^-(\Xi_0, \Xi_1)}{\Xi_0 \vdash \Theta \Uparrow f^-, \Gamma} \qquad \frac{\Xi_1 \vdash \Theta \Uparrow A, \Gamma \quad \Xi_2 \vdash \Theta \Uparrow B, \Gamma \quad \wedge_c^-(\Xi_0, \Xi_1, \Xi_2)}{\Xi_0 \vdash \Theta \Uparrow A \wedge^- B, \Gamma}$$

$$\frac{\Xi_1 \vdash \Theta \Uparrow A, B, \Gamma \quad \vee_c^-(\Xi_0, \Xi_1)}{\Xi_0 \vdash \Theta \Uparrow A \vee^- B, \Gamma} \qquad \frac{}{\Xi_0 \vdash \Theta \Uparrow t^-, \Gamma} \qquad \frac{(\Xi_1 y) \vdash \Theta \Uparrow [y/x]B, \Gamma \quad \forall_c(\Xi_0, \Xi_1)}{\Xi_0 \vdash \Theta \Uparrow \forall x.B, \Gamma}\ \dagger$$

Focused phase's rules

$$\frac{t_e^+(\Xi_0)}{\Xi_0 \vdash \Theta \Downarrow t^+} \qquad \frac{\Xi_1 \vdash \Theta \Downarrow B_1 \quad \Xi_2 \vdash \Theta \Downarrow B_2 \quad \wedge_e^+(\Xi_0, \Xi_1, \Xi_2)}{\Xi_0 \vdash \Theta \Downarrow B_1 \wedge^+ B_2}$$

$$\frac{\Xi_1 \vdash \Theta \Downarrow B_i \quad \vee_e^+(\Xi_0, \Xi_1, i)}{\Xi_0 \vdash \Theta \Downarrow B_1 \vee^+ B_2} \qquad \frac{\Xi_1 \vdash \Theta \Downarrow [t/x]B \quad \exists_e(\Xi_0, \Xi_1, t)}{\Xi_0 \vdash \Theta \Downarrow \exists x.B}$$

Identity rules

$$\frac{\Xi_1 \vdash \Theta \Uparrow F \quad \Xi_2 \vdash \Theta \Uparrow \neg F \quad \mathscr{C}_e(\Xi_0, \Xi_1, \Xi_2, F)}{\Xi_0 \vdash \Theta \Uparrow \cdot}\ cut \qquad \frac{\langle l, \neg P_a \rangle \in \Theta \quad \mathscr{I}_e(\Xi_0, l)}{\Xi_0 \vdash \Theta \Downarrow P_a}\ init$$

Structural rules

$$\frac{\Xi_1 \vdash \Theta \Uparrow N \quad \mathscr{R}_e(\Xi_0, \Xi_1)}{\Xi_0 \vdash \Theta \Downarrow N}\ release \qquad \frac{\Xi_1 \vdash \Theta \Downarrow P \quad \langle l, P \rangle \in \Theta \quad \mathscr{D}_e(\Xi_0, \Xi_1, l)}{\Xi_0 \vdash \Theta \Uparrow \cdot}\ decide$$

$$\frac{\Xi_1 \vdash \Theta, \langle l, C \rangle \Uparrow \Gamma \quad \mathscr{S}_c(\Xi_0, \Xi_1, l)}{\Xi_0 \vdash \Theta \Uparrow C, \Gamma}\ store$$

Here, P is a positive formula; N a negative formula; P_a a positive atom; N_a a negative atom; C a positive formula or negative atom; Other formulas have arbitrary polarity; s is a first-order term; y is an eigenvariable.

Figure 3.2: The augmented *LKF* proof system LKF^a.

Invertible phase's rules

$$\frac{\Xi_1 : \Gamma \Uparrow A \vdash B \Uparrow \quad \supset_c (\Xi_0, \Xi_1)}{\Xi_0 : \Gamma \Uparrow \ \vdash A \supset B \Uparrow} \supset_r \qquad \frac{(\Xi_1 y) : \Gamma \Uparrow \ \vdash [y/x]B \Uparrow \quad \forall_c(\Xi_0, \Xi_1)}{\Xi_0 : \Gamma \Uparrow \ \vdash \forall x.B \Uparrow} \forall_r$$

$$\frac{\Xi_1 : \Gamma \Uparrow \ \vdash A \Uparrow \quad \Xi_1 : \Gamma \Uparrow \ \vdash B \Uparrow \quad \wedge^-_c(\Xi_0, \Xi_1, \Xi_2)}{\Xi_0 : \Gamma \Uparrow \ \vdash A \wedge^- B \Uparrow} \wedge^-_r \qquad \frac{\wedge^-_c(\Xi_0)}{\Xi_0 : \Gamma \Uparrow \ \vdash t^- \Uparrow} t^-_r$$

$$\frac{\Xi_1 : \Gamma \Uparrow A, B, \Theta \vdash \Delta \quad \wedge^+_c(\Xi_0, \Xi_1)}{\Xi_0 : \Gamma \Uparrow A \wedge^+ B, \Theta \vdash \Delta} \wedge^+_l \qquad \frac{\Xi_1 : \Gamma \Uparrow \Theta \vdash \Delta \quad \wedge^+_c(\Xi_0, \Xi_1)}{\Xi_0 : \Gamma \Uparrow t^+, \Theta \vdash \Delta} t^+_l$$

$$\frac{\Xi_1 : \Gamma \Uparrow A, \Theta \vdash \Delta \quad \Xi_2 : \Gamma \Uparrow B, \Theta \vdash \Delta \quad \vee_c(\Xi_0, \Xi_1, \Xi_2)}{\Xi_0 : \Gamma \Uparrow A \vee B, \Theta \vdash \Delta} \vee_l \qquad \frac{f^+_c(\Xi_0)}{\Xi_0 : \Gamma \Uparrow f^+, \Theta \vdash \Delta} f^+_l$$

$$\frac{(\Xi_1 y) : \Gamma \Uparrow [y/x]B, \Theta \vdash \Delta \quad \exists_c(\Xi_0, \Xi_1)}{\Xi_0 : \Gamma \Uparrow \exists x.B, \Theta \vdash \Delta} \exists_l$$

Focused phase's rules

$$\frac{\Xi_1 : \Gamma \vdash A \Downarrow \quad \Xi_2 : \Gamma \Downarrow B \vdash D \quad \supset_e (\Xi_0, \Xi_1, \Xi_2)}{\Xi_0 : \Gamma \Downarrow A \supset B \vdash D} \supset_l$$

$$\frac{\Xi_1 : \Gamma \vdash A_i \Downarrow \quad \vee_e(\Xi_0, \Xi_1, i)}{\Xi_0 : \Gamma \vdash A_1 \vee A_2 \Downarrow} \vee_r \qquad \frac{\Xi_1 : \Gamma \Downarrow A_i \vdash D \quad \wedge^-_e(\Xi_0, \Xi_1, i)}{\Xi_0 : \Gamma \Downarrow A_1 \wedge^- A_2 \vdash D} \wedge^-_l$$

$$\frac{\Xi_1 : \Gamma \vdash A \Downarrow \quad \Xi_2 : \Gamma \vdash B \Downarrow \quad \wedge^+_e(\Xi_0, \Xi_1, \Xi_2)}{\Xi_0 : \Gamma \vdash A \wedge^+ B \Downarrow} \wedge^+_r \qquad \frac{t^+_e(\Xi_0)}{\Xi_0 : \Gamma \vdash t^+ \Downarrow} t^+_r$$

$$\frac{\Xi_1 : \Gamma \vdash [t/x]B \Downarrow \quad \exists_e(\Xi_0, \Xi_1, t)}{\Xi_0 : \Gamma \vdash \exists x.B \Downarrow} \exists_r \qquad \frac{\Xi_1 : \Gamma \Downarrow [t/x]B \vdash D \quad \forall_e(\Xi_0, \Xi_1, t)}{\Xi_0 : \Gamma \Downarrow \forall x.B \vdash D} \forall_l$$

Identity & structural rules

$$\frac{N_a \text{ atomic} \quad \mathbb{I}^l_e(\Xi_0)}{\Xi_0 : \Gamma \Downarrow N_a \vdash N_a} I_l \qquad \frac{P_a \text{ atomic} \quad (l, P_a) \in \Gamma \quad \mathbb{I}^r_e(\Xi_0, l)}{\Xi_0 : \Gamma \vdash P_a \Downarrow} I_r$$

$$\frac{\Xi_1 : \Gamma \Uparrow \ \vdash F \Uparrow \quad \Xi_2 : \Gamma \Uparrow F \vdash \ \Uparrow R \quad \mathbb{C}_e(\Xi_0, \Xi_1, \Xi_2, F)}{\Xi_0 : \Gamma \Uparrow \ \vdash \ \Uparrow R} Cut$$

$$\frac{\langle l, N \rangle \in \Gamma \quad \Xi_1 : \Gamma \Downarrow N \vdash R \quad \mathbb{D}^l_e(\Xi_0, \Xi_1, l)}{\Xi_0 : \Gamma \Uparrow \ \vdash \ \Uparrow R} D_l \qquad \frac{\Xi_1 : \Gamma \vdash P \Downarrow \quad \mathbb{D}^r_e(\Xi_0, \Xi_1)}{\Xi_0 : \Gamma \Uparrow \ \vdash \ \Uparrow P} D_r$$

$$\frac{\Xi_1 : \Gamma \Uparrow P \vdash \ \Uparrow R \quad \mathbb{R}^l_e(\Xi_0, \Xi_1)}{\Xi_0 : \Gamma \Downarrow P \vdash R} R_l \qquad \frac{\Xi_1 : \Gamma \Uparrow \ \vdash N \Uparrow \quad \mathbb{R}^r_e(\Xi_0, \Xi_1)}{\Xi_0 : \Gamma \vdash N \Downarrow} R_r$$

$$\frac{\Xi_1 : \langle l, C \rangle, \Gamma \Uparrow \Theta \vdash \Delta \quad \mathbb{S}^l_c(\Xi_0, C, \Xi_1, l)}{\Xi_0 : \Gamma \Uparrow C, \Theta \vdash \Delta} S_l \qquad \frac{\Xi_1 : \Gamma \Uparrow \ \vdash \ \Uparrow D \quad \mathbb{S}^r_c(\Xi_0, D, \Xi_1)}{\Xi_0 : \Gamma \Uparrow \ \vdash D \Uparrow} S_r$$

Here, P is a positive formula; N a negative formula; P_a a positive atom; N_a a negative atom; D a positive formula or negative atom; C a negative formula or positive atom; Other formulas have arbitrary polarity; s is a first-order term; y is an eigenvariable.

Figure 3.3: The augmented intuitionistic sequent calculus LJF^a.

Chapter 4

Foundational Proof Certification

The previous chapter outlined the general architecture of the kernel checker consisting of augmenting focused systems with an interface shared with proof certificates. After seeing the kernel component of the Foundational Proof Certification framework, this chapter describes the middle-man's workload: establishing a semantics for a given language. This is done through adjustment of five parameters: a *polarity* assignment of the formulas, a *region* delimitation of the reconstructed proof along with constructors for *case* objects, an *indexing* mechanism for the stored formulas, and the definitions of the *clerks* and *experts*.

The adjustment of these five axes, abbreviated as p.r.i.c.e., is described informally in the first section. Some guidelines and intuition are given for the adjustment of each of these five parameters.

Then section 4.2 describes most of these parameters in a more formal fashion, introducing types, signature and syntax for the definition of predicates.

Finally, the last section gives several examples of p.r.i.c.e. fragments consisting of cases and agents able to complete a small task. These p.r.i.c.e. fragments are referenced in later chapters and can be modularly used, alongside other p.r.i.c.e., in the definition of semantics.

4.1 Descriptive semantics

The last stage of the Foundational Proof Certification process, the actual checking, was the focus of the previous chapter. This chapter takes a step back and deals with the middle stage: establishing the semantics of the proof evidence language.

This stage can be described as a jigsaw puzzle with three pieces:

- a formula T that a prover Ω claims to be a theorem;

- a piece of proof evidence Φ_Ω, written in language $\mathscr{L}_\Omega$, which is given by the prover Ω to justify its claim that T is a theorem;
- a kernel checker able to reconstruct a proof of formula T when provided with a guideline written in Φ_Ω.

The semantics of $\mathscr{L}_\Omega$ is what holds these three elements together. The middleman solves the puzzle by describing $\mathscr{L}_\Omega$ in a well-defined, flexible and relational setting such that a kernel can interpret any proof evidence written in that language.

There are several ways to win this game because the semantics of a language and a p.r.i.c.e. definition are not the same thing. The former is an abstract notion while the latter is the concrete relational specification of the former. Indeed, there can be many descriptions of the same proof evidence language, just as there can be more than one algorithm to solve a given problem. And just as one can judge the algorithm both objectively (what is its complexity?) and subjectively (is it "elegant"?), one can also judge the description of semantics.

Guidelines are given in what follows to help establishing the semantics but a considerable amount of the judgment in value of a semantics definition is ultimately a matter of taste.

These five axes of p.r.i.c.e. should be thought of as a whole. They are comparable to components of a program: it often happens that a piece of code is repeated, then a new procedure is created and the repeated pieces of code are replaced by calls to that procedure. Similarly, a variable can become needed while writing the code, it is then declared and initialized, usually at the start of the program. This is why, albeit introduced below in a certain order, adjusting the five axes is done in a non-linear fashion.

4.1.1 Polarity assignment

In general, theorem provers do not use a polarized formula syntax[1]. The first step in checking the proofs is thus a choice of polarization of the different connectives. Polarity assignment of a formula F is made through the definition of the function $[\![F]\!]^{\pm}$. This function will be encoded as a relation.

As mentioned in section 2.2, polarities are nothing more than annotations indicating, for a given connective, during which phase it can be introduced. But how are these polarities chosen? What makes one polarity better than the other for a particular connective? There is no clear-cut rule on how to assign polarities. This

[1] Some do: Maetning, Psyche, Tac...

step of the definition of semantics is determined on a case-by-case basis. A clear visualization of the shape of reconstructed proofs is the starting point of the semantics definition and the most important part of this definition is the polarity assignment. Indeed, the clerks being the instruments of the invertible phase and the experts being instruments of the focused phase, assigning one polarity or the other to a connective already determines which agent would be in charge of it. The following points provide guidance and intuition. No p.r.i.c.e. can be defined without taking the following questions into consideration, along and in concordance with the other four axes of the semantics definition. However, it is not uncommon that the same answer to one of these questions leads to different choices of polarity, depending on the other parameters.

Determinacy: What part of the proof reconstruction process is deterministic and what part is open to guidance? As mentioned in section 2, an invertible phase can be seen as a deterministic computation while a focused phase can be taken as deduction. If this distinction is clear at the level of the prover, the polarity assignment can reflect this separation.

Are both disjuncts or conjuncts needed? The positive disjunction (on the right) and the negative conjunction (on the left) have introduction rules that discard one of the subformulas of the principal formula. Thus the polarity can be used for such management.

Availability of information: Will the proof evidence hold information regarding this connective? Because the kernel gets information only through experts, the polarity of connectives should be chosen accordingly. A connective's agent is strongly linked to its polarity, *i.e.,*if its introduction rule is focused its agent is an expert; if its introduction rule is invertible its agent is a clerk.

Contraction: Will multiple copies of this formula be needed? The focusing discipline explained in section 2.2.1 makes it so that the polarity of a formula determines its contractibility. Only positive formulas can be contracted on the right (in a classical sequent calculus) and only negative formulas can be contracted on the left (in an intuitionistic sequent calculus).

Side of appearance: In case of a two-sided sequent, will this (sub)formula appear on the left hand-side or on the right hand-side of a sequent? The contractibility and

invertibility of a formula of a given polarity are symmetric, *e.g.,* a positive formula belongs to the focused phase (thus is contractible) if it is on the right but belongs to the invertible phase if it is on the left.

Atoms' polarity: The polarity of atoms can sometimes distinguish between very different systems. For example, different polarity of atoms in the *LJF* system are the essential difference between the LJQ [Dyckhoff and Lengrand, 2006] and LJT [Herbelin, 1995b] systems.

Taming the focusing: The focusing discipline of *LKF* and *LJF* can be too rigid to flexibly implement the ideas of the middle-man (the section 5.4 on Expansion Trees shows such a situation). In those cases, delays can be inserted in front of subformulas to force the kernel to postpone introduction of these subformula to one phase or the other, allowing the middle-man to bend the focusing behavior at will.

The bigger picture: The polarity of a connective can also be chosen relative to the surrounding formula. When the overall structure of the reconstructed proof is visualized, it is stratified in focused and invertible phases. For example, if a formula containing, say, a conjunction is desired to appear in a given phase, that conjunction must be given the appropriate polarity. In particular, if the formula $\forall x.B \wedge \forall y.A$ should be introduced in one phase, then the conjunction must have the same polarity as the $\forall$ quantifier (*i.e.,*a negative one) to avoid an unwanted phase break. Similarly, if the formula $\exists x.B \wedge \exists y.C$ should be introduced in one phase, the conjunction must take a positive polarity.

4.1.2 Region delimitation

Because the agents are additional premises separate from the rest of the proof derivation to preserve soundness (see the Erasure theorem 3.4.1), they are also isolated from one another. Their sole means of communication is the added case (described in definition 3.4.1) defined by the middle-man.

These cases being the means of communication of the agents, they have to be defined at the same time as those agents. Similar to the polarity assignment, defining the case is only possible when the overall structure of the reconstructed proof is visualized. This structure can sometimes be separated in *regions* corresponding to different processes in, or different steps of, the proof checking.

For each of these regions, a case will be assigned, containing all information relevant to that region and a team of clerks and experts will be defined on that

region and will work on that case. These regions sometimes correspond to phases but they need not do so.

Practical examples will be given in following chapters. Here again, defining the cases for a proof evidence language semantics is comparable to defining the data structures for a given program: there can be many ways of arriving at a solution and these solutions are evaluated according to the same objective and subjective criteria.

4.1.3 Indexing

There is a duality between the store clerk that generates an index to store a formula and the decide expert that choses an index of a formula on which to engage the focused phase. An index can be:

- unique for each formula: this offers the possibility of having the indexing be a function;
- shared between some formulas: in this case an index denotes a set of formulas. If a decide expert gives the kernel that index, the kernel can engage a focused phase on any of the associated formulas while being able to backtrack if a particular choice leads to a failure;
- the same for all formulas: in this case, an indexed context is equivalent to a non indexed context. This relieves the decide expert of all responsibility regarding the choice of contracted formula.

An index can be generated by the store clerk based on earlier information or it can be present in the proof evidence from the beginning. In any case, the middle-man defines the indexes.

Remark 4.1.1. The above may seem repetitive to the levels of detail in guidance information given to the decide expert in section 3.4.1. However, there is a subtle difference: the above is a discussion on how indexes are defined while section 3.4.1 discusses how experts are defined.

4.1.4 Clerks & Experts

The agents, seen at length in the presentation of augmented systems (section 3.4) are central to the semantics definition. They are defined on the cases and, for some of them, on indexes, and their workload is partly determined by polarity assignment, *e.g.,* if a connective is given a negative polarity and appears on the right, it is handled

by a clerk. An agent relates not only to the rest of the components of the `p.r.i.c.e.` but also to other agents. Clerks and experts are the only link between the kernel and proof certificates.

When an agent appears as the extra premise of an augmented inference rule that is part of a certain delimited region R of a proof, it is said to *inhabit* that region, in other words, it has at least one definition with a case Ξ_R associated with R as its first argument. The set of agent definitions inhabiting a region is called the *team* of that region. An agent definition is said to be *naïve* if its output case is the same as its input case, and if it gives no guiding information. For example, an existential expert defined as $\exists_e(\Xi, \Xi, V)$ where V is a non-constrained logic variable is naïve.

Remark 4.1.2. The decide expert is generally not naïve because of the danger of non-termination. It can be, however, be naïve, if no release can be done after the decide rule augmented with that expert. This happens, in particular, when the only formulas in the context are strictly positive, implying that any decide rule will be followed by a unique focused phase with no release rule, thus no bipole.

If, in a proof, a region R_1 follows a region R_2, then the active case must be changed. This is done through an agent inhabiting R_1 that outputs a case relative to R_2 so that the following team of agents continue the investigation on their respective region.

4.2 Programmable semantics

All previous mentions of semantics description focused on relational specifications without considering their operational interpretation. However, from a pragmatic standpoint, one has to be aware of the underlying computational behavior necessary to implement the proof checking process [Chihani et al., 2013b]. Logic programming is a *relational* approach to programming, where computation is carried through a search process, following strict strategies that the programmer is made aware of so as to predict the behavior of the interpretation engine on his or her specification.

Aside from this section, this thesis voluntarily omits discussing such aspects for the following reasons. First, describing this framework in a chosen programming language puts the reader in a state of mind where all parts of that programming language are considered. In particular, non-logical artifacts such as the Prolog cut (or bang, noted "!") and input/output operations from a file may be understood as acceptable machinery for the Foundational Proof Certification framework. This is far from the original intention of the work described in this thesis. Indeed, the goal is to achieve the existence of proof certificates independently from any tech-

nology. To do so, semantics descriptions should only be expressed in logic so as to be self-contained. Any implementer of this framework *can* define experts to simply use a `readLine` command to appeal to the user, establishing an *interactive* proof checking tool, but such possibilities are not considered in this thesis. Similarly, any serious implementation of this framework should provide means to access libraries of theorems, such that a proof certificate can use a URL to access previously certified theorems, but again, this is not the main subject of the thesis.

A second reason for omitting the discussion of programming paradigm is that any description of a framework benefits from being as general and as far from implementation questions as possible. Consider, for example, that functions are special kinds of relations where some of the arguments are uniquely determined by others. If semantics is written so as to remove any non-determinacy or, in programming language terminology, backtrack points, then it is possible to define every part of the framework presented in this thesis using a functional programming language. Thus keeping the discussion in as general a setting as possible offers a global view of the Foundational Proof Certification framework. For example, consider the example of the relation between a list L and its length N. The description of this relation is simply written $\mathit{length}\, L\, N$. This relation is assumed satisfied with the empty list and the integer 0, with all singleton lists and the number 1 and with all lists of N elements and the integer N. Stating this relation is distinguished from its possible implementations, some of which are shown in figure 4.1, because such implementations depend on the choice of programming paradigm and language.

Nonetheless, the unconcealed aim of this project is, ultimately, implementations of the framework. For this reason, the following sections go into as much detail as possible without assuming any implementation choices beyond the desired properties seen in section 3.1.

4.2.1 Type signature

The Foundational Proof Certification framework uses a typed syntax, *i.e.,* each predicate is declared with its own type. The type of terms is ι, the type of propositions is o while the constructor of arrow types is noted $\rightarrow$. The agents, which are the bridges between the kernel and the proof certificate, have global types known to these two components. The types are declared using the keyword `type`, taking as first element the name of a predicate and as second its type. Taking the augmented classical focused sequent calculus, for instance, the type of the existential expert is:

$$\texttt{type} \quad \exists_e \quad \texttt{case} \rightarrow \texttt{case} \rightarrow \iota \rightarrow o$$

```
let length list =
  let rec aux n = function
    | [] -> n
    | _::t -> aux (n+1) t
  in aux 0 list;;
val length : 'a list -> int = <fun>
```

(a) Implementation in Ocaml

```
int length(struct node *head){
    int num = 0;
    while (head != NULL)    {
        num += 1;
        head = head->next;
    }
    return num;
}
```

(b) Implementation in C

```
type length list A -> int -> o.
length nil 0.
length [_|T] I :-
   length T I',
   I is I' + 1.
```

(c) Implementation in λProlog

Figure 4.1: Different implementations of length.

`type`	$\wedge_c^-$	$\texttt{case} \to \texttt{case} \to \texttt{case} \to o$
`type`	$\vee_c^-$	$\texttt{case} \to \texttt{case} \to o$
`type`	$\wedge_e^+$	$\texttt{case} \to \texttt{case} \to \texttt{case} \to o$
`type`	$\vee_e^+$	$\texttt{case} \to \texttt{case} \to \delta \to o$
`type`	$\exists_e$	$\texttt{case} \to \texttt{case} \to \iota \to o$
`type`	$\forall_c$	$\texttt{case} \to (\iota \to \texttt{case}) \to o$
`type`	t_e^+	$\texttt{case} \to o$
`type`	f_c^-	$\texttt{case} \to \texttt{case} \to o$
`type`	$\mathscr{S}_c$	$\texttt{case} \to \texttt{fm} \to \texttt{case} \to \xi \to o$
`type`	$\mathscr{D}_e$	$\texttt{case} \to \texttt{case} \to \xi \to o$
`type`	$\mathscr{R}_e$	$\texttt{case} \to \texttt{case} \to o$
`type`	$\mathscr{I}_e$	$\texttt{case} \to \xi \to o$
`type`	$\mathscr{C}_e$	$\texttt{case} \to \texttt{case} \to \texttt{case} \to \texttt{fm} \to o$

Figure 4.2: LKF^a Agents' type declarations

where `case` is the type of cases. The type of the store clerk $\mathscr{S}_c$ is:

$$\texttt{type} \quad \mathscr{S}_c \quad \texttt{case} \to \texttt{fm} \to \texttt{case} \to \xi \to o$$

where ξ is the type of indexes and `fm` is the type of (object-level) focused formulas (which is different from the type o). The type of the universal clerk $\forall_c$ is:

$$\texttt{type} \quad \forall_c \quad \texttt{case} \to (\iota \to \texttt{case}) \to o$$

where the second argument is of a higher-order type. The type of the disjunction expert $\vee_e^+$ is:

$$\texttt{type} \quad \vee_e^+ \quad \texttt{case} \to \texttt{case} \to \delta \to o$$

where δ is the type of directions with only two inhabitants: `left` and `right`. The complete list of type declarations for agents is given in figure 4.2 for the LKF^a system and in figure 4.3 for the LJF^a system.

These type declarations are part of a signature, denoted by Σ. Additionally, the signature contains type declarations for constructors of objects of type `case` and of

type	$\supset_c$	$\texttt{case} \to \texttt{case} \to o$
type	$\supset_e$	$\texttt{case} \to \texttt{case} \to \texttt{case} \to o$
type	$\vee_c$	$\texttt{case} \to \texttt{case} \to \texttt{case} \to o$
type	$\vee_e$	$\texttt{case} \to \texttt{case} \to \delta \to o$
type	$\wedge_c^+$	$\texttt{case} \to \texttt{case} \to o$
type	$\wedge_e^+$	$\texttt{case} \to \texttt{case} \to \texttt{case} \to o$
type	$\wedge_c^-$	$\texttt{case} \to \texttt{case} \to \texttt{case} \to o$
type	$\wedge_e^-$	$\texttt{case} \to \texttt{case} \to \delta \to o$
type	$\exists_c$	$\texttt{case} \to (\iota \to \texttt{case}) \to o$
type	$\exists_e$	$\texttt{case} \to \texttt{case} \to \iota \to o$
type	$\forall_c$	$\texttt{case} \to (\iota \to \texttt{case}) \to o$
type	$\forall_e$	$\texttt{case} \to \texttt{case} \to \iota \to o$
type	t_c	$\texttt{case} \to \texttt{case} \to o$
type	t_e	$\texttt{case} \to o$
type	f_c	$\texttt{case} \to o$
type	$\mathbb{I}_e^l$	$\texttt{case} \to o$
type	$\mathbb{R}_e^l$	$\texttt{case} \to \texttt{case} \to o$
type	$\mathbb{D}_e^l$	$\texttt{case} \to \texttt{case} \to \xi \to o$
type	$\mathbb{S}_c^l$	$\texttt{case} \to \texttt{fm} \to \texttt{case} \to \xi \to o$
type	$\mathbb{I}_e^r$	$\texttt{case} \to \xi \to o$
type	$\mathbb{R}_e^r$	$\texttt{case} \to \texttt{case} \to o$
type	$\mathbb{D}_e^r$	$\texttt{case} \to \texttt{case} \to o$
type	$\mathbb{S}_c^r$	$\texttt{case} \to \texttt{fm} \to \texttt{case} \to o$

Figure 4.3: LJF^a Agents' type declarations

type ξ as well as type declarations for any extra predicates needed by the middle-man. For example, if the proof evidence contains identifiers of formulas in the form of integers, one can declare an index constructor `idx` of type:

$$\texttt{type}\quad\texttt{idx}\quad \mathbb{N} \rightarrow \xi$$

where $\mathbb{N}$ denotes the type of integers. If the proof evidence contains a list of indexes which it is useful to keep track of (to restrict the number of contractions on them, for example), one can define a case constructor `track` of type:

$$\texttt{type}\quad\texttt{track}\quad\texttt{list}\ \xi \rightarrow \texttt{case}$$

where `list A` is the type of lists of elements of type `A`.

In what follows, the signature is omitted and type checking of all objects (terms, formulas, cases, etc) is assumed.

4.2.2 Predicate definitions

In addition to type declarations in the signature, predicate agents (and any extra predicates the middle-man deems necessary) are given a relational specification in the form of Horn clauses, noted $\mathcal{R}$. These type of formulas are also called H-formulas and follow the grammar:

$$G ::= A \mid G, G \qquad H ::= A \mid A \ \texttt{:-}\ G \mid \forall x.H$$

where the comma denotes conjunction, $A\ \texttt{:-}\ G$ is a reversed notation for the implication formula $G \supset A$ of which A is called the *head* and G is called the *body*, A is an atomic formula of type o and both the comma and the reversed implication `:-` are of type $o \rightarrow o \rightarrow o$. In this thesis, universal quantification of H-formulas over variables is left implicit when they start with a capital letter.

In addition, some constructors of terms are assumed to be defined, in particular the (polymorphic) constructor for lists of elements of type τ:

$$\texttt{type}\quad(\texttt{infix})\quad ::\quad \tau \rightarrow \texttt{list}\ \ \tau \rightarrow \texttt{list}\ \ \tau.$$

where the `infix` keyword indicates the fixity of the :: operator, and the constructor for the empty list:

$$\texttt{type}\quad \mathit{nil}\quad \texttt{list}\ \ \tau.$$

The syntax with square brackets, common in many programming languages is adopted: [] for *nil*, $[e_1, \cdots, e_n]$ for the list of elements $e_1, \cdots, e_n$ and $[e_1, \cdots, e_n|L]$ for the list with elements $e_1, \cdots, e_n$ followed by the elements of list L.

The underscore symbol _ is used to denote a variable that is implicitly universally quantified over the entire formula but is not used anywhere else. The occurrences of the underscore symbol, or *wildcard*, all denote distinct variables.

As an example of predicate definition, the following disjunction expert $\vee_e^+$ is defined on the case constructor:

$$\texttt{type}\ \ \texttt{path}\ \ \texttt{list}\ \ \delta \to \texttt{case}.$$

This case contains a sequence of right and left steps (of type δ) that the expert communicates to the kernel in order:

$$\vee_e^+(\texttt{path}\ [D|\Delta], \texttt{path}\ \Delta, D).$$

Omitting the definition of an agent $\mathcal{A}$ on a given case Ξ nullifies it on Ξ. This has the consequence of also nullifying the rules augmented with $\mathcal{A}$ when the conclusion is a sequent augmented with Ξ. For example, if a relational specification does not mention the cut expert $\mathscr{C}_e$, the kernel will only build cut-free derivations. Furthermore, the *LKF* and *LJF* systems are mostly syntax directed, *i.e.,* a sequent can be the conclusion of only one inference rule, with the exception of the sequents $\vdash \Theta \Uparrow \cdot$ for *LKF* and $\Gamma \Uparrow\ \vdash\ \Uparrow R$ for *LJF*. These sequents are conclusions of both the decide and the cut rules. In the augmented systems, this ambiguity can be solved by defining only one of the experts ($\mathscr{C}_e$ and $\mathscr{D}_e$ for LKF^a, $\mathbb{C}_e$, $\mathbb{D}_e^l$ or $\mathbb{D}_e^r$ for LJF^a).

4.3 Default teams of agents

In chapters 5 and 6, several examples of semantics definitions are given for classical and intuitionistic first-order logics. Parts of these semantics definitions use the same modularly defined p.r.i.c.e. fragments. These fragments, and more, are given here for LKF^a. The definition for LJF^a is similar.

One can read the following predefined teams of agents or postpone their reading until they are referenced in one of the upcoming chapters so that they are placed in a relevant context.

4.3.1 The `done` team

The `done` case constructor is relative to the last regions in a proof derivation, the ones ending with either an initial, t^+, or t^- rule. In other words, no decide or cut agents inhabit these final regions and all definitions of other agents are naïve. They can be seen in figure 4.4.

$\wedge_c^-(\texttt{done}, \texttt{done}, \texttt{done})$	$\vee_c^-(\texttt{done}, \texttt{done})$	$\wedge_e^+(\texttt{done}, \texttt{done}, \texttt{done})$
$\vee_e^+(\texttt{done}, \texttt{done}, C)$	$\exists_e(\texttt{done}, \texttt{done}, W)$	$\forall_c(\texttt{done}, \lambda x.\texttt{done})$
$t_e^+(\texttt{done})$	$f_c^-(\texttt{done}, \texttt{done})$	$\mathscr{S}_c(\texttt{done}, F, \texttt{done}, I)$
$\mathscr{R}_e(\texttt{done}, \texttt{done})$	$\mathscr{I}_e(\texttt{done}, I)$	

Figure 4.4: The definitions of the `done` team of agents.

A similar team is defined on case:

$$\texttt{type}\ \ \texttt{initWith}\ \ \texttt{list}\ \xi \rightarrow \texttt{case}.$$

The difference is that the final region of the proof is focused and ends with an initial rule using one of the indexes contained in the argument of `initWith`. This means that, unlike the team in figure 4.4, definitions of all clerks and the t_e^+ are omitted and the $\mathscr{I}_e$ definition is no longer naïve, as it succeeds only on the given indexes:

$$\mathscr{I}_e(\texttt{initWith}\ \Delta, I)\ \ \texttt{:-}\ \ I \in \Delta.$$

4.3.2 The `oneOf` team

The case:

$$\texttt{type}\ \ \texttt{oneOf}\ \ \texttt{list}\ \xi \rightarrow \texttt{case} \rightarrow \texttt{case}.$$

is relative to a region where the decomposition occurring in an invertible phase is not recorded. At the end of this invertible phase, a decision is made on one of the indexes in the first argument of the `oneOf` case, then the active case is changed to the one that is the second argument of `oneOf`, which is a continuation case. When called in a store rule, the store clerk defined on `oneOf` $_\Xi$ calls its homologue defined on the continuation case Ξ. The resulting team is shown in figure 4.5.

4.3.3 The `witness` case

This case constructor, defined as:

$$\texttt{type}\ \ \texttt{witness}\ \ \texttt{list}\ \iota \rightarrow \texttt{case} \rightarrow \texttt{case}.$$

is relative to a small region inhabited only by the existential expert $\exists_e$ that succeeds on one of the terms contained in the first argument of `witness`, then changes the

$$\wedge_c^-(\texttt{oneOf}\ \ \Delta\,\Xi, \texttt{oneOf}\ \ \Delta\,\Xi, \texttt{oneOf}\ \ \Delta\,\Xi) \qquad \vee_c^-(\texttt{oneOf}\ \ \Delta\,\Xi, \texttt{oneOf}\ \ \Delta\,\Xi)$$
$$\forall_c(\texttt{oneOf}\ \ \Delta\,\Xi, \lambda x.\texttt{oneOf}\ \ \Delta\,\Xi) \qquad f_c^-(\texttt{oneOf}\ \ \Delta\,\Xi, \texttt{oneOf}\ \ \Delta\,\Xi)$$
$$\mathscr{S}_c(\texttt{oneOf}\ \ \Delta\,\Xi, F, \texttt{oneOf}\ \ \Delta\,\Xi', I) \quad \texttt{:-} \quad \mathscr{S}_c(\Xi, F, \Xi', I)$$
$$\mathscr{D}_e(\texttt{oneOf}\ \ \Delta\,\Xi, \Xi, I) \quad \texttt{:-} \quad I \in \Delta$$

Figure 4.5: The definitions of the `oneOf` team of agents.

active case to the second argument of `witness`.

$$\exists_e(\texttt{witness}\ \ \Upsilon\,\Xi, \Xi, T) \quad \texttt{:-} \quad T \in \Upsilon.$$

4.3.4 The `tag` case

This case constructor is defined as:

$$\texttt{type}\ \ \ \texttt{tag}\ \ \ \xi \to \texttt{case} \to \texttt{case}.$$

and all agents (except the decide expert) are naïvely defined on it except the store clerk $\mathscr{S}_c$. The latter gives the index in its first argument to the kernel (regardless of the formula) then gives the continuation case in its second argument as output.

$$\mathscr{S}_c(\texttt{tag}I\Xi, _, I, \Xi).$$

Chapter 5

FPC for classical logic

For the things we have to learn before we can do them, we learn by doing them.

- Aristotle *The Nicomachean Ethics*

Following the guidelines of chapter 4 and within the framework seen in chapter 3, this chapter gives actual case studies of semantics definitions for classical proof formats, starting with a simple decision procedure in section 5.1, intended as a gentle introduction.

This chapter also contains the most detailed semantics definition of the thesis in section 5.3. The adjustment of the components of the p.r.i.c.e. is discussed in length, including all possible polarity assignment choices. This is done with the objective of detailing the notions that were only informally introduced in section 4.1. The following semantics definitions, however, are treated in a more straightforward manner.

5.1 The *CNF* decision procedure

Although the Foundational Proof Certification framework is centered around checking proof certificates, by joining some proof evidence expressed in a language with the semantics definition of that language, it can also be used to specify decision procedures, which can be seen as degenerate proof certificates since no proof evidence is given. This section illustrates such a specification with the belief that it eases understanding of the following (and more complicated) semantics definitions.

5.1.1 Preliminaries

Definition 5.1.1. *A* literal *is an atomic formula or the negation of an atomic formula. A* clause *is a disjunction of literals. A propositional formula with no implication ($\supset$) is in* conjunctive normal form *if it is a conjunction of clauses.* •

Any formula F can be put in its classically equivalent conjunctive normal form *cnf(F)* following these classically valid formula transformations:

1. compute the negation normal form of F
2. distribute all disjunctions over factored conjunctions

$$A \vee (B \wedge C) \rightarrow (A \vee B) \wedge (A \vee C)$$

A formula in conjunctive normal form is a tautology if there is a complementary pair of literals (*i.e.,*a and $\neg a$ for some atomic formula a) in every clause. Thus one can define the following *CNFdec* decision procedure for propositional classical logic:

Definition 5.1.2. *The decision procedure* CNFdec *consists of two steps:*

1. *compute the conjunctive normal form of the given formula*
2. *if each resulting clause contains a pair of complementary literals, the formula is a tautology.*

•

5.1.2 *CNFdec* in the *FPC* framework

The difference between using the Foundational Proof Certification framework for defining semantics and using it for specifying a decision procedure is that, for the latter, there is neither proof evidence to check nor language to describe. This simplifies the definition of `p.r.i.c.e.` and constitutes an ideal introduction to this new approach to proof certification.

One first has to interpret the *CNFdec* decision procedure on a formula F through an *LKF* derivation of the sequent $\vdash \cdot \Uparrow [\![F]\!]^{\pm}$, then define a `p.r.i.c.e.` so as to guide the kernel to such a derivation. Fortunately, the invertible phase of *LKF* can be seen as a conjunctive normal form computation where each clause contains either literals or positive formulas.

To make this computation clear, recall the invertible conjunction rule:

$$\frac{\vdash \Theta \Uparrow A, \Gamma \qquad \vdash \Theta \Uparrow B, \Gamma}{\vdash \Theta \Uparrow A \wedge^- B, \Gamma}$$

Because the comma is the sequent-level disjunction, reading this rule bottom-up moves from attempting to prove the sequent $\vdash \Theta \Uparrow A \wedge^- B, \Gamma$, which has the same validity as the formula $(A \wedge^- B) \vee^- ((\bigvee^- \Theta) \vee^- (\bigvee^- \Gamma))$, to attempting to prove the (meta) conjunction of the sequents $\vdash \Theta \Uparrow A, \Gamma$ and $\vdash \Theta \Uparrow B, \Gamma$, respectively having the same validity as the formulas $A \vee^- ((\bigvee^- \Theta) \vee^- (\bigvee^- \Gamma))$ and $B \vee^- ((\bigvee^- \Theta) \vee^- (\bigvee^- \Gamma))$. Thus the conjunction rule can be seen as distributing the sequent-level disjunction over the formula-level conjunction.

Hence, to compute the conjunctive normal form of a propositional formula F, the negative phase should stop at no positive formula except the atoms. Thus the negative versions of both binary connectives is chosen, while atoms can have arbitrary polarity.

Definition 5.1.3. *The multiset of literals of a formula F is denoted $\Lambda(F)$ and defined as:*

$$\Lambda(\star) = \{\star\} \quad \Lambda(A \circ B) = \Lambda(A) \cup \Lambda(B) \quad \Lambda(\diamond x.B) = \Lambda(B)$$

Where $\circ$ stands for the polarized and unpolarized binary connectives, $\star$ stands for unpolarized literals and polarized atoms, and $\diamond$ stands for quantifiers. $\bullet$

If $cnf(F) = \bigwedge_{i=1}^{n} C_i$ then the sequent $\vdash \cdot \Uparrow F$ starts an invertible phase that yields n premises of the form $\vdash \Lambda(C_i) \Uparrow \cdot$ where $i = 1..n$.

Example 5.1.1. *Let a, b, c, d, e be atomic formulas, the following invertible phase simulates the conjunctive normal form computation for the formula:*

$$(a \wedge b) \vee (c \wedge (d \vee e)) \rightsquigarrow (a \vee c) \wedge (a \vee d \vee e) \wedge (b \vee c) \wedge (b \vee d \vee e)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a, c \Uparrow \cdot}\,str}{\vdash a \Uparrow c}\,str \quad \dfrac{\dfrac{\dfrac{}{\vdash a, d, e \Uparrow \cdot}}{\vdash a \Uparrow d, e}\,2 \times str}{\vdash a \Uparrow d \vee^- e}\,\vee^-}{\vdash a \Uparrow c \wedge^- (d \vee^- e)}\,\wedge^-}{\vdash \cdot \Uparrow a, c \wedge^- (d \vee^- e)}\,str \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash b, c \Uparrow \cdot}}{\vdash b \Uparrow c}\,str \quad \dfrac{\dfrac{\dfrac{}{\vdash b, d, e \Uparrow \cdot}}{\vdash b \Uparrow d, e}\,2 \times str}{\vdash b \Uparrow d \vee^- e}\,\vee^-}{\vdash b \Uparrow c \wedge^- (d \vee^- e)}\,\wedge^-}{\vdash \cdot \Uparrow b, c \wedge^- (d \vee^- e)}\,str}{\dfrac{\vdash \cdot \Uparrow a \wedge^- b, c \wedge^- (d \vee^- e)}{\vdash \cdot \Uparrow (a \wedge^- b) \vee^- (c \wedge^- (d \vee^- e))}\,\vee^-}\,\wedge^-$$

$\odot$

$$[\![B \circ C]\!]^{\pm} = [\![B]\!]^{\pm} \circ^{-} [\![C]\!]^{\pm} \quad [\![a]\!]^{\pm} = a^{+} \quad [\![\neg a]\!]^{\pm} = a^{-}$$

Where $\circ \in \{\vee, \wedge\}$ and a is atomic.

Figure 5.1: negative polarity for the conjunctive normal form decision procedure.

Polarities

Because the invertible phase of *LKF* deals only with negatives, all connectives of the propositional formula are negative, as seen in figure 5.1. The polarity of the atoms is not important here, so the atoms are polarized positively (therefore their negated occurrences are polarized negatively).

Regions

Describing the shape of the proof is straightforward: a single invertible phase behaves as the first step of the procedure in definition 5.1.2. Because there are no non-atomic positive formulas, this invertible phase will only store atoms. The decide rule starting the following focused phase can be applied only on positive atoms.

The proof can be seen as having two regions, one for each of the steps of definition 5.1.2. The case of the first region is henceforth called `cnf` and the case of the second region is called `complement`:

```
type  cnf,complement    case.
```

Indexes

There is no indexing information because there is no proof evidence. The same index is given to all stored atoms:

```
type  lit    ξ.
```

Clerks & Experts

Before knowing what clerks and experts to define, one has to visualize the reconstructed proof. Following the polarization in figure 5.1, a formula can only have negative and positive atoms and negative disjunctions and conjunctions. Thus the only rules that can appear in the derivation are the negative conjunction and disjunction rules, and the store, decide, and initial rules. This prompts the definition, respectively, of $\wedge_c^-$,$\vee_c^-$ $\mathscr{S}_c$, $\mathscr{D}_e$ and $\mathscr{I}_e$. The decide expert $\mathscr{D}_e$ changes the active case from `cnf` to `complement` because it is at the border of the two regions of the proof.

$$\wedge_c^-(\texttt{cnf}, \texttt{cnf}, \texttt{cnf}) \quad \vee_c^-(\texttt{cnf}, \texttt{cnf}) \quad \mathscr{S}_c(\texttt{cnf}, F, \texttt{cnf}, \texttt{lit})$$
$$\mathscr{D}_e(\texttt{cnf}, \texttt{complement}, \texttt{lit}) \quad \mathscr{I}_e(\texttt{complement}, \texttt{lit})$$

Figure 5.2: Clerks and Experts for the conjunctive normal form decision procedure.

The five agents are all naïve and are shown in figure 5.2.

Proposition 5.1.1. *Through the above* p.r.i.c.e.*, if the propositional formula F is a tautology, there exists a proof in LKF^a of the sequent* $\texttt{cnf} \vdash \cdot \Uparrow [\![F]\!]^{\pm}$.

Proof. By the completeness of *LKF*, there exists a proof in *LKF* of the sequent $\vdash \cdot \Uparrow [\![F]\!]^{\pm}$. To prove that there also exists a proof in LKF^a using the above p.r.i.c.e., it is sufficient to show that whenever there is a correct cut-free proof in *LKF*, there is also a correct proof in LKF^a guided by the above p.r.i.c.e.. In particular, there are no false negatives (*i.e.,*unwanted failures) of the LKF^a system where the *LKF* system would succeed.

The only reason for failure would be for an agent to be given an active case on which it is not defined. The proof starts with the `cnf` case, on which the three clerks are defined. At the end of the invertible phase, the decide expert $\mathscr{D}_e$ changes the case to `complement`, on which the initial expert $\mathscr{I}_e$ is defined. Therefore, No agent can fail.

As mentioned in section 4.1.3, giving the same index `lit` for all stored formulas does not constraint the kernel (provided the decide and initial experts do communicate `lit` to the kernel).This allows the kernel to decide freely from the same set of atoms as one an unguided kernel would have decided from. □

5.2 Mating

Matings were introduced by Peter Andrews and used for theorem proving [Andrews, 1981] as the heart of the TPS system. A similar approach was independently developed by Bibel and called the Connection Method [Bibel, 1987]. While these methods work on Classical First-Order Logic, the semantics described here concerns only propositional logic. This example builds on the conjunctive normal form decision procedure description to check a mating proof.

Definition 5.2.1. *A* path *and the* subformula at path*, written* $|$*, are inductively defined as:*

- $\boxtimes$ *is the empty path*
- *if* ρ *is a path,* $\prec \rho$ *and* $\succ \rho$ *are paths*
- $F|_{\boxtimes} = F$
- $F|_{\rho} = A \circ B$*, iff* $F|_{\prec\rho} = A$ *and* $F|_{\succ\rho} = B$*. where* $\circ$ *is any binary connective.*

A mating *is a collection* $\mathscr{M}$ *of pairs of paths. It is correct proof evidence for the formula* F *if every clause in the conjunctive normal form of* F *has a complementary pair of literals* $\langle a, \neg a\rangle$ *such that these literals appear at positions* $F|_{\rho_i}$ *and* $F|_{\rho_j}$ *and such that* $\langle \rho_i, \rho_j\rangle \in \mathscr{M}$*.* •

Example 5.2.1. *The mating* $\{\langle \prec \boxtimes, \succ\prec\succ \boxtimes\rangle, \langle \prec \boxtimes, \succ\succ\succ \boxtimes\rangle\}$ *is correct proof evidence for the formula* $a \vee ((b \vee \neg a) \wedge (c \vee \neg a))$ *because the conjunctive normal form of this formula is* $(a \vee b \vee \neg a) \wedge (a \vee c \vee \neg a)$ *and each clause in this conjunctive normal form contains a pair of complementary predicates appearing at positions in the original formulas whose paths are paired in the mating.* ⊙

5.2.1 Mating's p.r.i.c.e.

The same methodology applies to the definition of this p.r.i.c.e.. The polarity assignment and the region delimitation are the same as for the *CNFdec* decision procedure's p.r.i.c.e.. The reconstructed proof of the formula in *LKF* starts in the same way: an invertible phase on negative connectives yielding premises containing only stored atoms. But unlike the previous p.r.i.c.e., the decide rule and the subsequent initial rule are restricted. A Mating $\mathscr{M}$ is checked as proof evidence for a formula F if for each premise $\vdash \Gamma \Uparrow \cdot$ of the invertible phase starting with the sequent $\vdash \cdot \Uparrow [\![F]\!]^{\pm}$, $\exists a^{+}, a^{-} \in \Gamma$ such that there exists an ordered pair $\langle \rho_i, \rho_j\rangle \in \mathscr{M}$ with $F|_{\rho_i} = a$ and $F|_{\rho_j} = \neg a$.

To be able to check this, one has to make sure that the atoms are stored with the paths to their position relative to the original formula. To do so, each time a formula B is decomposed (at the $\wedge^-$ and $\vee^-$ rules) into subformulas B_1 and B_2, the respective clerks maintain the paths of B_1 and B_2 relative to the path of B itself. Thus the region inhabited by these clerks will have a case `path` containing the correspondence between the formulas in the sequent and their paths. The store clerk and the decide and initial experts will use these paths as indexes by means of this type declaration:

$$\begin{array}{lll} \texttt{type} & \boxtimes & \xi. \\ \texttt{type} & \prec & \xi \rightarrow \xi. \\ \texttt{type} & \succ & \xi \rightarrow \xi. \end{array}$$

and the `path` case has the following type declaration:

$$\texttt{type} \quad \texttt{path} \quad \texttt{list}\ \xi \to \texttt{case}.$$

The end-sequent of the proof is:

$$\texttt{path}[\boxtimes] \vdash \cdot \Uparrow F$$

Example 5.2.2. *For atoms a, b, c; and a_ρ the indexed formula $\langle \rho, a \rangle$ (in accordance with remark 3.4.1).*

$$\dfrac{\dfrac{\dfrac{}{\texttt{path}[] \vdash a_{\prec\boxtimes}, b_{\prec\succ\boxtimes} \Uparrow \cdot}}{\texttt{path}[\prec\succ \boxtimes] \vdash a_{\prec\boxtimes} \Uparrow b} \quad \dfrac{\dfrac{}{\texttt{path}[] \vdash a_{\prec\boxtimes}, c_{\succ\succ\boxtimes} \Uparrow \cdot}}{\texttt{path}[\succ\succ \boxtimes] \vdash a_{\prec\boxtimes} \Uparrow c}}{\dfrac{\texttt{path}[\succ \boxtimes] \vdash a_{\prec\boxtimes} \Uparrow b \wedge^- c}{\dfrac{\texttt{path}[\prec \boxtimes, \succ \boxtimes] \vdash \cdot \Uparrow a, b \wedge^- c}{\texttt{path}[\boxtimes] \vdash \cdot \Uparrow a \vee^- b \wedge^- c}}}$$

$\odot$

The clerks are as follows:

$$\wedge_c^-(\texttt{path}\ [\rho|R], \texttt{path}\ [\prec \rho|R], \texttt{path}\ [\succ \rho|R])$$

$$\vee_c^-(\texttt{path}\ [\rho|R], \texttt{path}\ [\prec \rho, \succ \rho|R])$$

Because the right-hand side of the uparrow is a list order, the store rule is always applied to the left-most formula. The store clerk $\mathscr{S}_c$ can simply remove the head of the list of paths to serve as index.

$$\mathscr{S}_c(\texttt{path}\ [\rho|R], F, \texttt{path}\ R, \rho)$$

At the end of the invertible phase, the `path` case should be empty. The decide expert then picks (that is to say, succeeds with) a pair from the mating, gives the first element as index for the decide rule and places the second element inside the `initWith` case.

$$\mathscr{D}_e(\texttt{path}\ [], \texttt{initWith}\ [\rho_j], \rho_i) \quad \texttt{:-} \quad \langle \rho_i, \rho_j \rangle \in \mathscr{M}$$

The initial expert given in section 4.3.1 finishes the proof.

The agents are summarized in figure 5.3

$$\wedge_c^- \ (\texttt{path}\ [\rho|R], \texttt{path}\ [\prec \rho|R], \texttt{path}\ [\succ \rho|R])$$
$$\vee_c^- \ (\texttt{path}\ [\rho|R], \texttt{path}\ [\prec \rho, \succ \rho|R])$$
$$\mathscr{S}_c(\texttt{path}\ [\rho|R], F, \texttt{path}\ R, \rho)$$
$$\mathscr{D}_e(\texttt{path []}, \texttt{initWith}\ [\rho_j], \rho_i) \quad \texttt{:-} \quad \langle \rho_i, \rho_j \rangle \in \mathscr{M}$$

Figure 5.3: Clerks and Experts for the Mating proof evidence description

Proposition 5.2.1. *Given the clerks in figure 5.3, the following fact is invariant throughout the invertible phase: the paths contained in the* `path` *case are in one-to-one correspondence with the formulas in the workbench of the sequent and designate their actual paths in the original formula.*

Proof. The invariant is satisfied at the end-sequent: $\texttt{path}[\boxtimes] \vdash \cdot \Uparrow F$.

By case analysis on the rules.

$$\frac{\texttt{path}\ [\prec \rho, \succ \rho|R] \vdash \Theta \Uparrow B, C, \Gamma \quad \vee_c^- \ (\texttt{path}\ [\rho|R], \texttt{path}\ [\prec \rho, \succ \rho|R])}{\texttt{path}\ [\rho|R] \vdash \Theta \Uparrow B \vee^- C, \Gamma}$$

By induction, R is the list of paths of all formulas in the list Γ, $\prec \rho$ is the path of B, $\succ \rho$ is the path of C. Following definition 5.2.1, ρ is the path of the formula $B \vee^- C$, which preserves the invariant. The cases for the other rules follow in the same way. □

Corollary 5.2.1. *All atoms are stored with their correct path.*

Remark 5.2.1. The elements of the mating are defined as ordered pairs, the first element is a path to a non negated atom and the second is a path to a negated atom. If the pairs of the mating were unordered then the decide expert would require an additional definition:

$$\mathscr{D}_e(\texttt{path []}, \texttt{initWith}\ [\rho_j], \rho_i) \quad \texttt{:-} \quad \langle \rho_j, \rho_i \rangle \in \mathscr{M}$$

Ordering the pairs therefore results in a precision gain. While the gain is negligible for this particular case study, simple requirements such as ordering of proof evidence can result in exponential gains in precision and, consequently, in checking time.

5.3 Resolution refutations

The notion of *clause* in definition 5.1.1 is generalized to the first-order level:

Definition 5.3.1. *A* clause *is the universal closure of a disjunction of literals. An empty clause is false. A* ground *clause contains only ground terms.* i.e.,*terms with no variables.* •

Resolution is a valid inference rule originally introduced in [Davis and Putnam, 1960] where it is called *Rule for Eliminating Atomic Formulas.* This rule states that from two *ground clauses* containing complementary literals, called *resolved clauses*, one can obtain a third clause, called the *resolvent*, which is the disjunction of the resolved clauses without the complementary literals.

More formally, let C_1 and C_2 be two ground clauses and let a be an atomic formula. Applying the resolution rule to $C_1 \vee a$ and $C_2 \vee \neg a$ yields the resolvent clause C where C is the disjunction $C_1 \vee C_2$ without repeated literals.

The resolution inference rule is written:

$$\frac{C_1 \vee a \quad C_2 \vee \neg a}{C} \tag{5.1}$$

This resolution rule operates only on ground clauses, a deficiency that Robinson solved by generalizing the resolution rule and using his unification algorithm to obtain what is called the *Resolution Principle* [Robinson, 1965b]. Today, resolution is at the heart of many provers, among which Vampire and Carine[1], and the unification algorithm made the current notion of logic programming possible.

Henceforth, resolution will always refer to Robinson's resolution.

Refutations: A finite set of clauses $\mathcal{C}$ is refuted if the conjunction of its clauses implies falsehood, *i.e.,*if the following formula is a tautology.

$$(\bigwedge_{C \in \mathcal{C}} C) \supset f \tag{5.2}$$

Repeated application of resolution is used for deriving a *refutation* of an unsatisfiable set of clauses.

Definition 5.3.2. *A clause C is* usable *in a resolution step for the refutation of a set of clauses $\mathcal{C}$ if $C \in \mathcal{C}$ or C is the result of a previous resolution step. A resolution refutation is a sequence of resolution rule applications on usable formulas. If this process reaches an empty clause then the set is refuted.* •

[1]Respectively http://www.vprover.org/ and http://www.atpcarine.com/

Example 5.3.1. *The following is a resolution refutations of the set*

$$\mathcal{C} = \{a \vee b, \neg a, \neg b\}$$

The original clauses are separated from the resolvent (or intermediary) clauses by a horizontal bar.

1. $a \vee b$
2. $\neg a$
3. $\neg b$

———

4. a *from* 1 *and* 3
5. f *from* 4 *and* 2

In the beginning, only clauses with indices from 1 *to* 3 *are usable. Clause* 4 *is derived from applying a resolution rule to* 1 *and* 3 *and becomes usable by all subsequent resolution rules. Here it is used with 2 to derive the empty clause, or f.* ⊙

Recall that the first step in defining the semantics of a language $\mathscr{L}$ using the Foundational Proof Certification framework is to find an interpretation of the constructs of $\mathscr{L}$ in a focused sequent calculus (here *LKF*). Using this interpretation, one then defines the semantics of $\mathscr{L}$ through a relational specification. This specification, or `p.r.i.c.e.`, forms a proof certificate when added to some proof evidence written in $\mathscr{L}$. The proof certificate guides the kernel to a proof (in focused sequent calculus) of the proposed theorem, whose validity is claimed by the proof evidence. The existence of this proof, by the soundness of the focused sequent calculus, entails validity of the proposed theorem T.

Modular separation

A resolution refutation can be seen as two processes, a *sequence checker* and a *step checker*. The former asks the latter about the validity of each resolution step and, depending on its answer, either fails or adds the resolvent clause to the set usable clauses and continues until reaching the empty clause. The *step checker* verifies that the resolved clauses are usable and that the resolvent clause logically follows from them.

Notice that the *sequence checker* needs not have knowledge of what particular checking the *step checker* does. It simply supplies it with clauses to resolve and

expects a resolvent clause. This separation allows to reuse the *sequence checker* with more than just binary resolution. Indeed, resolution was refined and extended in multiple ways, two of which (hyperresolution and paramodulation) are presented in this thesis.

Hence the following sections are articulated around the separation of processes *sequence checker* and *step checker*. First, the refutation sequence is given a precise semantics definition (with no prior assumption on the organization of the resolved clauses), then that definition is used with binary resolution (section 5.3.2) and hyper-resolution (section 5.3.3) and, in a later chapter, with paramodulation (section 7.3).

5.3.1 Semantics of refutation sequences

This section follows a plan in the definition of a `p.r.i.c.e.` for a refutation sequence. First, the theorem (refutation of a finite set of clauses) is embedded in an *LKF* end-sequent. Then, the structure of *LKF* proofs of that sequent is defined following the standard proof evidence format for resolution (definition 5.3.3). To do so, the different choices of polarities are discussed in detail. Finally, case constructors and agent definitions are given to guide the kernel into the discussed structure of *LKF* proofs. This section focuses on the *sequence checker* process, without discussing the *step checker* process.

Refuted clause sets as *LKF* sequents

The unsatisfiability of a finite set of clauses $\mathcal{C}$ is equivalent to the validity of the tautologous implication 5.2 which, following the material conditional interpretation of classical logic, is equivalent to:

$$(\bigvee_{C \in \mathcal{C}} \neg C) \vee f$$

which, in turn can be simplified to

$$(\bigvee_{C \in \mathcal{C}} \neg C)$$

Proposition 5.3.1. *The sequent* $\vdash \cdot \Uparrow [\![(\bigvee_{C \in \mathcal{C}} \neg C)]\!]^{\pm}$ *is provable if the set of clauses* $\mathcal{C}$ *is refutable.*

Proof. Because the implication 5.2 is tautologous, a proof exists in *LK* by the completeness and soundness of *LK* with respect to classical logic. The proposition follows by the completeness and soundness of *LKF* with respect to *LK*. □

Interpreting refutations in *LKF*

Definition 5.3.3. *An* indexed clause *is a pair $\langle i, C_i \rangle$ of an index and a clause. A* refutation sequence *consists of tuples $\langle \mathcal{S}, k, C_k \rangle$ where $\mathcal{S}$ represents some organization of indexes of resolved and usable clauses, simply called a* step*. The clause C_k is the result of applying some variant of resolution to that step and receives index k. The last element of the final tuple is* f *(empty clause).*

A coclause *$\neg C_i$ is the negation of the clause C_i in negation normal form.* •

The interpretation of the refutation of a set of clauses $\mathcal{C}$ starts with the sequent seen in proposition 5.3.1:

$$\vdash \cdot \Uparrow [\![(\bigvee_{C \in \mathcal{C}} \neg C)]\!]^{\pm}$$

Thus interpreting a refutation revolves around coclauses as well as clauses. Because the negation flips the polarity (a negative disjunction in the clauses entails positive conjunctions in the coclauses and vice versa), the discussion on polarity assignment considers both. To make a pertinent choice of polarity for the above disjunctions, one can ask the questions in section 4.1.1 and assess each choice of polarity by how it answers the needs of the verification process. One such assessment follows for the outer disjunction of the coclauses; the polarity choices for the coclauses themselves is seen later on. In what follows, $C_1 \cdots C_n \in \mathcal{C}$.

A positive disjunction will cause the formula to be stored immediately. Every time a coclause $\neg C_k$ is used in needed, the whole disjunction is contracted and a focused phase is engaged on it.

$$\vdash (\overset{+}{\bigvee_{C \in \mathcal{C}}} [\![\neg C]\!]^{\pm}) \Downarrow (\overset{+}{\bigvee_{C \in \mathcal{C}}} [\![\neg C]\!]^{\pm})$$

Assuming the disjunction is right-associative, to reach the coclause $\neg C_k$ one needs to apply the $\vee^+$ rule $k-1$ times and pick the right disjunct, then apply an additional such application to pick the left disjunct. For instance, accessing the 2nd clause is done through the following derivation:

$$\dfrac{\dfrac{\begin{array}{c}\vdots \\ \vdash \Gamma \Downarrow [\![\neg C]\!]^{\pm}_2\end{array}}{\vdash \Gamma \Downarrow [\![\neg C_2]\!]^{\pm} \vee^+ ([\![\neg C_3]\!]^{\pm} \vee^+ \cdots \vee^+ [\![\neg C_n]\!]^{\pm}) \cdots)}\ \vee^+_l}{\vdash \Gamma \Downarrow [\![\neg C_1]\!]^{\pm} \vee^+ ([\![\neg C_2]\!]^{\pm} \vee^+ ([\![\neg C_3]\!]^{\pm} \vee^+ \cdots \vee^+ [\![\neg C_n]\!]^{\pm}) \cdots)}\ \vee^+_r$$

This is arguably far from the original spirit of the indexes, traditionally used for direct access. This direct access can be achieved by storing each individual coclause with its index which leads to a negative polarity for the above disjunctions and a positive polarity for the coclauses.

A negative disjunction will be decomposed immediately and replaced with a comma:

$$\frac{\vdash \Uparrow [\![\neg C_1]\!]^{\pm}, ([\![\neg C_1]\!]^{\pm} \vee^- \cdots \vee^- [\![\neg C_n]\!]^{\pm})}{\vdash \Uparrow [\![\neg C_1]\!]^{\pm} \vee^- ([\![\neg C_1]\!]^{\pm} \vee^- \cdots \vee^- [\![\neg C_n]\!]^{\pm})}$$

To use the index of each coclause for its storage, they need to be stored immediately which requires them to be of a positive polarity. This is achieved either:

- by using the positive version of the conjunction in the coclauses or
- by using the negative version of the conjunction and surrounding them with a positive delay $\partial^+(\cdot)$.

Remark 5.3.1. The polarity assignment for the conjunctions in the coclauses need not be set immediately. Indeed, the *sequence checker* process never decomposes the coclauses; only the *step checker* process, seen later, needs a precise polarity. For now let B^+ stand for some polarization of arbitrary formula B that makes it of a positive polarity.

So far, the original coclauses are stored with their indexes, yielding the sequent:

$$\vdash \{\langle i, (\neg C_i)^+ \rangle | \langle i, C_i \rangle \in \mathcal{C}\} \Uparrow \cdot$$

Remark 5.3.2. Henceforth, the shortened notation F_I in the storage stands for the stored and indexed formula $\langle I, F \rangle$.

Remark 5.3.3. The *raison d'être* of sequent calculus is proving conditional tautologies. Once the link has been made between the sequent $\vdash \cdot \Uparrow (\bigvee^-_{C \in \mathcal{C}} \neg C)$ (the provability of which determines the refutation of the set of clauses $\mathscr{C}$) and the sequent $\vdash \{\neg C | C \in \mathcal{C}\} \Uparrow \cdot$ (equiprovable to the former sequent from which it follows using only invertible rules), the checker can directly start verifying the latter sequent. Furthermore, in practice, resolution refutations are seldom given as a conjunction of clauses but as a set of clauses. Thus constructing the disjunctions out of the set of coclauses and then breaking the disjunctions to recover the set of coclauses can be avoided altogether.

The above sequent marks the beginning of the checking of the refutation sequence. To do so, an interpretation in *LKF* for a call to the process *step checker* remains to be established. This interpretation must have, as conclusion, the above sequent (containing all usable coclauses) and as premise a similar sequent with one extra usable coclause: the resolvent coclause. In other words, for each tuple $\langle \mathcal{S}, k, C_k \rangle$, this interpretation should go from a sequent:

$$\vdash \Delta \Uparrow \cdot$$

where Δ represents the current set of usable clauses of which $\mathcal{S}$ is a subset, to the sequent:

$$\vdash \langle k, (\neg C_k)^+ \rangle, \Delta \Uparrow \cdot$$

after having received confirmation of the *step checker* for the validity of the $\langle \mathcal{S}, k, C_k \rangle$.

This process can be simulated using the following partial derivation:

$$\dfrac{\dfrac{\textit{step checker}}{\vdash \Delta \Uparrow C_k^-} \quad \dfrac{\dfrac{\textit{continue with sequence}}{\vdash \langle k, (\neg C_k)^+ \rangle, \Delta \Uparrow \cdot}}{\vdash \Delta \Uparrow (\neg C_k)^+}}{\vdash \Delta \Uparrow \cdot}\ \textit{cut} \tag{5.3}$$

where one notes that the negation of the positive coclause $(\neg C_k)^+$ introduced by the cut is the negative clause C_k^-.

Agents and cases for refutations

As mentioned in 4.1.2, a reconstructed proof in *LKF* can be viewed as split into regions that correspond to different processes of the checking. In the case of a refutation, these are the processes *sequence checker* and *step checker*. This section details the region relative to the former, with its case constructors and agent definitions.

The case constructors depend on the information needed by the agents inhabiting this region to complete their task. The reconstructed proof has the shape seen in figure 5.4 where each π_k is the proof verifying the step $\langle \mathcal{S}_k, k, C_k \rangle$.

The two regions are visible as:

- the back-bone of cuts followed by a decision and a t^+ rule, for the *sequence checker* process and
- the derivations π_i for the successive calls to the *step checker* process. These derivation are, at most, of a decide depth equal to the number of resolved clauses.

$$
\dfrac{\dfrac{\pi_1}{\vdash \Delta_0 \Uparrow C_1} \qquad \dfrac{\dfrac{\dfrac{\pi_2}{\vdash \Delta_1 \Uparrow C_2} \qquad \dfrac{\dfrac{\dfrac{\pi_r}{\vdash \Delta_{r-1} \Uparrow C_r} \qquad \dfrac{\dfrac{\dfrac{\pi_{r+1}}{\vdash \Delta_r \Uparrow f^-} \qquad \dfrac{\dfrac{\dfrac{}{\vdash \Delta_r, t^+ \Downarrow t^+}\ t^+}{\vdash \Delta_r, t^+ \Uparrow \cdot}\ \mathit{decide}}{\vdash \Delta_r \Uparrow t^+}}{\vdash \Delta_r \Uparrow \cdot}}{\vdash \Delta_{r-1} \Uparrow \neg C_r}}{\vdash \Delta_{r-1} \Uparrow \cdot}\ \mathit{cut}}{\vdots \atop \vdash \Delta_1 \Uparrow \neg C_2}}{\vdash \Delta_1 \Uparrow \cdot}\ \mathit{cut}}{\vdash \Delta_0 \Uparrow \neg C_1}\ \mathit{store}}{\vdash \Delta_0 \Uparrow \cdot}\ \mathit{cut}
$$

Figure 5.4: The structure of a reconstructed resolution refutation proof. Here Δ_0 is is the set of original coclauses, C_i is the ith resolvent clause and $\Delta_i = \Delta_{i-1} \cup \{\neg C_i\}$

The derivation of example 5.3.1 is shown in figure 5.5 and one can follow the behavior of the agents, step-by-step, as they are defined below. They are introduced in the order in which they appear in the derivation (bottom-up) and summarized in figure 5.6.

The main inhabitants of the first region are the cut expert $\mathscr{C}_e$ and the store clerk $\mathscr{S}_c$. The former needs access to the sequence $\mathcal{Q}$ of resolution steps $\langle \mathcal{S}_k, k, C_k \rangle$; the latter needs the index k to store the resolvent coclause C_k. The case constructor is given the following type declaration:

$$\texttt{type} \quad \texttt{rlist} \qquad \texttt{list}\ (\texttt{stuple} * \mathbb{N} * \texttt{fm}) \to \texttt{case}.$$

where `stuple` is the type of the step (some organization of the resolved clauses whose details this region need not know), $\mathbb{N}$ is the type of integer and `fm` is the type of the polarized formulas.

Remark 5.3.4. In the example derivations, a light notation is sometimes used to save space. For example, `step` L instead of `step br`L.

The augmented end-sequent has the following form:

$$\texttt{rlist}\ \mathcal{Q} \vdash \Delta \Uparrow \cdot$$

where Δ is the set of stored original indexed coclauses. The cut expert is defined on this case constructor in this way:

$$\mathscr{C}_e(\texttt{rlist}\ [\langle \mathcal{S}_k, K, C_K \rangle | \mathcal{Q}], \texttt{step}\ \mathcal{S}, \texttt{rlisti}\ \ K\ \mathcal{Q}, C_K).$$

$$
\dfrac{\dfrac{\pi}{\Xi_2 \vdash \Gamma \Uparrow a^+} \qquad \dfrac{\dfrac{\dfrac{\pi'}{\Xi_8 \vdash \Gamma \Uparrow f^-} \qquad \dfrac{\dfrac{\dfrac{}{\Xi_{11} \vdash \Gamma, \langle \mathtt{idx}\ \ 4, a^- \rangle, \langle \mathtt{idx}\ \ 5, t^+ \rangle \Downarrow t^+}\ t_e^+(\Xi_{11})}{\Xi_{10} \vdash \Gamma, \langle \mathtt{idx}\ \ 4, a^- \rangle, \langle \mathtt{idx}\ \ 5, t^+ \rangle \Uparrow \cdot}\ \mathscr{D}_e(\Xi_{10}, \Xi_{11}, \mathtt{idx}\ \ 5)}{\Xi_9 \vdash \Gamma, \langle \mathtt{idx}\ \ 4, a^- \rangle \Uparrow t^+}\ \mathscr{S}_c(\Xi_9, _, \Xi_{10}, \mathtt{idx}\ \ 5)}{\Xi_4 \vdash \Gamma, \langle \mathtt{idx}\ \ 4, a^- \rangle \Uparrow \cdot}\ \mathscr{C}_e(\Xi_4, \Xi_8, \Xi_9, f^-)}{\Xi_3 \vdash \Gamma \Uparrow a^-}\ \mathscr{S}_c(\Xi_3, _, \Xi_4, \mathtt{idx}\ \ 4)}{\Xi_1 \vdash \Gamma \Uparrow \cdot}\ \mathscr{C}_e(\Xi_1, \Xi_2, \Xi_3, a^+)
$$

$$
\begin{array}{ll}
\Gamma = \{\langle \mathtt{idx}\ \ 1, a^- \wedge^+ b^- \rangle, \langle \mathtt{idx}\ \ 2, a^+ \rangle, \langle \mathtt{idx}\ \ 3, b^+ \rangle\} & \Xi_2 = \mathtt{step}\ \ [1,3] \\
\Xi_1 = \mathtt{rlist}\ \ [\langle [1,3], 4, a^+ \rangle, \langle [2,4], 5, f^- \rangle] & \Xi_4 = \mathtt{rlist}\ \ [\langle [2,4], 5, f^- \rangle] \\
\Xi_3 = \mathtt{rlisti}\ \ 4\, [\langle [2,4], 5, f^- \rangle] & \Xi_5 = \mathtt{endWith}\ \ 4 \\
\Xi_8 = \mathtt{step}\ \ [2,4] & \Xi_9 = \mathtt{rlisti}\ \ 5\, [] \\
\Xi_{10} = \mathtt{decOn}\ \ (\mathtt{idx}\ \ 5) & \Xi_{11} = \mathtt{done}
\end{array}
$$

Figure 5.5: The `rlist` region of the derivation for example 5.3.1

The first argument (the input case) is the `rlist` case containing the sequence of steps from which $\mathscr{C}_e$ removes the head and does three things:

- it communicates the cut formula, here C_K, to the kernel
- it creates, for the left premise, a `step` case with the resolved clauses $\mathcal{S}_k$, and
- it gives to the right branch the case `rlisti`, containing the rest of the sequence of tuples as well as the index K, used immediately after by the store clerk $\mathscr{S}_c$ to index the coclause $\neg C_K$.

For now, attention is focused on the right premise of the cut rule, as it is part of the same region (the left premise is part of the region related to the *step checker* process whose p.r.i.c.e. is seen later). Once a cut rule is applied on the resolvent clause at the head of $\mathcal{Q}$, the head is removed from $\mathcal{Q}$. The tail of $\mathcal{Q}$ is preserved in an auxiliary case, `rlisti`, along with the index of the most recent resolvent clause. The store clerk $\mathscr{S}_c$ uses this index to store the coclause. The auxiliary case constructor has a similar type to that of `rlist`:

$$\mathtt{type}\ \ \mathtt{rlisti} \qquad \mathbb{N} \to \mathtt{list}\ \ (\mathtt{stuple} * \mathbb{N} * \mathtt{fm}) \to \mathtt{case}.$$

while the `step` case constructor has type:

$$\mathtt{type}\ \ \mathtt{step} \qquad \mathtt{stuple} \to \mathtt{case}.$$

Remark 5.3.5. One can also use a single case `rlist` and define the agents accordingly. However, they are left separate to stress the difference between the notions of *region* and *case*, a difference comparable to the one between a process and the variables used in that process.

Separating the two cases has the additional advantage of readability, which one should always keep in mind when writing a p.r.i.c.e.. Just as it is possible but not always practical to create a single global data structure in a program where all information is stored, it is possible to define a single case and place all information in it.

The store clerk is defined on `rlisti`:

$$\mathscr{S}_c(\texttt{rlisti}\ K\, R, _, \texttt{rlist}\ R, \texttt{idx}\ K).$$

It reverts the active case to `rlist` and uses the integer given in the tuple to generate an index through the constructor `idx`:

$$\texttt{type}\quad \texttt{idx}\qquad \mathbb{N} \to \xi.$$

The proof continues until the sequence of steps is emptied. The last step is given by a tuple $\langle \mathcal{S}_l, L, f^- \rangle$; thus the last cut rule is

$$\frac{\vdash \Delta \Uparrow f^- \quad \vdash \Delta \Uparrow t^+}{\vdash \Delta \Uparrow \cdot}$$

The right premise is augmented with a case `rlisti` K []. The proof of this last sequent is obtained by storing the formula t^+ then immediately deciding on it and finishing the proof with a t^+ rule. Thus the store clerk has another definition for this final situation:

$$\mathscr{S}_c(\texttt{rlisti}\ K\, [], _, \texttt{oneOf}\ [\texttt{idx}\ K]\, \texttt{done}, \texttt{idx}\ K)$$

The agents defined in sections 4.3.1 and 4.3.2, in particular the decide and true experts, finish the proof.

This concludes the p.r.i.c.e. relative to the process *sequence checker*.

5.3.2 Semantics of a binary resolution step

Following the same process as the one described in the previous section, the resolution rule is first interpreted using an *LKF* sequent, then the structure of *LKF* proofs of that sequent is visualized, finally a p.r.i.c.e. is defined to guide the kernel towards finding such proofs.

$$\mathscr{C}_e(\texttt{rlist}\ [\langle \mathcal{S}_k, K, C_k\rangle | R], \texttt{step}\ \mathcal{S}_k, \texttt{rlisti}\ \ K\ R, C_k)$$
$$\mathscr{S}_c(\texttt{rlisti}\ \ K\ R, _, \texttt{rlist}\ \ R, \texttt{idx}\ \ K)$$
$$\mathscr{S}_c(\texttt{rlisti}\ \ K\ [], _, \texttt{oneOf}\ \ [\texttt{idx}\ \ K]\ \texttt{done}, \texttt{idx}\ \ K)$$

Figure 5.6: Clerks and Experts working on the `rlist` case

Binary resolution steps as *LKF* sequents

A resolution inference rule

$$\frac{C_i \quad C_j}{C_k}$$

with a conflict atom $a \in \Lambda(C_i)$ and $\neg a \in \Lambda(C_j)$ (for $\Lambda(\cdot)$ shown in definition 5.1.3) can also be seen as the tautologous implication

$$C_i \wedge C_j \supset C_k$$

which, by the material implication interpretation, is classically equivalent to the formula :

$$\neg C_i \vee \neg C_j \vee C_k$$

Proposition 5.3.2. *If the resolution inference rule is applicable on clauses C_i and C_j to produce the resolvent clause C_k, then the following sequent is provable:*

$$\vdash \cdot \Uparrow [\![\neg C_i \vee \neg C_j \vee C_k]\!]^{\pm}$$

Proof. The arguments for the proof of proposition 5.3.1 apply. □

One has to link the sequent that is the left premise of the cut rule in the partial derivation 5.3:

$$\vdash \Delta \Uparrow C_k^- \tag{5.4}$$

to the sequent from proposition 5.3.2

$$\vdash \cdot \Uparrow [\![\neg C_i \vee \neg C_j \vee C_k]\!]^{\pm}$$

By giving a negative polarity to the two disjunctions in $\neg C_i \vee \neg C_j \vee C_k$ and to C_k, and a positive polarity to $\neg C_i$ and $\neg C_j$, the above sequent becomes:

$$\vdash \cdot \Uparrow (\neg C_i)^+ \vee^- (\neg C_j)^+ \vee^- C_k^-$$

which, by application of only invertible rules, produce the following partial derivation:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{\vdash (\neg C_j)^+, (\neg C_i)^+ \Uparrow C_k^-}}{\vdash (\neg C_i)^+ \Uparrow (\neg C_j)^+, C_k^-}\ \mathscr{S}}{\vdash (\neg C_i)^+ \Uparrow (\neg C_j)^+ \vee^- C_k^-}\ \vee^-}{\vdash \cdot \Uparrow (\neg C_i)^+, (\neg C_j)^+ \vee^- C_k^-}\ \mathscr{S}}{\vdash \cdot \Uparrow (\neg C_i)^+ \vee^- (\neg C_j)^+ \vee^- C_k^-}\ \vee^-$$

Thus the sequent:

$$\vdash (\neg C_i)^+, (\neg C_j)^+ \Uparrow C_k^- \tag{5.5}$$

is provable if resolution step $\langle [i,j], k, C_k \rangle$ is valid. Furthermore, If $\langle [i,j], k, C_k \rangle$ is a valid resolution step, then $(\neg C_i)^+, (\neg C_j)^+ \in \Delta$ (where Δ is the set of usable coclauses). The only rules that interact with the storage are the decide rule, when a formula is contracted out of storage, and the initial rule, when a negative atom in the storage is mated with the positive atom under focus.

Proposition 5.3.3. *If proofs of the sequent 5.4 never interact with the set of coclauses $\Delta \setminus \{(\neg C_i)^+, (\neg C_j)^+\}$, then the sequent 5.4 is equiprovable to the sequent 5.5.*

Proof. By induction on the given derivations. □

Remark 5.3.6. To prevent the proof of sequent 5.4 to use formulas in $\Delta \setminus \{(\neg C_i)^+, (\neg C_j)^+\}$, it is sufficient to define the decide $\mathscr{D}_e$ and initial $\mathscr{I}_e$ experts to never give indexes of formulas in $\Delta \setminus \{(\neg C_i)^+, (\neg C_j)^+\}$.

Interpreting binary resolution steps in *LKF*

The choice of polarity postponed in remark 5.3.1 will now be discussed. So far, the coclauses were made positive so that the original coclauses are stored with their indexes. Because negation flips polarity, the choice of a positive polarity for the coclauses entails a negative polarity for the clauses. In the cut rule 5.3, simulating a resolution step application, the (negative) resolvent clause C_k appears on the left premise while its (positive) coclause $\neg C_k$ appears on the right premise:

$$\cfrac{\vdash \Delta \Uparrow C_k^- \quad \vdash \Delta \Uparrow (\neg C_k)^+}{\vdash \Delta \Uparrow \cdot}\ cut$$

As mentioned when discussing polarity assignment of the coclauses, a formula is made positive (resp. negative) either by using a positive (resp. negative) delay

or by giving a positive (resp. negative) polarity to its top connective. Through the negation, if the coclauses are made positive by a positive delay then the clauses would symmetrically be made negative by a negative delay. If the coclauses are made positive by using the positive version of the conjunction then the clauses would be made negative by the negative version of the disjunction.

Because the left premise starts in the invertible phase, a negative delay will be immediately removed:

$$\frac{\vdash \Delta \Uparrow C_k}{\vdash \Delta \Uparrow \partial^-(C_k)}$$

Therefore, the rest of the phase depends on the polarity of the disjunction.

A positive disjunction will cause the immediate next rule to be the storage of the clause C_k. Furthermore, the proof will decide (contract) on the clause C_k each time one of its literals is needed, and the ensuing focused phase will potentially contain as many $\vee^+$ rules as there are disjunctions in the clause. Without direction information, of which there is none, the $\vee^+$ rules greatly increase the search space. This is the first objection to the positive polarity assignment for disjunctions in clauses.

Moreover, because there is only one conflict literal per binary resolution step, the remaining literals in the coclauses must each be mated with a literal in resolvent clause. This means that a positive polarity for the disjunctions in C_k would require the proof of the above sequent to contract $\|\Lambda(C_i)\| + \|\Lambda(C_j)\|$ times on the clause C_k, where C_i and C_j are the resolved clauses. The final argument against the positive polarity is that all of the literals in C_k are used in the proof, so there is no advantage in having a disjunction rule that discards one of the disjuncts.

A negative polarity to the disjunctions in the clauses results an invertible phase where all disjunctions in C_k are replaced by commas and the atoms $\Lambda(C)$ stored, resulting in the sequent

$$\vdash \{A | A \in \Lambda(C_k)\}, \Delta \Uparrow \cdot$$

The polarity of atoms is of little consequence for a binary resolution step, they are hence given a global arbitrary polarity. The polarity assignment is summarized in figure 5.7

Now that the polarity is completely specified, formulas will not be written with a superscript informing of they polarity.

$$[\![B \vee C]\!]^{\pm} = [\![B]\!]^{\pm} \vee^{-} [\![C]\!]^{\pm} \quad [\![B \wedge C]\!]^{\pm} = [\![B]\!]^{\pm} \wedge^{+} [\![C]\!]^{\pm}$$
$$[\![f]\!]^{\pm} = f^{-} \quad [\![t]\!]^{\pm} = t^{+} \quad [\![atm]\!]^{\pm} = atm^{+} \quad [\![\neg atm]\!]^{\pm} = atm^{-}$$

Figure 5.7: Polarity assignment for the resolution refutation p.r.i.c.e..

Agents and cases for binary resolution steps

A binary resolution `stuple` constructor is defined on an ordered pair as:

$$\texttt{type} \quad \texttt{br} \quad \texttt{list}\ \xi \to \texttt{stuple}.$$

where the argument in the list contains two indexes, the first is the index of the clause containing the non-negated conflict literal while the second is the index of the clause containing the complementary negated literal.

Recall that the case of this region is `step`:

$$\texttt{type} \quad \texttt{step} \qquad \texttt{stuple} \to \texttt{case}.$$

In particular, the augmented end-sequent of the left premise of the cut is:

$$\texttt{step}(\texttt{br}[i, j]) \vdash \Delta \Uparrow C_k$$

Concerning the indexing, the only stored formulas in these subderivations are atomic: they are given an indiscriminate index `lit`. It is important to give a different index than for the coclauses, following remark 5.3.6.

The gents are defined below in order of appearance and can be followed on the rest of the derivation for example 5.3.1 shown in figure 5.9.

The derivation of this sequent will be a sequence of $\vee^-$ rules augmented with the naïve definition of the $\vee_c^-$:

$$\vee_c^-(\texttt{step}R, \texttt{step}R)$$

The store clerk gives the index `lit` to each store rule:

$$\mathscr{S}_c(\texttt{step}R, _, \texttt{step}R, \texttt{lit})$$

regardless of what the atom is (hence the wildcard). At the end of the invertible phase, the sequent is

$$\vdash \{\langle \texttt{lit}, A\rangle | A \in \Lambda(C_k)\}, \Gamma \Uparrow \cdot$$

To define the rest of the agents, one has to think of the order in which the resolvent clauses are given. For a resolution step on clauses $a \vee B$ and $\neg a \vee B'$, the clauses are polarized to become $a^+ \vee^- [\![B]\!]^\pm$ and $a^- \vee^- [\![B']\!]^\pm$, then negated to yield the coclauses $a^- \wedge^+ \neg[\![B]\!]^\pm$ and $a^+ \wedge^+ \neg[\![B']\!]^\pm$. Let Δ be the multiset $\{\langle \texttt{lit}, A\rangle | A \in \Lambda(C_k)\} \cup \Gamma$, a decide on the second coclause produces the following partial derivation:

$$\dfrac{\vdash \Delta \Downarrow a^+ \quad \vdash \Delta \Downarrow \neg[\![B']\!]^\pm}{\vdash \Delta \Downarrow a^+ \wedge^+ \neg[\![B']\!]^\pm} \wedge^+$$

The left premise of this rule is a sequent focused on the positive occurrence of the conflict atom which, by definition, does not appear in the resolvent clause. This atom has no complementary in Δ and this derivation cannot be completed. Thus the decide rule has to be applied first on the coclause containing the negative atom ($i.e.$,$a^- \wedge^+ \neg[\![B]\!]^\pm$) producing the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\vdash a^-, \Delta \Uparrow \cdot}}{\vdash \Delta \Uparrow a^-} \mathscr{S}}{\vdash \Delta \Downarrow a^-} \mathscr{R} \quad \dfrac{\pi'}{\vdash \Delta \Downarrow \neg[\![B']\!]^\pm}}{\vdash \Delta \Downarrow a^- \wedge^+ \neg[\![B]\!]^\pm} \wedge^+$$

The remaining atoms of the coclauses *do appear* in the resolvent clause and the derivation π' can be completed. Indeed in a resolution step, all the sequents focused on a positive member of $\Lambda(B) \cup \Lambda(B')$ would end in success because there is exactly one conflict literal (here a) in each resolved clause of a resolution step, thus each atom $q \in \Lambda(B) \cup \Lambda(B')$ has a complementary atom $\neg q \in \Lambda(C_k)$. The derivation π continues with a decide rule on the second coclause, $a^+ \wedge^+ \neg[\![B']\!]^\pm$, and is able to continue because, at this point, the complementary of the conflict atom a^+ is stored.

The decide expert has many definitions. The first one is straightforward: decide on the first coclause in the order given in the `step` case.

$$\mathscr{D}_e(\texttt{step}(\texttt{br}[I|R]), \texttt{step}(\texttt{br}\ R), \texttt{idx}\ \ I)$$

After a decision on a coclause $\neg C_I \in \Gamma$ producing the partial derivation:

$$\dfrac{\vdash \Theta \Downarrow \neg C_I}{\vdash \Theta \Uparrow \cdot}$$

then follows a number of $\wedge^+$ rules, resulting in $\|\Lambda(\neg C_I)\|$ sequent focused on atoms of the form

$$\vdash \Theta \Downarrow A' \tag{5.6}$$

where $A' \in \Lambda(\neg C_I)$. If the original clauses contain no duplicated literals and the resolvent clauses are factored to also prevent duplicated literals, then only one of the above sequents (5.6) is focused on a conflict atom. If this focused phase follows on the decide on the first coclause, then this atom is negative. In that case, only this premise needs the index of the second coclause to finish the proof. The $\wedge_e^+$ can give this information to one of the premises of a $\wedge^+$ rule and a `step` with an empty list to the other premise. This prevents unnecessary decide rules (and subsequent focused phases) on the other coclause.

$$\wedge_e^+(\mathtt{step}(\mathtt{br}[I]), \mathtt{step}(\mathtt{br}[]), \mathtt{step}(\mathtt{br}[I]))$$
$$\wedge_e^+(\mathtt{step}(\mathtt{br}[I]), \mathtt{step}(\mathtt{br}[I]), \mathtt{step}(\mathtt{br}[]))$$

If the above sequents (5.6) follow from a decide on the second coclause, then the focus is on the positive conflict atom whose complement is already stored and the `step` case is empty.

$$\wedge_e^+(\mathtt{step}(\mathtt{br}[]), \mathtt{step}(\mathtt{br}[]), \mathtt{step}(\mathtt{br}[]))$$

Each premise of this focused phase will end either with an initial on a positive atom or with a release on a negative atom. The initial expert $\mathscr{I}_e$ gives the index `lit` to the kernel.

$$\mathscr{I}_e(\mathtt{step}(\mathtt{br}[]), \mathtt{lit})$$

The release expert $\mathscr{R}_e$ is naïve:

$$\mathscr{R}_e(\mathtt{step}R, \mathtt{step}R)$$

When the list of remaining coclauses is empty, one can decide only on a literal, which brings about an additional definition for the decide expert $\mathscr{D}_e$:

$$\mathscr{D}_e(\mathtt{step}(\mathtt{br}[]), \mathtt{step}(\mathtt{br}[]), \mathtt{lit})$$

Remark 5.3.7. When the first coclause is atomic, the kernel cannot decide on it because it is necessary a negative atom. Indeed, the first resolved coclause must at least have the conflict atom, and the ordering of the coclauses list requires the conflict atom of the first coclause to be the negative occurrence (*e.g.,* resolving a^- and $a^+ \wedge^+ B$).

Some additional agent definitions are needed to handle the special situation of remark 5.3.7 and of the final resolution step. Indeed, the latter is verified with a cut rule whose left premise is of the following form:

$$\vdash \Gamma \Uparrow f^-$$

$$
\begin{array}{ll}
\vee_c^- \ (\mathtt{step}R, \mathtt{step}R) & \mathscr{S}_c(\mathtt{step}R, _, \mathtt{step}R, \mathtt{lit}) \\
\mathscr{D}_e(\mathtt{step}(\mathtt{br}[I|R]), \mathtt{step}R, \mathtt{idx}\ \ I) & f_c^-(\mathtt{step}R, \mathtt{step}R) \\
\mathscr{D}_e(\mathtt{step}(\mathtt{br}[]), \mathtt{step}(\mathtt{br}[]), \mathtt{lit}) & \mathscr{D}_e(\mathtt{step}(\mathtt{br}[I, J]), \mathtt{step}(\mathtt{br}[I]), \mathtt{idx}\ \ J) \\
\wedge_e^+ \ (\mathtt{step}(\mathtt{br}[I]), \mathtt{step}(\mathtt{br}[]), \mathtt{step}(\mathtt{br}[I])) & \mathscr{R}_e(\mathtt{step}R, \mathtt{step}R) \\
\wedge_e^+ \ (\mathtt{step}(\mathtt{br}[I]), \mathtt{step}(\mathtt{br}[I]), \mathtt{step}(\mathtt{br}[])) & \mathscr{I}_e(\mathtt{step}(\mathtt{br}[I]), \mathtt{idx}\ \ I) \\
\wedge_e^+ \ (\mathtt{step}(\mathtt{br}[]), \mathtt{step}(\mathtt{br}[]), \mathtt{step}(\mathtt{br}[])) & \mathscr{I}_e(\mathtt{step}(\mathtt{br}[]), \mathtt{lit})
\end{array}
$$

Figure 5.8: Clerks and Experts working on the `step` case

Therefore, the false clerk f_c^- must be defined.

$$f_c^-(\mathtt{step}R, \mathtt{step}R)$$

When the the first coclause is atomic, one has to decide directly on the second coclause.

$$\mathscr{D}_e(\mathtt{step}(\mathtt{br}[I, J]), \mathtt{step}(\mathtt{br}[I]), \mathtt{idx}\ \ J)$$

The following $\wedge^+$ rules use the definitions of $\wedge_e^+$ given above. The initial expert is also defined, on a `step` case with a non-empty list, to succeed only with the index contained in that list:

$$\mathscr{I}_e(\mathtt{step}(\mathtt{br}[I]), \mathtt{idx}\ \ I)$$

To summarize, all the team of agents is shown in figure 5.8.

Remark 5.3.8. This definition of agents relies heavily on the fact that the resolved clauses in the tuples are ordered. This makes for a concise and precise definition. Similar to the discussion in remark 5.2.1, one can extend these definitions to deal with less detailed proof evidence formats.

Extending to first-order resolution

Thanks in no small part to the relational aspect of the Foundational Proof Certification framework, the semantics for propositional resolution seen in section 5.3.2 can be easily extended to handle first-order resolution refutations. Support for quantifiers through clerks and experts is added to the unmodified `p.r.i.c.e.` fragment for the *step checker* process.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\Xi_7 \vdash \langle \texttt{lit}, a^- \rangle, \langle \texttt{lit}, a^+ \rangle, \Gamma \Downarrow a^+}\ \mathscr{I}_e(\Xi_7, \texttt{lit})
}{\Xi_7 \vdash \langle \texttt{lit}, a^- \rangle, \langle \texttt{lit}, a^+ \rangle, \Gamma \Uparrow \cdot}\ \mathscr{D}_e(\Xi_7, \Xi_7, \texttt{lit})
}{\Xi_7 \vdash \langle \texttt{lit}, a^+ \rangle, \Gamma \Uparrow a^-}\ \mathscr{S}_c(\Xi_7, _, \Xi_7, \texttt{lit})
}{\Xi_7 \vdash \langle \texttt{lit}, a^+ \rangle, \Gamma \Downarrow a^-}\ \mathscr{R}_e(\Xi_7, \Xi_7)
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\Xi_6 \vdash \langle \texttt{lit}, b^- \rangle, \langle \texttt{lit}, a^+ \rangle, \Gamma \Downarrow b^+}\ \mathscr{I}_e(\Xi_7, \texttt{lit})
}{\Xi_6 \vdash \langle \texttt{lit}, b^- \rangle, \langle \texttt{lit}, a^+ \rangle, \Gamma \Uparrow \cdot}\ \mathscr{D}_e(\Xi_6, \Xi_7, \texttt{idx 3})
}{\Xi_6 \vdash \langle \texttt{lit}, a^+ \rangle, \Gamma \Uparrow b^-}\ \mathscr{S}_c(\Xi_6, _, \Xi_6, \texttt{lit})
}{\Xi_6 \vdash \langle \texttt{lit}, a^+ \rangle, \Gamma \Downarrow b^-}\ \mathscr{R}_e(\Xi_6, \Xi_6)
}{
\dfrac{
\dfrac{\Xi_6 \vdash \langle \ \texttt{lit}, a^+ \rangle, \Gamma \Downarrow a^- \wedge^+ b^-}{\Xi_2 \vdash \langle \texttt{lit}, a^+ \rangle, \Gamma \Uparrow \cdot}\ \mathscr{D}_e(\Xi_2, \Xi_6, \texttt{idx 1})
}{\Xi_2 \vdash \Gamma \Uparrow a^+}\ \mathscr{S}_c(\Xi_2, _, \Xi_2, \texttt{lit})
}\ \wedge_e^+(\Xi_6, \Xi_7, \Xi_6)
$$

$$\Xi_6 = \texttt{step}[3] \quad \Xi_7 = \texttt{step}(\texttt{br}[])$$

Figure 5.9: The step region of the derivation π for example 5.3.1 in figure 5.5

$$\forall_c(\texttt{step}R, \lambda x.(\texttt{step}R)) \qquad\qquad \exists_e(\texttt{step}R, \texttt{step}R, V)$$

Figure 5.10: Clerks and Experts working on the step case

Aside from the clauses, which are now considered first-order, the previous notion of proof evidence remains unchanged. The added agents are naïve and instantiations of quantifiers are left out to emphasize the proof reconstruction capabilities offered by unification, one of the desiderata in section 3.1.5.

5.3.3 Interpreting a hyperresolution step

Resolution is arguably one of the most widely used proof technique in the area of automated deduction and gave rise to several extensions and refinements. This section explores a refinement called *hyperresolution*[Robinson, 1965a].

Definition 5.3.4. *An* affirmative *clause is a disjunction of only non-negated literals. A* positive resolution step *is a step where one of the resolved clauses is affirmative.* •

The refinement to positive resolution stems from the observation that resolution retains completeness when every resolution step is restricted to be a positive resolution step. In the same paper[Robinson, 1965a], positive resolution is generalized to hyperresolution.

Consider a positive resolution step between an affirmative clause $L_1 \vee Q$ and another clause $\neg L_1 \vee \neg L_2 \vee \cdots \vee \neg L_n \vee P$ where P is an affirmative (sub)clause. The resolvent clause will be $\neg L_2 \vee \cdots \vee \neg L_n \vee P \vee Q$. If $n > 1$, then this clause contains at least a negated literal, making it a non-affirmative clause which means, in a positive resolution step, it can only be resolved with an affirmative clause. This affirmative clause will have exactly one literal that is the compliment of one of the L_i $(i = 2 \cdots n)$ and if the resolvent clause is still not affirmative $(n > 3)$, it will also have to be resolved with an affirmative clause and so on.

A hyperresolution refutation groups together the steps of positive resolution where the clause $\neg L_1 \vee \neg L_2 \vee \cdots \vee \neg L_n \vee P$ participates. More formally:

Definition 5.3.5. *A* hyperresolution *step is a refinement of a resolution step where a number of affirmative clauses are resolved with a non affirmative clause called a* nucleus *to yield an affirmative clause as resolvent. The notion of a sequence of refutations is the same as that of definition 5.3.3,* i.e.,*a sequence of tuples $\langle \mathcal{S}, K, C_K \rangle$, but now C_K is affirmative and $\mathcal{S}$ is a pair $\langle AL, Nu \rangle$ where AL is the list of indexes of affirmative clauses and Nu is the index of the nucleus. A negated affirmative clause is called* affirmative coclause *and a negated nucleus is called a* conucleus. •

The `p.r.i.c.e.` given in section 5.3.1 for the region responsible for checking the sequence of refutations can mostly be used as it is for hyperresolution. Similarly, a hyperresolution step follows the same interpretation as that of a resolution step seen in 5.3.2 and its polarity choice and indexing are the same. Slight changes are required for the cases and agents, however, and they are introduced in the following section.

Agents and cases of a hyperresolution step

Example 5.3.2. *The agents will be introduced in order of appearance and one can follow their definitions in figure 5.12 that shows the hyperresolution step resolving the affirmative clauses* $a \vee p, b$ *and the nucleus* $\neg a \vee \neg b \vee q$*, where* a, b, c, p, q *are atoms, into the affirmative clause* $p \vee q$*. Polarization produces the coclauses* $a^- \wedge^+ p^-, b^-, a^+ \wedge^+ b^+ \wedge^+ q^-$ *that resolve into the clause* $p^+ \vee^- q^+$*.* $\odot$

The constructor for the `stuple` type for this region is:

$$\texttt{type}\quad\texttt{hyp}\quad\texttt{list}\ \mathbb{N} \to \mathbb{N} \to \texttt{list}\ \xi \to \texttt{stuple}.$$

where the arguments are the resolved clauses in two pieces: the list of indexes for affirmative clauses AL and the index of the nucleus Nu. The purpose of the third argument, the list of indexes $Atms$, will be explained later.

In keeping with remark 5.3.4, the `step` case is written `step` $AL\,Nu\,[\texttt{lit}]$ instead of `step` (`hyp` $AL\,Nu\,[\texttt{lit}]$).

The end sequent of the left premise of a cut rule checking the step $\langle\langle AL, Nu\rangle, K, C_K\rangle$ is as follows:

$$\texttt{step}\ AL\,Nu\,[\texttt{lit}] \vdash \Delta \Uparrow C_k$$

The invertible phase applies a series of $\vee^-$ rules augmented with the following naïve clerk:

$$\vee_c^-(\texttt{step}\ AL\,Nu\,Atms, \texttt{step}\ AL\,Nu\,Atms)$$

Each time an atom is reached, it is stored under the guidance of the following clerk:

$$\mathscr{S}_c(\texttt{step}\ AL\,Nu\,Atms, _, \texttt{step}\ AL\,Nu\,Atms, \texttt{lit})$$

When the invertible phase is over, a decide rule is applied.

Remark 5.3.9. Before defining the decide expert $\mathscr{D}_e$, one can make several remarks:

- Because the affirmative clauses contain only non negated literals, their polarized version contains only positive atoms, and the corresponding affirmative coclauses contain only negative atoms. This means that a decide on an affirmative non atomic coclause will result in a focused phase of which all the premises end in a release *i.e.,*a positive phase that will not fail.

- The nucleus is the only clause containing negated literals and those literals are exactly the conflict literals. These literals become positive atoms in the conucleus and are the only atoms whose complements are not in the resolvent

clause's atoms but in the affirmative coclauses. This entails that no decide on the conucleus can succeed until all the complementary (negative) atoms of the (positive) conflict atoms are stored, *i.e.,*until all the affirmative coclauses have been processed and their conflict literals stored.

- An atomic affirmative coclause contains only a negative atom and is thus unsuited for a decide rule.

Because it is "safe" to decide on a non atomic affirmative clause, the decide expert can pick the indexes in order:

$$\mathscr{D}_e(\texttt{step}\ \ [I|AL]\ Nu\ Atms, \texttt{step}\ \ AL\ Nu\ Atms, \texttt{idx}\ \ I)$$

The focused phase starts on a positive conjunction. Because there is only one nucleus, all the affirmative clauses have exactly one atom that can only be mated with an atom in the nucleus, all the others appear in the resolvent. This observation allows for a $\wedge_e^+$ definition similar to the one seen for the binary resolution, where only one of the premises receive the rest of the resolved clauses while all the other finish after one decision:

$$\wedge_e^+(\texttt{step}\ \ AL\ Nu\ Atms, \texttt{step}\ \ AL\ Nu\ Atms, \texttt{oneOf}\ \ \texttt{[lit]})$$

$$\wedge_e^+(\texttt{step}\ \ AL\ Nu\ Atms, \texttt{oneOf}\ \ \texttt{[lit]}, \texttt{step}\ \ AL\ Nu\ Atms)$$

All the premises eventually focus on a negative atom and thus release. For the premises focused on the non conflict literals, the release expert $\mathscr{R}_e$ is already defined on the default case `oneOf`. The following additional definition of $\mathscr{R}_e$ is for the one premise with the focus on the conflict (negative) atom:

$$\mathscr{R}_e(\texttt{step}\ \ AL\ Nu\ Atms, \texttt{step}\ \ AL\ Nu\ Atms)$$

The decide expert has also a definition for the case where the head of the affirmative list AL is the index of an atom. If it is, that index is added to the third argument which holds the indexes of the formulas that were not subject to a decide, and the decide expert analyzes the following index.

$$\mathscr{D}_e(\texttt{step}\ \ [I|AL]\ Nu\ Atms, \Xi, X)\ \ \texttt{:-}\ \ \mathscr{D}_e(\texttt{step}\ \ AL\ Nu\ [\texttt{idx}\ \ I|Atms], \Xi, X)$$

Remark 5.3.10. This last definition of the decide expert involves an unwanted search space because this definition puts the index `idx` I in the third argument without knowledge of whether or not it is the index of an atomic coclause. The overall derivation cannot succeed when the index of a non atomic coclause is put in the third argument and, therefore, not decided on. This issue is ignored for now but discussed further in this chapter.

A final definition of the decide expert is needed for when the list of affirmative clauses is empty. In this case, if all the non atomic coclauses were decided on and decomposed, all the complementary atoms of the conflict literals withing the conucleus are present in the storage. The decide expert simply decides on the conucleus:

$$\mathscr{D}_e(\texttt{step}\ []\ Nu\ Atms, \texttt{last}\ \ Atms, \texttt{idx}\ \ Nu)$$

The returned case is a new case called `last` containing the list of atoms. If the conucleus is an atom, it is necessary a positive atom because it must contain the negated conflict literal which becomes a positive atom in the conucleus. Thus, the next rule is an initial and is guided by the following expert:

$$\mathscr{I}_e(\texttt{last}\ \ Atoms, I)\ \ \texttt{:-}\ \ I \in Atoms$$

If the conucleus is not atomic, it is a conjunction and thus the positive conjunction expert $\wedge_e^+$ must be defined on the `last` case. There is no advantage in ordering the premises so an arbitrary order is chosen.

$$\wedge_e^+(\texttt{last}\ \ Atoms, \texttt{last}\ \ Atoms, \texttt{last}\ \ Atoms)$$

When arriving at the end of the focused phase with a sequent focused on a positive atom, this atom is necessarily a conflict literal and must end with a member of the list *Atms*. The above definition of initial expert $\mathscr{I}_e$ still works. If the focused phase ends with a sequent focused on a negative atom, this atom does not come from a conflict literal and therefore is complemented in the resolvent clause's stored atoms. The phase ends in a release:

$$\mathscr{R}_e(\texttt{last}\ \ Atoms, \texttt{last}\ \ Atoms)$$

After a store the case changes to a `oneOf` case:

$$\mathscr{S}_c(\texttt{last}\ \ Atoms, _, \texttt{oneOf [lit]}, \texttt{lit})$$

The team of clerks and experts is summarized in figure 5.11

Reviewing polarity assignment

One can take advantage of the structure of a hyperresolution rule to shorten the size of the proofs of the left premises of cuts. To answer the unwanted behavior stated in remark 5.3.10, the effect of another polarity choice for the atoms is discussed here.

$$\vee_c^- (\texttt{step}\ AL\ Nu\ Atms, \texttt{step}\ AL\ Nu\ Atms)$$
$$\mathscr{S}_c(\texttt{step}\ AL\ Nu\ Atms, _, \texttt{step}\ AL\ Nu\ Atms, \texttt{lit})$$
$$\mathscr{D}_e(\texttt{step}\ [I|AL]\ Nu\ Atms, \texttt{step}\ AL\ Nu\ Atms, \texttt{idx}\ I)$$
$$\mathscr{R}_e(\texttt{step}\ AL\ Nu\ Atms, \texttt{step}\ AL\ Nu\ Atms)$$
$$\wedge_e^+ (\texttt{step}\ AL\ Nu\ Atms, \texttt{step}\ AL\ Nu\ Atms, \texttt{oneOf [lit]})$$
$$\wedge_e^+ (\texttt{step}\ AL\ Nu\ Atms, \texttt{oneOf [lit]}, \texttt{step}\ AL\ Nu\ Atms)$$
$$\mathscr{D}_e(\texttt{step}\ [I|AL]\ Nu\ Atms, \Xi, X)\ \ \texttt{:-}\ \ \mathscr{D}_e(\texttt{step}\ AL\ Nu\ [\texttt{idx}\ I|Atms], \Xi, X)$$
$$\mathscr{D}_e(\texttt{step}\ []\ Nu\ Atms, \texttt{last}\ Atms, \texttt{idx}\ Nu)$$
$$\mathscr{I}_e(\texttt{last}\ Atoms, I)\ \ \texttt{:-}\ \ I \in Atoms$$
$$\wedge_e^+ (\texttt{last}\ Atoms, \texttt{last}\ Atoms, \texttt{last}\ Atoms)$$
$$\mathscr{R}_e(\texttt{last}\ Atoms, \texttt{last}\ Atoms)$$
$$\mathscr{S}_c(\texttt{last}\ Atoms, _, \texttt{oneOf [lit]}, \texttt{lit})$$

Figure 5.11: Team of agents for checking a hyperresolution step.

The polarity assignment for the connectives is the same as in figure 5.7 the polarity of literals is reversed:

$$[\![a]\!]^{\pm} = a^{-} \quad [\![\neg a]\!]^{\pm} = a^{+}$$

With this polarity, one can make symmetrical notes to those in remark 5.3.9. Now all the conflict literals in the conucleus are negative atoms, and the positive atoms in the conucleus have their complements in the resolvent clause (*i.e.,*already stored under index `lit`). So the decide cannot fail when done on the conucleus.

The premises of the positive phase engaged on the conucleus are of two kinds:

- either they are focused sequents on positive atoms, then these atoms are necessarily in the resolvent clause and the premise is accepted through the initial rule or
- they are focused sequents on negative atoms, which are the conflict literals, then the sequents are the conclusion of a release rule.

On these latter sequents, at most one extra decide rule is needed on the unique coclause containing the complement of the conflict literal.

In the special case where the conucleus is an atomic formula, it is necessarily a negative atomic formula (because it has to contain at least one conflict literal, which

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\Xi_4 \vdash \Gamma_2, (a^-)_{\texttt{lit}} \Downarrow a^+}\,\mathscr{I}_e(\Xi_4, \texttt{lit})
\qquad \pi
}{\Xi_4 \vdash \Gamma_2, (a^-)_{\texttt{lit}} \Downarrow a^+ \wedge^+ b^+ \wedge^+ q^-}\,\wedge_e^+(\Xi_4, \Xi_4, \Xi_4)
}{\Xi_2 \vdash \Gamma_2, (a^-)_{\texttt{lit}} \Uparrow \cdot}\,\mathscr{D}_e(\Xi_2, \Xi_4, \texttt{idx 3})
}{\Xi_2 \vdash \Gamma_2 \Uparrow a^-}\,\mathscr{S}_c(\Xi_2, _, \Xi_2, \texttt{lit})
}{\Xi_2 \vdash \Gamma_2 \Downarrow a^-}\,\mathscr{R}_e(\Xi_2, \Xi_2)
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\Xi_5 \vdash \Gamma_2, (p^-)_{\texttt{lit}} \Downarrow p^+}\,\mathscr{I}_e(\Xi_5, \texttt{lit})
}{\Xi_3 \vdash \Gamma_2, (p^-)_{\texttt{lit}} \Uparrow \cdot}\,\mathscr{D}_e(\Xi_3, \Xi_5, \texttt{lit})
}{\Xi_3 \vdash \Gamma_2 \Uparrow p^-}\,\mathscr{S}_c(\Xi_3, _, \Xi_3, \texttt{lit})
}{\Xi_3 \vdash \Gamma_2 \Downarrow p^-}\,\mathscr{R}_e(\Xi_3, \Xi_3)
}{
\dfrac{
\dfrac{\Xi_2 \vdash \Gamma_2 \Downarrow a^- \wedge^+ p^-}{\Xi_1 \vdash \Gamma_2 \Uparrow \cdot}\,\mathscr{D}_e(\Xi_1, \Xi_2, \texttt{idx 1})
}{\Xi_1 \vdash \Gamma_1 \Uparrow p^+ \vee^- q^+}\,\vee_c^-(\Xi_1, \Xi_1) + 2 \times \mathscr{S}_c(\Xi_1, _, \Xi_1, \texttt{lit})
}\,\wedge_e^+(\Xi_2, \Xi_2, \Xi_3)
$$

$$
\dfrac{
\dfrac{
\dfrac{}{\Xi_4 \vdash \Gamma_2, (a^-)_{\texttt{lit}} \Downarrow b^+}\,\mathscr{I}_e(\Xi_4, \texttt{idx 2})
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\Xi_5 \vdash \Gamma_2, (a^-)_{\texttt{lit}}, (q^-)_{\texttt{lit}} \Downarrow q^+}\,\mathscr{I}_e(\Xi_5, \texttt{lit})
}{\Xi_4 \vdash \Gamma_2, (a^-)_{\texttt{lit}}, (q^-)_{\texttt{lit}} \Uparrow \cdot}\,\mathscr{D}_e(\Xi_4, \Xi_5, \texttt{lit})
}{\Xi_4 \vdash \Gamma_2, (a^-)_{\texttt{lit}} \Uparrow q^-}\,\mathscr{S}_c(\Xi_4, _, \Xi_4, \texttt{lit})
}{\Xi_4 \vdash \Gamma_2, (a^-)_{\texttt{lit}} \Downarrow q^-}\,\mathscr{R}_e(\Xi_4, \Xi_4)
}{\Xi_4 \vdash \Gamma_2, (a^-)_{\texttt{lit}} \Downarrow b^+ \wedge^+ q^-}\,\wedge_e^+(\Xi_4, \Xi_4, \Xi_4)
}{\pi}
$$

with

$$
\Gamma_1 = \{(a^- \wedge^+ p^-)_{\texttt{idx 1}}, (b^-)_{\texttt{idx 2}}, (a^+ \wedge^+ b^+ \wedge^+ q^-)_{\texttt{idx 3}}\} \quad \Gamma_2 = \Gamma_1 \cup \{(p^+)_{\texttt{lit}}, (q^+)_{\texttt{lit}}\}
$$

$$
\begin{aligned}
\Xi_1 &= \texttt{step}\ [1,2][3][\texttt{lit}] & \Xi_2 &= \texttt{step}\ [2][3][\texttt{lit}] \\
\Xi_3 &= \texttt{oneOf [lit]} & \Xi_4 &= \texttt{last}\ [\texttt{idx 2}, \texttt{lit}] \\
\Xi_5 &= \texttt{done} & &
\end{aligned}
$$

Figure 5.12: Derivation of example 5.3.2

$$
\begin{aligned}
&\vee_c^- (\texttt{step}\ \ AL\, Nu, \texttt{step}\ \ AL\, Nu)\\
&\mathscr{S}_c(\texttt{step}\ \ AL\, Nu, _, \texttt{step}\ \ AL\, Nu, \texttt{lit})\\
&\mathscr{D}_e(\texttt{step}\ \ AL\, Nu, \texttt{oneOf}\ \ AL, \texttt{idx}\ \ Nu)\\
&\mathscr{D}_e(\texttt{oneOf}\ \ AL, \texttt{initWith}\ [\texttt{lit}], \texttt{idx}\ \ I)\ \ \texttt{:-}\ \ I \in AL\\
&\mathscr{D}_e(\texttt{step}\ \ [I]\, Nu, \texttt{initWith}\ [\texttt{lit}, \texttt{idx}\ \ Nu], \texttt{idx}\ \ I)\\
&\wedge_e^+ (\texttt{oneOf}\ \ AL, \texttt{oneOf}\ \ AL, \texttt{oneOf}\ \ AL)\\
&\mathscr{R}_e(\texttt{oneOf}\ \ AL, \texttt{oneOf}\ \ AL)\\
&\mathscr{S}_c(\texttt{oneOf}\ \ AL, _, \texttt{oneOf}\ \ AL, \texttt{lit})\\
&\mathscr{I}_e(\texttt{oneOf}\ \ AL, \texttt{lit})
\end{aligned}
$$

Figure 5.13: Modified clerks and experts for reversed atoms' polarity

will be negative). This means the list of affirmative clauses is a singleton, because there is only one conflict literal and there must be as many coclauses as conflict literals.

Reviewing example 5.3.2 with this new polarization, one gets the coclauses b^+, $a^+ \wedge^+ p^+$, $a^- \wedge^+ b^- \wedge^+ q^+$ resolving into the clause $p^- \vee^- q^-$.

The modified team of clerks and experts is given in figure 5.13, they are defined on a modified `step` case now only having as arguments the list of affirmative coclauses indexes AL and the index of conucleus Nu. An auxiliary case, `oneOf` holds a list of indexes the decide expert can decide on.

5.4 Expansion Trees

From Herbrand's theorem [Herbrand, 1930], it is known that recording instantiations of existential quantifiers is sufficient to describe a classical cut-free proof of a formula in prenex normal form. This compact formalism is used by many automated proof tools, especially for instantiation based reasoning [Korovin, 2013].

Building on Herbrand's insight, Miller defined *expansion trees* for full higher-order logic [Miller, 1987] as a structure to record such instantiation information without restriction to prenex normal form.

Remark 5.4.1. While the terminology used in [Miller, 1987] include the terms "positive" and "negative", these terms are unrelated to their meaning in this thesis. If one is familiar with the original work by Miller [1987], one notices that the present section does not mention certain notions that are irrelevant in this setting. In particular, dual expansion trees are not use because the present work is built on one sided *LKF* where the formulas are in negation normal form.

Decades later, expansion trees were generalized [Hetzl and Weller, 2013] to describe not only analytic cut-free proofs but also proofs with cuts. This section, however, investigates only the original setting by [Miller, 1987] for first-order proofs.

Definition 5.4.1. *An* expansion tree *E of a formula A is defined inductively as follows:*

- *If A is a unit or a literal then E is a final node labelled A.*
- *If E_1 and E_2 are expansion trees of formulas A_1 and A_2, and $\circ \in \{\wedge, \vee\}$ then $E_1 \dot{\circ} E_2$ is an expansion tree of formula $A_1 \circ A_2$ with top node $\dot{\circ}$.*
- *If E is an expansion tree of formula $[y/x]A$ and y is not an eigenvariable of any node in E then $\dot{\forall} +^{y} E$ is an expansion tree of formula $\forall x.A$ with top node $\dot{\forall}$. The variable y is called a* selection variable *of its top node.*
- *If $\{t_1, \ldots, t_n\}$ is a set of terms and $E_1, \ldots, E_n$ are expansion trees of formulas $[t_i/x]A$ for $i = 1, \ldots, n$, then $E' = \dot{\exists} +^{t_1} E_1 \ldots +^{t_n} E_n$ is an expansion tree of formula $\exists x.A$ with top node $\dot{\exists}$. The terms $t_1, \ldots, t_n$ are known as the* expansion terms *of its top node.*

•

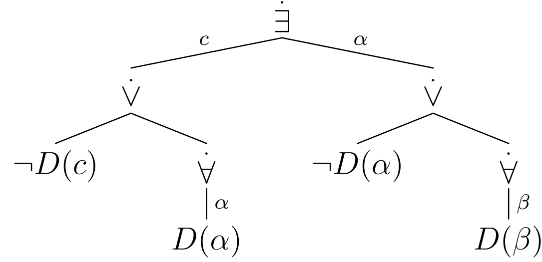
Remark 5.4.2. A variable can be both a selection variable for a universal node $\dot{\forall}$ and appear in an expansion term for an existential node $\dot{\exists}$, just as an eigenvariable can

appear inside a witness for an existential after having been introduced by a universal rule application.

Example 5.4.1. *The expansion tree corresponding to the proof of the Drinker's Paradox,* $\exists x.(\neg D(x) \vee \forall y.D(y))$ *is as follows:*

$$E \;=\; \dot{\exists} +^{c} [\neg D(c) \mathbin{\dot{\vee}} (\dot{\forall} +^{\alpha} D(\alpha))] +^{\alpha} [\neg D(\alpha) \mathbin{\dot{\vee}} (\dot{\forall} +^{\beta} D(\beta))]$$

which gives the following graphical representation:



$\odot$

Definition 5.4.2. *Let* E *be an expansion tree, the deep formula* $Dp(E)$ *is defined inductively as*

- $Dp(L) = L'$ *where* L *is a leaf node annotated with literal or unit* L'
- $Dp(E_1 \mathbin{\dot{\circ}} E_2) = Dp(E_1) \circ Dp(E_2)$
- $Dp(\dot{\exists} +^{t_1} E_1 \ldots +^{t_n} E_n) = \bigvee_{i=1}^{n} Dp(E_i)$
- $Dp(\dot{\forall} +^{t} E) = Dp(E)$

$\bullet$

Definition 5.4.3. *A node* n dominates *a node* m *if* n *appears higher than* m *on a common branch (expansion trees are viewed with the root on the top and leaves at the bottom). Let* E *be an expansion tree and let* $\prec_E$ *be the binary relation on the occurrences of expansion terms in* E *defined by* $t \prec_E s$ *if there is an* α *which is free in* s *and is a selection variable of a node dominated by* t*. Then, the transitive closure of* $\prec_E$*, designated by* $\prec_E^*$*, is called the* dependency relation *of* E*.* e.g., *in the tree of example 5.4.1, the expansion terms of the root,* c *and* α*, satisfy the relation* $c \prec_E \alpha$ *because* α *(which is free in* α*) is also the selection variable of a* $\dot{\forall}$ *node dominated by* c*.* $\bullet$

Definition 5.4.4. *An expansion* tree E *on its own is not considered proof. It constitutes an expansion* proof *only if:*

- *$Dp(E)$ is a tautology*
- *The dependency relation $\prec_E^*$ is acyclic.*

•

5.4.1 Expansion trees in *LK*

In this section, a modified classical sequent calculus for expansion trees is introduced. This sequent calculus shows how to elaborate an *LK* proof from an expansion tree and will simplify understanding of the semantics definition in next section.

Remark 5.4.3. Once an LK proof is obtained, the formula is proved to be valid. There is no further need of proving the deep formula of the expansion tree tautologous or the dependency relation acyclic (definition 5.4.4). As a result, while the expansion tree alone does not constitute sufficient *proof*, it is enough *evidence* to guide the checker towards proof of a given formula.

Definition 5.4.5. Sequentializing *an expansion tree is the process of building a sequent calculus proof out of that expansion tree. An* expansion sequent *is of the form*

$$\sigma \ddagger \Psi \ddagger \Phi \vdash \mathscr{A} \ddagger \mathscr{E} \ddagger \Delta$$

where σ is a set of substitutions of the form $[y/u]$ where y is an eigenvariable and u is a selection variable, Ψ is a set of tuples $\langle T, \tau, I\rangle$ where T is an expansion term, τ is an expansion tree and I is an index, Φ is a list of expansion trees, $\mathscr{A}$ is a set of literals, $\mathscr{E}$ is a set indexed existential formulas of the form $\langle I, \exists x.F\rangle$ and finally Δ is a list of formulas. The expansion sequent calculus *is the sequent calculus given in 5.14. Because the deep formula of the expansion tree is not explicitly verified to be tautologous, there is no need to label the leaves of the tree with the atoms and units. Thus the definition of expansion tree changes slightly in that the leaves are simply labeled by the symbol $\diamond$, adding to the compactness of the proof certificate.* •

Remark 5.4.4. One notices in the representation of expansion trees that both the disjuncts are present at the disjunction nodes $\dot{\vee}$, which influenced the rule for the disjunction in the expansion sequent calculus to be invertible. Section 2.2.1, discussing invertibility in *LK*, mentions that existential formulas are the only *LK* formulas for which there are no invertible right-introduction rule. Contraction can, then, be restricted to existential formulas. This relates to Herbrand's expansion and to the branching in expansion trees at existential nodes $\dot{\exists}$.

$$\frac{\sigma \ddagger \Psi \ddagger \tau_1, \Phi \vdash \mathscr{A} \ddagger \mathscr{E} \ddagger F_1, \Delta \quad \sigma \ddagger \Psi \ddagger \tau_2, \Phi \vdash \mathscr{A} \ddagger \mathscr{E} \ddagger F_2, \Delta}{\sigma \ddagger \Psi \ddagger \tau_1 \dot{\wedge} \tau_2, \Phi \vdash \mathscr{A} \ddagger \mathscr{E} \ddagger F_1 \wedge F_2, \Delta}\ \dot{\wedge}$$

$$\frac{\sigma \ddagger \Psi \ddagger \tau_1, \tau_2, \Phi \vdash \mathscr{A} \ddagger \mathscr{E} \ddagger F_1, F_2, \Delta}{\sigma \ddagger \Psi \ddagger \tau_1 \dot{\vee} \tau_2, \Phi \vdash \mathscr{A} \ddagger \mathscr{E} \ddagger F_1 \vee F_2, \Delta}\ \dot{\vee}$$

$$\frac{\sigma \ddagger \Psi \ddagger \Phi \vdash a, \mathscr{A} \ddagger \mathscr{E} \ddagger \Delta}{\sigma \ddagger \Psi \ddagger \dot{a}, \Phi \vdash \mathscr{A} \ddagger \mathscr{E} \ddagger a, \Delta}\ atm$$

$$\frac{\{[y/u]\} \uplus \sigma \ddagger \Psi \ddagger \tau, \Phi \vdash \mathscr{A} \ddagger \mathscr{E} \ddagger F[y/x], \Delta}{\sigma \ddagger \Psi \ddagger \dot{\forall} +^u \tau, \Phi \vdash \mathscr{A} \ddagger \mathscr{E} \ddagger \forall x.F, \Delta}\ \dot{\forall}^{\natural}$$

$$\frac{\sigma \ddagger \{\langle t_1, \tau_1, i\rangle, \cdots, \langle t_n, \tau_n, i\rangle\} \uplus \Psi \ddagger \Phi \vdash \mathscr{A} \ddagger \{\langle i, \exists x.F\rangle\} \uplus \mathscr{E} \ddagger \Delta}{\sigma \ddagger \Psi \ddagger \dot{\exists} +^{t_1} \tau_1 + \cdots +^{t_n} \tau_n, \Phi \vdash \mathscr{A} \ddagger \mathscr{E} \ddagger \exists x.F, \Delta}\ \dot{\exists}^{\natural}$$

$$\frac{\sigma \ddagger \Psi \ddagger \tau \vdash \mathscr{A} \ddagger \{\langle i, \exists x.F\rangle\} \uplus \mathscr{E} \ddagger F[t\sigma/x]}{\sigma \ddagger \{\langle t, \tau, i\rangle\} \uplus \Psi \ddagger \cdot \vdash \mathscr{A} \ddagger \{\langle i, \exists x.F\rangle\} \uplus \mathscr{E} \ddagger \cdot}\ instantiate\star$$

$$\frac{a \text{ atomic}}{\sigma \ddagger \cdot \ddagger \cdot \vdash a, \neg a, \mathscr{A} \ddagger \mathscr{E} \ddagger \cdot}\ init$$

Figure 5.14: Expansion classical sequent calculus.In $\star$, the proviso is that t is minimal, and thus $t\sigma$ is a sequent-level term. In $\natural$, i and y are fresh (do not appear anywhere in the conclusion of the rule).

Definition 5.4.6. *The selection variable u of a universal node $\dot{\forall} +^u \tau$ is* introduced *when a $\dot{\forall}$ rule is applied on that node, generating a fresh eigenvariable y for the corresponding formula $\forall x F$ and adding the substitution $[y/u]$ to the list of substitutions σ, y is called the* mirror *of u. An expansion term is called* minimal *if it doesn't depend on any non-*introduced *selection variable. A branch of an existential node is* unlocked *(resp* locked*) if its expansion term is* minimal *(resp. not* minimal*). A tree-level expansion term can be used as a sequent-level witness in an* instantiate *rule only if it is minimal and only after substituting all its (tree-level) selection variables by their mirror (sequent-level) eigenvariables, which requires their presence in σ.* •

Proposition 5.4.1. *Provability of an expansion sequent $\cdot \ddagger \cdot \ddagger \tau \vdash \cdot \ddagger \cdot \ddagger F$ entails provability of the LK sequent $\vdash F$*

Proof. By erasing the left-hand side of the expanded sequents in an expansion sequent calculus proof, one gets *LK* sequents in a valid *LK* proof. (By induction on the derivation of the expansion proof) □

$$
\begin{array}{cl}
\hline
[s/\beta][z/\alpha] \ddagger \cdot \ddagger \cdot \vdash D(s), \neg D(z), D(z), \neg D(c) \ddagger \langle i, F\rangle \ddagger \cdot & \textit{init} \\
\hline
[s/\beta][z/\alpha] \ddagger \cdot \ddagger D(\beta) \vdash \neg D(z), D(z), \neg D(c) \ddagger \langle i, F\rangle \ddagger D(s) & \textit{atom} \\
\hline
[z/\alpha] \ddagger \cdot \ddagger \dot\forall +^{\beta} D(\beta) \vdash \neg D(z), D(z), \neg D(c) \ddagger \langle i, F\rangle \ddagger \forall y.D(y) & \dot\forall \\
\hline
[z/\alpha] \ddagger \cdot \ddagger \neg \dot D(z), \dot\forall +^{\beta} D(\beta) \vdash D(z), \neg D(c) \ddagger \langle i, F\rangle \ddagger \neg D(z), \forall y.D(y) & \textit{atom} \\
\hline
[z/\alpha] \ddagger \cdot \ddagger \tau_3 \vdash D(z), \neg D(c) \ddagger \langle i, F\rangle \ddagger \neg D(z) \vee \forall y.D(y) & \dot\vee \\
\hline
[z/\alpha] \ddagger \langle \alpha, \tau_3, i\rangle \ddagger \cdot \vdash D(z), \neg D(c) \ddagger \langle i, F\rangle \ddagger \cdot & \textit{instantiate} \\
\hline
[z/\alpha] \ddagger \langle \alpha, \tau_3, i\rangle \ddagger \dot D(\alpha) \vdash \neg D(c) \ddagger \langle i, F\rangle \ddagger D(z) & \textit{atom} \\
\hline
\cdot \ddagger \langle \alpha, \tau_3, i\rangle \ddagger \dot\forall +^{\alpha} D(\alpha) \vdash \neg D(c) \ddagger \langle i, F\rangle \ddagger \forall y.D(y) & \dot\forall \\
\hline
\cdot \ddagger \langle \alpha, \tau_3, i\rangle \ddagger \neg \dot D(c), \dot\forall +^{\alpha} D(\alpha) \vdash \cdot \ddagger \langle i, F\rangle \ddagger \neg D(c), \forall y.D(y) & \textit{atom} \\
\hline
\cdot \ddagger \langle \alpha, \tau_3, i\rangle \ddagger \tau_2 \vdash \cdot \ddagger \langle i, F\rangle \ddagger \neg D(c) \vee \forall y.D(y) & \dot\vee \\
\hline
\cdot \ddagger \langle \alpha, \tau_3, i\rangle, \langle c, \tau_2, i\rangle \ddagger \cdot \vdash \cdot \ddagger \langle i, F\rangle \ddagger \cdot & \textit{instantiate}\star \\
\hline
\cdot \ddagger \cdot \ddagger \tau_1 \vdash \cdot \ddagger \cdot \ddagger F & \dot\exists
\end{array}
$$

$$\tau_1 = \dot\exists +^{c} \tau_2 +^{\alpha} \tau_3 \quad \tau_2 = [\neg D(c) \mathbin{\dot\vee} (\dot\forall +^{\alpha} D(\alpha))] \quad \tau_3 = [\neg D(\alpha) \mathbin{\dot\vee} (\dot\forall +^{\beta} D(\beta))]$$
$$F = \exists x.(\neg D(x) \vee \forall y.D(y))$$

Figure 5.15: Expansion sequent proof for example 5.4.1. At $\star$, the branch labeled α cannot expand yet, α contains a non introduced selection variable: α itself.

The expansion sequent calculus proof of example 5.4.1 is shown in figure 5.15.

5.4.2 Sequentialization of expansion trees to LKF^a

After seeing the sequentialization of an expansion trees in *LK*, this section defines the `p.r.i.c.e.` for checking expansion trees. Augmentation of *LKF* sequents through indexes and cases will closely resemble the expansion of the sequents seen in definition 5.4.5. Augmentation of *LKF* rules with agents is, in effect, similar to the augmentation of rules that yields the expanded sequent calculus seen in 5.14. The

type of expansion trees is given the name χ, it has the following constructors:

$$\begin{array}{lll} \texttt{type} & \diamond & \chi \\ \texttt{type} & \dot{\vee} & \chi \to \chi \to \chi \\ \texttt{type} & \dot{\wedge} & \chi \to \chi \to \chi \\ \texttt{type} & \dot{\forall} & (\kappa \times \chi) \to \chi \\ \texttt{type} & \dot{\exists} & \texttt{list}\ (\varrho \times \chi) \to \chi \end{array}$$

where: κ is the type of selection variables and ϱ is the type of expansion terms.

Polarity

All ambiguous connectives are given negative polarity and atoms can be arbitrarily and globally polarized. Similar to the polarity assignment for the conjunctive normal form decision procedure seen in figure 5.1:

$$[\![B \circ C]\!]^{\pm} = [\![B]\!]^{\pm} \circ^{-} [\![C]\!]^{\pm} \quad [\![a]\!]^{\pm} = a^{+} \quad [\![\neg a]\!]^{\pm} = a^{-}$$

where $\circ \in \{\vee, \wedge\}$ and a is atomic.

In the expansion sequent calculus, all existential formulas are potentially subject to contraction *separately* which is not the case in *LKF*. To understand the difference, consider the following formula:

$$\exists x_1.\exists x_2. \cdots \exists x_n.\exists y.F \tag{5.7}$$

where F is some formula containing x and y, with expansion tree:

$$\dot{\exists} +^{a_1} (\dot{\exists} +^{a_2} \cdots (\dot{\exists} +^{a_n} (\dot{\exists} +^{c} \tau_1 +^{b} \tau_2)) \cdots)$$

where τ_1 and τ_2 are some expansion trees.

In expansion sequent calculus, the formula 5.7 will be instantiated once with a_1, as will all its existential subformulas $\exists x_2. \cdots \exists x_n.\exists y.F$ with a_2, $\exists x_3. \cdots \exists x_n.\exists y.F$ with a_3, and so forth until $\exists y.F$, which will be instantiated twice, once with b and once with c. However, in the aggressively focused *LKF* where focused phases are maximal, when deciding on the formula 5.7 the focused phase will proceed with the subformulas without releasing. This means that the whole chain of instantiation with the terms a_i will be repeated twice, the first time ending with the instantiation of y

with b, and the second time ending with the instantiation of y with c. This requires that all the expansion terms a_i be stored and maintained while, in the expansion sequent calculus, they were discarded after use by *instantiate* rule.

A better embedding is thus to use a negative delay $\partial^-(\cdot)$ after each existential formula, prompting a release after instantiation in an *LKF* derivation and subsequent storing of the existential subformula.

$$[\![\exists x.B]\!]^{\pm} = \exists x.\partial^-([\![B]\!]^{\pm})$$

5.4.3 Indexing

As a result of the negative polarity assignment to all ambiguous connectives, the only formulas to be stored are existential formulas and atoms. The proof certificate holds no information on the pairing of complementary atoms (as was the case for mating). The atoms can, therefore, be given dummy index:

$$\texttt{type}\ \ \texttt{lit}\ \ \xi.$$

The existential formulas, as seen in the $\dot{\exists}$ figure 5.14, are stored with a fresh index. To provide this fresh index, the store clerk relies on a counter initialized at 0 and incremented after each storage of an existential formula. The constructor for these indexes is:

$$\texttt{type}\ \ \texttt{id}\ \ \mathbb{N} \rightarrow \xi.$$

whith $\mathbb{N}$ the type of integers.

5.4.4 Region delimitation

The sequentialization can be separated into two main processes, closely related to tree traversals.

The first process starts at a node (initially, the root) and traverses the $\dot{\vee}$, $\dot{\wedge}$ and $\dot{\forall}$ nodes of the tree from father to son (these are the invertible rules). For the $\dot{\forall}$ node, the selection variable introduced at the level of the expansion tree is paired with the eigenvariable generated at the level of sequent calculus proof, and this pair is added to the substitutions list σ. When $\dot{\exists}$ nodes are reached, the traversals halts at these nodes and the out-going branches of these nodes are added to a set of triples in Ψ from which the traversal can restart. When a leaf is reached, the traversal of that branch also halts.

The second process takes place when all the branches have been traversed. Then if there are still nodes in Ψ, the process picks a branch with a minimal expansion

term, removes it from Ψ, and repeats process one. If Ψ is not empty but none of its expansion terms are minimal, the expansion tree is false. If Ψ is empty then a pair of complementary atoms should be present in the storage: finish with an initial rule.

Regarding the cases, the pieces of information that need to be maintained are the set of substitutions σ and the set of existential branches Ψ. These must be globally accessible: σ is needed to make a minimal expansion term into a sequent-level term to serve as a witness for existential formulas; the branches in Ψ are supplied by the store clerk and used by the decide expert to start sequentialization at unlocked branches.

The main case, named `exp`, has the following type:

$$\texttt{type}\quad\texttt{exp}\quad \mathbb{N} \to \texttt{list}\ (\iota \times \kappa) \to \texttt{list}\ (\varrho \times \chi \times \mathbb{N}) \to \texttt{list}\ \chi$$

where the first argument, an integer, is the counter used to provide fresh indexes and the rest of arguments mirror the elements on the left-hand side of an expansion sequent. In other words:

- the list of substitutions presented as pairs of ι, the type of sequent level terms, and κ, the type of selection variables
- the list of expansion branches which are triples of, respectively, an expansion term (of type ϱ), an expansion tree (of type χ) and the integer corresponding to the index of the existential formula
- the list of expansion branches and the current nodes being traversed, in one-to-one correspondence with the formula in the workbench

5.4.5 Clerks and experts

The end-sequent of a derivation guided by an expansion tree's proof certificate is thus:

$$\texttt{exp}\ 0\,[]\,[]\,[\tau] \vdash \cdot \Uparrow F$$

where τ is the tree associated with formula F.

The negative conjunction $\wedge_c^-$ and disjunction $\vee_c^-$ clerks simulates what happens on the left-hand side of the expanded sequent in the $\dot\wedge$ and $\dot\vee$ rules seen in figure 5.14:

$$\wedge_c^-(\texttt{exp}\ \ c\,\sigma\,\Psi\,[\tau_1 \mathbin{\dot\wedge} \tau_2|\Phi], \texttt{exp}\ \ c\,\sigma\,\Psi\,[\tau_1|\Phi], \texttt{exp}\ \ c\,\sigma\,\Psi\,[\tau_2|\Phi])$$

$$\vee_c^-(\texttt{exp}\ \ c\,\sigma\,\Psi\,[\tau_1 \mathbin{\dot\vee} \tau_2|\Phi], \texttt{exp}\ \ c\,\sigma\,\Psi\,[\tau_1, \tau_2|\Phi])$$

The universal clerk $\forall_c$ must add to σ a link between the fresh eigenvariable introduced by the kernel (*LKF*) at the $\forall$ rule and the selection variable known to the expansion tree and introduced by the $\dot{\forall}$ node. This maneuver is necessary because the selection variable in an expansion tree is known to the whole expansion tree, which lays inside the `exp` case, *i.e.,*it is *not fresh* in the sequent proof. Thus a fresh variable is mapped to each selection variable and is used, at the sequent level, whenever that selection variable is used.

$$\forall_c(\texttt{exp}\ \ c\,\sigma\,\Psi\,[\dot{\forall} +^u \tau|\Phi], \lambda x.\texttt{exp}\ \ c\,([x/u] :: \sigma)\,\Psi\,[\tau|\Phi])$$

To see how these clerks simulate the rules seen in figure 5.14, one has to consider the entire augmented rule, *e.g.,* the augmented $\forall$ rule of LKF^a is the following:

$$\frac{\Xi'\,y \vdash \Gamma \Uparrow B\,y \quad \forall_c(\Xi, \Xi')}{\Xi \vdash \Gamma \Uparrow \forall B}$$

where $\Xi = \texttt{exp}\ \ c\,\sigma\,\Psi\,[\dot{\forall} +^u \tau|\Phi]$ and $\Xi' = \lambda x.\texttt{exp}\ \ c\,([x/u] :: \sigma)\,\Psi\,[\tau|\Phi]$. By applying the same fresh eigenvariable to both the formula under a universal connective ($B\,y$) and the case abstracted over a variable ($\Xi'\,y = \texttt{exp}\ \ c\,([y/u] :: \sigma)\,\Psi\,[\tau|\Phi]$), the kernel and the continuation case (and through this case, all the agents defined on it) will share variables.

The store clerk $\mathscr{S}_c$ behaves differently according to the head tree of Φ. When it is a leaf, the formula should be an atom and is stored with a dummy index `lit`.

$$\mathscr{S}_c(\texttt{exp}\ \ c\,\sigma\,\Psi\,[\diamond|\Phi], _, \texttt{exp}\ \ c\,\sigma\,\Psi\,\Phi, \texttt{lit})$$

If the node at the head of the list of trees is an existential node $\dot{\exists}$, the store clerk assumes the head formula of the workbench is an existential formula. Using the counter c, the store clerk generates a fresh index (`id` c) and increments the counter. This clerk also adds all the outgoing branches to the list Ψ along with the index c given to their relative existential formula. To avoid obfuscation, this can be done through the use of an auxiliary predicate `expand` adding the expansion branches list EB to Ψ to get Ψ'.

$$\mathscr{S}_c(\texttt{exp}\ \ c\,\sigma\,\Psi\,[\dot{\exists}EB|\Phi], \exists_, \texttt{exp}\ \ (c+1)\,\sigma\,\Psi'\,\Phi, \texttt{id}\ \ c)\ \ \texttt{:-}\ \ \texttt{expand}\,\Psi\,c\,EB\,\Psi'.$$

where `expand` is declared with this type:

$$\texttt{type}\ \ \texttt{expand}\ \ \texttt{list}\ \ (\varrho \times \chi \times \mathbb{N}) \to \mathbb{N} \to \texttt{list}\ \ (\varrho \times \chi) \to \texttt{list}\ \ (\varrho \times \chi \times \mathbb{N}) \to o$$

and defined with the following horn clauses:

$$\texttt{expand}\ \Psi\ _\ [\,]\ \Psi.$$
$$\texttt{expand}\ \Psi\ c\ [+^t\tau|EB][\langle t,\tau,\texttt{id}\ \ c\rangle|\Psi']\ \ \texttt{:-}\ \ \texttt{expand}\ \Psi\ c\ EB\ \Psi'.$$

The *instantiate* rule in figure 5.14 is the combination of two rules: a contraction and an instantiation of existential. Therefore, it is mapped in LKF^a to a decide rule followed by an existential rule. At the end of the invertible phase, the decide expert $\mathscr{D}_e$ succeeds with a member $\langle T, \tau, I\rangle$ of Ψ and removes it from Ψ. This expert gives to the kernel the index I of the corresponding (previously stored) existential formula and generates the term $T\sigma$, if the term T is minimal, this substitution yields a sequent-level term (with no selection variables). If it is not minimal, it contains a selection variable not yet introduced, *i.e.,*if the expansion branch $\langle T, \tau, I\rangle$ is *locked* (see definition 5.4.6) and thus the term $T\sigma$ is not usable as a sequent-level term. If substitution fails to produce any sequent-level term out of all members of Ψ, the dependency relation is cyclic and the proof checking halts. If the substitution succeeds, the resulting term $T\sigma$ is encapsulated in the default case `witness` that can be seen as placing a post-it note on the `exp` case (see section 4.3.3 for the existential expert defined on this case).

$$\mathscr{D}_e(\texttt{exp}\ \ c\,\sigma\,(\{\langle T,\tau,I\rangle\}\uplus\Psi)\,[\,],\texttt{witness}\ \ (T\sigma)\,(\texttt{exp}c\,\sigma\,\Psi\,[\tau]),I)$$

Chapter 6

FPC for intuitionistic logic

This chapter offers a series of case studies for various semantics definitions in intuitionistic logic, starting with simple formats and moving to more elaborated ones. While the number of rules in the (two-sided) LJF^a system is twice that of the rules in the (one-sided) LKF^a system, communication between proof certificate and kernel follows the same mechanics. Invertible phases are populated with clerks, focused phases are populated with experts, formulas in the left storage are paired with an index, cases serve as the main of communication between agents, etc.

6.1 Mimic

This section is intended as an easy first step in understanding the Foundational Proof Certification framework in the case of intuitionistic logic. This first section, like the first section of the previous chapter, shows the elaboration of a `p.r.i.c.e.` for a decision procedure. In particular, a decision procedure that succeeds if two formulas are equal. A similar `p.r.i.c.e.` can be defined in LKF^a for classical logic.

6.1.1 Initial in LJF^a

As seen in figure 3.3, initial rules of LJF^a are restricted to atomic formulas. As a result, when the derivation reaches, for instance, a sequent $\cdot \Uparrow F \vdash F \Uparrow \cdot$ where F is a potentially large formula, the derivation must continue decomposing F on either side of the turnstile until reaching the atoms. While one might see this restriction to atomic initial as a weakness of focusing and believe there could be advantages in allowing the (perfectly admissible) non-atomic initial, adding this rule is not an optimization for the following reasons:

1. there is only one sequent that is the conclusion of more than one rule in (the otherwise syntax directed) LJF^a: $\Gamma \Uparrow \cdot \vdash \cdot \Uparrow R$ can be both the conclusion of a cut rule and of one of the decide rules. Adding a non-atomic initial rule will make every sequent the potential conclusion of an initial rule. With proof search as an objective, this behavior is far from ideal and a related disadvantage of this is the discontinuation of the communication protocol between proof certificate and kernel,

2. to check applicability of initial rule, the entire sequent is searched and an equality test is done on each of its formulas,

3. on the one hand, testing equality of formulas is done through a (direct) traversal of their syntax tree: if the head connectives are the same, check equality of the respective subformulas. On the other hand, applying introduction rules on the connectives of a formula is, in effect, also a traversal of its syntax tree: introduce the head connective, then continue with the subformulas. If the process of applying the introduction rules on a formula is given enough guidance, the ensuing derivation can be made to succeed on equal formulas with a similar cost as the direct traversal required for testing equality.

The next sections investigate how to guide the kernel to build the proof of equal formulas.

6.1.2 Negative/positive alternation

The sequent $\cdot \Uparrow F \vdash F \Uparrow \cdot$ appears in an invertible phase. Depending on the polarity of F, one of its two occurrences is stored and called the *mirror* F. In particular, if F is negative (resp. positive) the left (resp. right) occurrence is stored while the other occurrence, belonging in the invertible phase, is decomposed, *i.e.,*its top-level connective is introduced. Of the resulting subformulas of F, those also belonging to the invertible phase are decomposed, then their invertible-belonging subformulas, and so on until reaching non invertible-belonging formulas (positive on the right, negative on the left, or atoms) which are then stored.

A focused phase is engaged by a decide on the mirror F. This focused phase will introduce the same connectives of the mirror formula as those connectives of the other occurrence of F introduced during the previous invertible phase. The focused phase ends at the same subformulas as the previous invertible phase did. Of those subformulas, some may be introduced by an initial (if focused on atoms that belong in the focused phase -negative on the left, positive on the right), others are

released, causing another alternation of invertible and focused phases similar to the one described above.

The pattern appears to be that the invertible phase (starting with the end-sequent or with a release) is the challenge phase where player A makes some steps and then gives the hand to the opponent B and dares him to do the same steps (introduce the same connectives). Player B then decides on the head of the (sub)formula that the invertible phase of A introduced and engages in a focused phase that introduces the same connectives. Figure 6.2 shows this alternation.

Example 6.1.1. *As a further illustration, consider the formula $F = ((a \wedge^- b) \vee c) \supset (d \wedge^+ e)$ where a, b, d are negative atoms and c, e are positive atoms. Figure 6.1 shows the derivation of the sequent $\cdot \Uparrow F \vdash F \Uparrow \cdot$. Player A starts the derivation with an invertible phase on F on the right-hand side. After decomposing the part of F that belongs to the invertible phase ($\supset$ and $\vee$), it ends with the following premises:*

$$a \wedge^- b, F \Uparrow \cdot \vdash \cdot \Uparrow d \wedge^+ e$$

and

$$c, F \Uparrow \cdot \vdash \cdot \Uparrow d \wedge^+ e$$

In both of these premises, player B is challenged to mimic the steps player A just took. The double horizontal line in figure 6.1 represents the change in players. Player B starts with a decide rule on F, and the subsequent focused phases decompose the same connectives as the previous invertible phase, i.e.,*$\supset$ and $\vee$, then either end with an initial if an atom is reached (as is the case for c) and player B successfully accepts that branch of F, or release the focus after having mimicked player A up to the following premises:*

$$c, F \Uparrow d \wedge^+ e \vdash \cdot \Uparrow d \wedge^+ e$$
$$a \wedge^- b, F \Uparrow d \wedge^+ e \vdash \cdot \Uparrow d \wedge^+ e$$
$$a \wedge^- b, F \Uparrow \cdot \vdash a \wedge^- b \Uparrow \cdot$$

then, player B decompose deeper into F the $\wedge^+$ and $\wedge^-$ connectives. After these invertible phases, it is time (notice the double horizontal line in figure 6.1) for player A to engage in a focused phase and mimic player B: the derivations starting with the first two of the three above sequents will decide on the right and focus on formula $d \wedge^+ e$, while the derivation starting with the third sequent will decide on the left-hand side and focus on formula $a \wedge^- b$, and the alternation continues. $\odot$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a \wedge^- b, F \Downarrow a \vdash a}\ \mathbb{I}^l}{a \wedge^- b, F \Downarrow a \wedge^- b \vdash a}\ \wedge^-_l}{a \wedge^- b, F \Uparrow \cdot \vdash \cdot \Uparrow a}\ \mathbb{D}^l}{a \wedge^- b, F \Uparrow \cdot \vdash a \Uparrow \cdot}\ \mathbb{S}^r \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{a \wedge^- b, F \Downarrow b \vdash b}\ \mathbb{I}^l}{a \wedge^- b, F \Downarrow a \wedge^- b \vdash b}\ \wedge^-_l}{a \wedge^- b, F \Uparrow \cdot \vdash \cdot \Uparrow b}\ \mathbb{D}^l}{a \wedge^- b, F \Uparrow \cdot \vdash b \Uparrow \cdot}\ \mathbb{S}^r}{a \wedge^- b, F \Uparrow \cdot \vdash a \wedge^- b \Uparrow \cdot}\ \wedge^-_r}{a \wedge^- b, F \vdash a \wedge^- b \Downarrow}\ \mathbb{R}^r}{a \wedge^- b, F \vdash (a \wedge^- b) \vee c \Downarrow}\ \vee_r \qquad \pi(a \wedge^- b)}{a \wedge^- b, F \Downarrow F \vdash d \wedge^+ e}\ \supset_l}{a \wedge^- b, F \Uparrow \cdot \vdash \cdot \Uparrow d \wedge^+ e}\ \mathbb{D}^l}{a \wedge^- b, F \Uparrow \cdot \vdash d \wedge^+ e \Uparrow \cdot}\ \mathbb{S}^r}{F \Uparrow a \wedge^- b \vdash d \wedge^+ e \Uparrow \cdot}\ \mathbb{S}^l \qquad \pi'}{F \Uparrow (a \wedge^- b) \vee c \vdash d \wedge^+ e \Uparrow \cdot}\ \vee_l}{F \Uparrow \cdot \vdash F \Uparrow \cdot}\ \supset_r}{\cdot \Uparrow F \vdash F \Uparrow \cdot}\ \mathbb{S}^l
$$

π':

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{c, F \vdash c \Downarrow}\ \mathbb{I}^r}{c, F \vdash (a \wedge^- b) \vee c \Downarrow}\ \vee_r \qquad \pi(c)}{c, F \Downarrow F \vdash d \wedge^+ e}\ \supset_l}{c, F \Uparrow \cdot \vdash \cdot \Uparrow d \wedge^+ e}\ \mathbb{D}^l}{c, F \Uparrow \cdot \vdash d \wedge^+ e \Uparrow \cdot}\ \mathbb{S}^r}{F \Uparrow c \vdash d \wedge^+ e \Uparrow \cdot}\ \mathbb{S}^l
$$

$\pi(\Box)$:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{e, d, \Box, F \Downarrow d \vdash d}\ \mathbb{I}^r}{e, d, \Box, F \Uparrow \cdot \vdash \cdot \Uparrow d}\ \mathbb{D}^l}{e, d, \Box, F \Uparrow \cdot \vdash d \Uparrow \cdot}\ \mathbb{S}^r}{e, d, \Box, F \vdash d \Downarrow}\ \mathbb{R}^r \qquad \dfrac{}{e, d, \Box, F \vdash e \Downarrow}\ \mathbb{I}^r}{e, d, \Box, F \vdash d \wedge^+ e \Downarrow}\ \wedge^+_r}{e, d, \Box, F \Uparrow \cdot \vdash \cdot \Uparrow d \wedge^+ e}\ \mathbb{D}^r}{d, \Box, F \Uparrow e \vdash \cdot \Uparrow d \wedge^+ e}\ \mathbb{S}^l}{\Box, F \Uparrow d, e \vdash \cdot \Uparrow d \wedge^+ e}\ \mathbb{S}^l}{\Box, F \Uparrow d \wedge^+ e \vdash \cdot \Uparrow d \wedge^+ e}\ \wedge^+_l}{\Box, F \Downarrow d \wedge^+ e \vdash d \wedge^+ e}\ \mathbb{R}^l
$$

Figure 6.1: Derivation of example 6.1.1

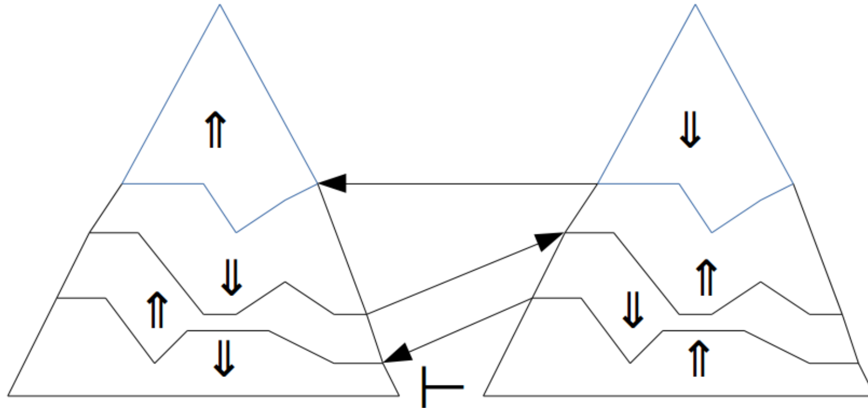


Figure 6.2: Alternation of phases according to polarity and side of the sequent

6.1.3 A p.r.i.c.e. for mimic

This decision procedure is set inside of *LJF*, *i.e.*,on polarized formulas, thus there is no polarity assignment to discuss. The formulas are stored using their path (see definition 5.2.1). The derivations are separated into the challenging region, containing the `dare` case having this declaration:

$$\texttt{type}\ \ \texttt{dare}\ \ \xi \to \texttt{list}\ \xi \to \xi \to \texttt{case}$$

where the first argument is the index of the formula on which to do the next decide rule, the second argument is a list of indexes in one-to-one correspondence with the left-hand side workbench and the third argument is the index associated with the formula on the right-hand side of the turnstile (be it stored or not).

The second region is the mimicking region. The sequents focused on the left are augmented with the `mimL` case:

$$\texttt{type}\ \ \texttt{mimL}\ \ \xi \to \xi \to \texttt{case}$$

where the first argument is the index of the formula under the left focus while the second argument is the index of the formula on the right-hand side of the turnstile. The sequents focused on the right are augmented with the `mimR` case:

$$\texttt{mimR}\ \ : \xi \to \texttt{case}$$

where the argument is the index of the formula on the right-hand side of the turnstile.

The challenging phase is inhabited by clerks. Those clerks relative to a connective simply maintain the correspondence between the paths in the second and third arguments of the `dare` case and the formulas in the workbenches.

$$\supset_c (\texttt{dare}\ I\,[]\,J, \texttt{dare}\ I\,[\prec J]\,(\succ J))$$
$$\wedge_c^- (\texttt{dare}\ I\,[]\,J, \texttt{dare}\ I\,[]\,(\prec J), \texttt{dare}\ I\,[]\,(\succ J))$$
$$\wedge_c^+ (\texttt{dare}\ I\,[H|R]\,J, \texttt{dare}\ I\,[\prec H, \succ H|R]\,J)$$
$$\vee_c (\texttt{dare}\ I\,[H|R]\,J, \texttt{dare}\ I\,[\prec H|R]\,J, \texttt{dare}\ I\,[\succ H|R]\,J)$$

The store clerks ensure that formulas are stored with their corresponding path:

$$\mathbb{S}_c^r(\texttt{dare}\ I\,[]\,J, _, \texttt{dare}\ I\,[\,]J)$$
$$\mathbb{S}_c^l(\texttt{dare}\ I\,[H|R]\,J, _, H, \texttt{dare}\ I\,R\,J)$$

The mimicking phase is inhabited by experts. The first expert called, the decide expert, starts the mimicking on the formula under the first argument of the `dare` case.

$$\mathbb{D}_e^r(\texttt{dare}\ I\,[]\,I, \texttt{mimR}\ I)$$
$$\mathbb{D}_e^l(\texttt{dare}\ I\,[]\,J, I, \texttt{mimL}\ I\,J)$$

The paths are also maintained during this phase, be it while introducing connectives on the left:

$$\wedge_e^- (\texttt{mimL}\ I\,J, \texttt{left}, \texttt{mimL}\ (\prec I)\,J)$$
$$\wedge_e^- (\texttt{mimL}\ I\,J, \texttt{right}, \texttt{mimL}\ (\succ I)\,J)$$
$$\supset_e (\texttt{mimL}\ I\,J, \texttt{mimR}\ (\prec I), \texttt{mimL}\ (\succ I)\,J)$$

or on the right:

$$\vee_e (\texttt{mimR}\ I, \texttt{left}, \texttt{mimR}\ (\prec I))$$
$$\vee_e (\texttt{mimR}\ I, \texttt{right}, \texttt{mimR}\ (\succ I))$$
$$\wedge_e^+ (\texttt{mimR}\ I, \texttt{mimR}\ (\prec I), \texttt{mimR}\ (\succ I))$$

and the initial expert relies on this information to allow ending the proof on certain indexes and not others:

$$\mathbb{I}^l_e(\texttt{mimL}\ \ I\ I)$$
$$\mathbb{I}^r_e(\texttt{mimR}\ \ I, I)$$

The release expert is the one that sets the new first argument of the `dare` case for the following mimicking phase:

$$\mathbb{R}^l_e(\texttt{mimL}\ \ I\ J, \texttt{dare}\ \ I\ [I]\ J)$$
$$\mathbb{R}^r_e(\texttt{mimR}\ \ I, \texttt{dare}\ \ I\ []\ I)$$

6.1.4 Extending to first order

Support for first order formulas is simply a matter of adding the agents and expanding the notion of path. Now, in addition of the path constructors introduced in definition 5.2.1, if ρ is a path then $(t \circledast \rho)$ is also a path, where t is a term, or more formally:

$$\texttt{type}\quad .\circledast.\quad \iota \to \xi \to \xi.$$

where the notation $.\circ.$ means that the operator $\circ$ is infix.

The $\exists_c$ and $\forall_c$ have similar behaviors:

$$\exists_c(\texttt{dare}\ \ I\ [H|R]\ J, \lambda y.\texttt{dare}\ \ I\ [(y \circledast H)|R]\ J)$$
$$\forall_c(\texttt{dare}\ \ I\ []\ J, \lambda y.\texttt{dare}\ \ I\ []\ (y \circledast J))$$

and the $\exists_e$ and $\forall_e$ have similar behaviors:

$$\exists_e(\texttt{mimR}\ \ I, \texttt{mimR}\ \ (X \circledast I), X)$$
$$\forall_e(\texttt{mimL}\ \ I\ J, \texttt{mimL}\ \ (X \circledast I)\ J, X)$$

where X is a logic variable. At the initial rule, the initial experts check if the paths of the complementary atoms match. This ensures that logic variables introduced for a connective only unify with their corresponding eigenvariables introduced at the same connective (same position) in the mirror formula.

Remark 6.1.1. Note that the index of the root formula, noted $\boxtimes$ in the mimic section 6.1, does not have to be $\boxtimes$, it can be any element of type ξ. This allows mimic p.r.i.c.e. to be used as an add-on to another p.r.i.c.e. to check non atomic initial.

6.2 λ-calculus

The untyped λ-calculus is a formal system for expressing computation introduced in [Church, 1936]. A few years later, the simply-typed version of λ-calculus was introduced [Church, 1940] and achieved Church's goal of having a formal logic system [Benzmüller and Miller, 2014]. Later on, simply-typed λ-calculus was found to be equivalent to intuitionistic natural deduction through what is called the proofs-as-programs and formulas-as-types interpretation. A typed λ-calculus term can thus be taken as the intuitionistic natural deduction proof of its type. This section relates to this correspondence.

Untyped λ-calculus

Terms in the λ-calculus, or λ-terms, are syntactically defined as

$$T, U ::= \ x \mid T\,U \mid \lambda x.\,T$$

respectively designating variables, function applications and function abstractions. The set $FV(t)$ of *free* variables of a λ-term t is defined as

$$FV(x) = \{x\} \quad FV(t\,u) = FV(t) \cup FV(u) \quad FV(\lambda x.t) = FV(t) \setminus \{x\}$$

A variable that is not free is *bound* by the abstraction operator λ that is closest to it, *e.g.,* in the term $\lambda x.\lambda x.x$, the variable x is bound by the second λ operator. Function application associates to the left, *i.e.,*the term $t\,u\,z$ is the same as the term $(t\,u)\,z$ but not the same as the term $t\,(u\,z)$ The λ operator binds as far right as possible, *i.e.,*the term $\lambda x.t\,u$ is understood as $\lambda x.(t\,u)$ not as $(\lambda x.t)\,u$.

An α-conversion of a λ-term is the renaming of its bound variables without changing the meaning of the λ-term itself, *e.g.,* the term $\lambda x.y\,x$ can be α-converted to the term $\lambda z.y\,z$ but not to the term $\lambda y.y\,y$ as it is no longer the same term.

A β-redex is a term of the form $(\lambda x.t)\,u$. A term is in β-normal form if it contains no subterm which is a β-redex.

Simply-typed λ-calculus

Simple types are built using the following grammar:

$$\tau, \sigma ::= \iota \mid \tau \to \sigma$$

denoting the base type and arrow types respectively. An arrow-typed variable x of type $\iota_1 \to ... \to \iota_n \to \iota$ is said to be *fully applied* in a term t if all the occurrences of

x are applied to n arguments of the right type $u_i : \iota_i$. The typing relation $\Gamma \Vdash t : \sigma$ is valid if the λ-term t has type σ in context Γ. The rules for this relation are the following:

$$\frac{\Gamma, x : \tau \Vdash t : \sigma}{\Gamma \Vdash \lambda x.t : \tau \to \sigma}\ abs \qquad \frac{\Gamma \Vdash u : \tau \to \sigma \quad \Gamma \Vdash t : \tau}{\Gamma \Vdash u\, t : \sigma}\ app \qquad \frac{}{x : \tau, \Gamma \Vdash x : \tau}\ var$$

As said above, there is a correspondence with natural deduction. In particular, the rules *abs, app, var* correspond, respectively, to implication introduction, implication elimination and initial rule:

$$\frac{\Gamma, B \vdash C}{\Gamma \vdash B \supset C} \supset_i \qquad \frac{\Gamma \vdash B \supset C \quad \Gamma \vdash B}{\Gamma \vdash C} \supset_e \qquad \frac{}{B, \Gamma \vdash B}\ init$$

A λ-term t of type $\tau \to \sigma$ can be η-expanded into the term $\lambda x.t\, x$ of the same type. A λ-term t is said to be in η-long normal form if all occurrences of all its arrow-typed variables are fully applied.

Example 6.2.1. *The typed term:*

$$\lambda x \lambda y.y\, x : (a \to b) \to ((a \to b) \to (c \to d) \to e) \to (c \to d) \to e$$

is not in η-long-normal form. Its η-long-normal form is the term

$$\lambda x \lambda y \lambda t.y\ (\lambda z.x\, z)(\lambda h.t\, h)$$

$\odot$

Notations

De Bruijn indices [de Bruijn, 1972] are a notation for λ-calculus where variable names are replaced by integers such that α-convertible named terms have syntactically equal de Bruijn notations.

A λ-term in de Bruijn notation is generated from the following grammar:

$$M, N ::= n \mid M\ N \mid \lambda\, M$$

where each variable is replaced by the number of binders between its binder and its occurrence. In addition to de Bruijn indices, terms are written in *spine notation* [Herbelin, 1995a] where a variable is applied to a list of terms. If the term is in β-normal form, the λ symbol is omitted. A β-normal form term written using de Bruijn indices where the λ symbol is omitted is said to be in *light de Bruijn* notation.

$$\frac{c+1|t \Vdash \Gamma, B_c \vdash D}{c|t \Vdash \Gamma \vdash B \supset D} \supset_r$$

$$\frac{c|t^1 \Vdash \Gamma \vdash B^1 \quad \cdots \quad c|t^n \Vdash \Gamma \vdash B^n \quad (B^1 \supset \cdots \supset B^n \supset D)_{(c-i-1)} \in \Gamma}{c|i[t^1, \cdots, t^n] \Vdash \Gamma \vdash D} \supset_l$$

$$\frac{(c-i-1 : B) \in \Gamma}{c|i \Vdash \Gamma \vdash B} \ init$$

Figure 6.3: Modified sequent calculus for checking guided by λ-terms proof evidence

Example 6.2.2. *$\lambda x.\lambda t.\lambda y.x\, t\, (\lambda z.z\, y)$ is written in de Bruijn notation as $\lambda\lambda\lambda.2\, 1\, (\lambda.0\, 1)$. In spine notation, the same term becomes $\lambda\lambda\lambda.2\, [1, (\lambda.0\, [1])]$. Since it is in β-normal form, its light de Bruijn notation is $2\, [1, 0\, [1]]$.* $\odot$

Remark 6.2.1. A light de Bruijn notation can be mapped to more than one lambda term. For instance, the term $1\, [0\, [1]]$ is the light de Bruijn notation of both the term $\lambda x.\lambda y.x\, (\lambda z.z\, y)$ and the term $\lambda x.\lambda y.x\, (y\, x)$.

Using η-long β-normal form λ-terms as proof evidence

Using an η-long β-normal form λ-term s as proof evidence for checking a formula B is *not* the same as checking that the λ-term s has type B. If $s \Vdash \cdot \vdash B$ is taken as the notation for: "the term s provides evidence that the sequent $\cdot \vdash B$ is provable", then the only implication needed is:

$$\cdot \Vdash s : B \quad \Rightarrow \quad s \Vdash \cdot \vdash B$$

A modified intuitionistic sequent calculus is given in figure 6.3. The sequents of this calculus are of the form:

$$c|t \Vdash \Gamma \vdash D$$

where c is an integer counter, t is a λ-term, Γ is a multiset of indexed formulas and D is a formula. The end sequent of derivations in this sequent calculus is:

$$0|t \Vdash \cdot \vdash D$$

that is to say, the counter is initialized to 0. In the $\supset_l$ rule, an implicit sequent is the conclusion of an initial rule on the (necessarily atomic) formula D.

To illustrate the mechanism of this calculus, the derivation of the formula

$$(b \supset ((c \supset d) \supset d) \supset e) \supset b \supset c \supset e$$

$$
\dfrac{\dfrac{b_{(3-1-1)} \in \Gamma}{3|1 \Vdash \Gamma \vdash b}\,init \qquad \dfrac{\dfrac{\dfrac{\dfrac{c_{(4-1-1)} \in \Gamma}{4|1 \Vdash \Gamma, (c \supset d)_3 \vdash c}\,init}{4|0\,[1] \Vdash \Gamma, (c \supset d)_3 \vdash d}\supset_l}{3|0\,[1] \Vdash \Gamma \vdash (c \supset d) \supset d}\supset_r}{}}{\dfrac{3|2\,[1,0\,[1]] \Vdash \Gamma \vdash e}{\dfrac{2|2\,[1,0\,[1]] \Vdash (b \supset ((c \supset d) \supset d) \supset e)_0, b_1 \vdash c \supset e}{\dfrac{1|2\,[1,0\,[1]] \Vdash (b \supset ((c \supset d) \supset d) \supset e)_0 \vdash b \supset c \supset e}{0|2\,[1,0\,[1]] \Vdash \cdot \vdash (b \supset ((c \supset d) \supset d) \supset e) \supset b \supset c \supset e}\supset_r}\supset_r}\supset_r}\supset_l
$$

$$
\Gamma = (b \supset ((c \supset d) \supset d) \supset e)_0, b_1, c_2
$$

Figure 6.4: Example derivation for example 6.2.2

guided by the η-long β-normal form term $2\,[1, 0\,[1]]$ from example 6.2.2 is given in figure 6.4. The proof certificates defined in the sections below are inspired from the simple sequent calculus in figure 6.3.

6.2.1 p.r.i.c.e. for simply-typed η-long β-normal form λ-terms

The only polarity assignment to choose is that of atoms since the only other formula constructor is the implication $\supset$, which is inherently negative (see section 2.2.1). Atoms are also given a negative polarity.

The extra information in the modified sequent calculus in figure 6.3 is the λ-term and the counter. A direct way of maintaining this information is the following case constructor:

$$\texttt{type}\ \ \texttt{lam}\quad \mathbb{N} \to \lambda_t \to \texttt{case}.$$

where λ_t is the type of the inner representation of λ-terms in light de Bruijn notation. Constructors for λ_t are the following:

`type`	`infix`	@	$\lambda_t \to \texttt{list}\ \lambda_t \to \lambda_t$.
`type`	v		$\mathbb{N} \to \lambda_t$.

but for convenience, the (more natural) notation $i\,[u_1, \cdots, u_n]$ and the notation $(v\ i)@[u_1, \cdots, u_n]$ are used interchangeably.

The rules of the system in figure 6.3 can be simulated with sequences of LJF^a rules. For example, the $\supset_r$ is simulated through:

$$\dfrac{\dfrac{\overline{\texttt{lam}\ (c+1)\,t\colon \Gamma, B_c \Uparrow \cdot \vdash D \Uparrow \cdot}}{\texttt{lam}\ c\,t\colon \Gamma \Uparrow B \vdash D \Uparrow \cdot}}{\texttt{lam}\ c\,t\colon \Gamma \Uparrow \cdot \vdash B \supset D \Uparrow \cdot}$$

while the $\supset_l$ rule is simulated through:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{\texttt{lam}\ c\,t^1\colon \Gamma \Uparrow \cdot \vdash B^1 \Uparrow \cdot}}{\Gamma \vdash B^1 \Downarrow}\ \mathbb{R}^r\star \quad \dfrac{\vdots\ \dfrac{\dfrac{\overline{\texttt{lam}\ c\,t^n\colon \Gamma \Uparrow \cdot \vdash B^n \Uparrow \cdot}}{\Gamma \vdash B^n \Downarrow}\ \mathbb{R}^r\star \quad \dfrac{}{\Gamma \Downarrow d \vdash d}\ init}{\Gamma \Downarrow B^n \supset d \vdash d}\ \supset_l}{\Gamma \Downarrow B^2 \supset \cdots \supset B^n \supset d \vdash d}}{\Gamma \Downarrow B^1 \supset \cdots \supset B^n \supset d \vdash d}\ \supset_l}{\texttt{lam}\ c\,i[t^1, \cdots, t^n]\colon \Gamma \Uparrow \cdot \vdash \cdot \Uparrow d}\ \mathbb{D}^l}{\texttt{lam}\ c\,i[t^1, \cdots, t^n]\colon \Gamma \Uparrow \cdot \vdash d \Uparrow \cdot}\ \mathbb{S}^r$$

where $(B^1 \supset \cdots \supset B^n \supset d)_{(c-i-1)} \in \Gamma$ and d is an atomic formula. By construction, variables are always fully applied. In $\star$, right release rule is the only applicable rule because the formulas can only have negative atoms and implications.

The omission of case augmenting parts of this derivation (namely the focused phase) is to clarify the simulation of the $\supset_l$ rule in the calculus in figure 6.3 using LJF^a derivations. To complete the augmentation of the rest of the derivation, the following case constructor maintains the argument list so that the $\supset_e$ matches each argument with the corresponding `lam` case:

$$\texttt{type}\quad \texttt{arg}\quad \mathbb{N} \to \texttt{list}\ \lambda_t \to \texttt{case}.$$

The indexes are generated through a simple constructor:

$$\texttt{type}\quad \texttt{id}\quad \mathbb{N} \to \xi.$$

but for convenience, the notation B_i will stand for the indexed formula $\langle \texttt{id}\ i, B\rangle$. Figure 6.5 shows both the teams of agents.

6.2.2 Simply-typed β-normal form λ-terms evidence

This section shows how the p.r.i.c.e. for mimic, defined in section 6.1, can be used modularly alongside a p.r.i.c.e. for simply typed η-long β-normal form λ-terms to check λ-terms that are in β-normal form but not necessarily η-long-normal form.

Team $\texttt{lam}$:

$$\begin{array}{l}
\supset_c (\texttt{lam}\ C\, T, \texttt{lam}\ C\, T).\\
\mathbb{S}^l_c(\texttt{lam}\ C\, T, _, \texttt{lam}\ (C+1)\, T, (\texttt{id}\ C)).\\
\mathbb{S}^r_c(\texttt{lam}\ C\, T, _, \texttt{lam}\ C\, T).\\
\mathbb{D}^l_e(\texttt{lam}\ C\ (I @ L), \texttt{arg}\ C\, L, (\texttt{id}\ C - I - 1)).\\
\mathbb{D}^l_e(\texttt{lam}\ C\ (v\, I), \texttt{arg}\ C\, [], (\texttt{id}\ C - I - 1)).\\
\mathbb{R}^r_e(\texttt{lam}\ C\, T, \texttt{lam}\ C\, T).
\end{array}$$

Team $\texttt{arg}$:

$$\begin{array}{l}
\supset_e (\texttt{arg}\ C\, [T|L], \texttt{lam}\ C\, T, \texttt{arg}\ C\, L).\\
\mathbb{I}^l_e(\texttt{arg}\ C\, []).
\end{array}$$

Figure 6.5: Experts and clerks for teams $\texttt{lam}$ and $\texttt{arg}$

Remark 6.2.2. As seen above, checking the validity of a formula B using a λ-term s as proof evidence and type checking the λ-term s against the type B are two different processes. In particular, while the λ-term has only one type:

$$\cdot \Vdash \lambda x \lambda x.x : A \to B \to B$$

there is no objection to use that same term as evidence for the formula $B \to A \to B$ as well:

$$\lambda x \lambda x.x \Vvdash \cdot \vdash B \to A \to B \quad \textit{and} \quad \lambda x \lambda x.x \Vvdash \cdot \vdash A \to B \to B$$

The light de Bruijn notation would not be sufficient evidence for dealing with non-η-long-normal form. For this section, the spine notation is used but not de Bruijn. In other words terms follow this grammar

$$T, T_1 \cdots T_n ::= \lambda x.T \mid (v\, x)@[T_1 \cdots T_n]$$

where x is a variable name (*e.g.,* a string), and the types of the constructors are:

$$\begin{array}{ll}
\texttt{type}\ \ v & \texttt{string} \to \lambda_t.\\
\texttt{type}\ \ \lambda & \texttt{string} \to \lambda_t \to \lambda_t.
\end{array}$$

The application symbol's type (@) remains unchanged. A variable $(v\ x)$ is written as an application of that variable to an empty list of arguments, *i.e.,*$(v\ x)@[]$.

Modified `case` and index ξ constructors are used for the p.r.i.c.e. definition:

$$\begin{array}{lll} \texttt{type} & \texttt{lam} & \lambda_t \rightarrow \texttt{case}. \\ \texttt{type} & \texttt{arg} & \texttt{list}\ \lambda_t \rightarrow \texttt{case}. \\ \texttt{type} & \texttt{id} & \texttt{string} \rightarrow \xi. \end{array}$$

Example 6.2.3. *Before presenting the agents for this* p.r.i.c.e. *definition, an example of LJF^a derivation is shown in figure 6.6 for the λ-term:*

$$\lambda x \lambda y.y\ [x\ []] \Vdash \cdot \vdash c \rightarrow F \rightarrow (a \rightarrow b) \rightarrow a$$

where $F = c \rightarrow (a \rightarrow b) \rightarrow a$. Notice that the conclusion of the derivation π' is augmented with the case for mimic, seen in 6.1. $\odot$

The agents of this p.r.i.c.e., unsurprisingly similar to the ones seen for the η-long β-normal form λ-terms, are shown in figure 6.7. The differences are: there is now only one decide expert (because variables are considered as being applications with no arguments), no numerical operations are needed as the de Bruijn notation is not used, the store clerk simply uses the name of the variable as index. The most notable difference is marked with $\star$: the store clerk returns the root index $\boxtimes$ (defined for mimic in section 6.1) if the formula to store does not correspond to any λ-abstraction (which means the term is not in η-long-normal form). For the same reason, when the implication expert is called for an implication introduction that does not correspond to any argument, the mimic p.r.i.c.e. takes over and assumes the role of the interlocutor for the kernel.

6.2.3 Dependently-typed β-normal form λ-terms evidence

Dependent types extend the simply typed λ-terms with types that may depend on terms and occupy one of the corners of the Lambda Cube [Barendregt, 1991]. The grammar for the terms may stay the same, using abstraction to bind universally quantified variables, but the grammar for types is changed to:

$$\tau, \sigma ::= \iota \mid \tau \rightarrow \sigma \mid \Pi x : \tau.\sigma$$

where ι can contain λ-terms and $\Pi x : \tau.\sigma$. Dependently typed λ-calculus is a more expressive language where, for example, the type of vectors can depend on an integer indicating their size.

$$\texttt{arg}([\,]) : (a \supset b)_{\boxtimes}, F_y, c_x \Downarrow c \vdash c$$
$$\texttt{lam}(x[\,]) : (a \supset b)_{\boxtimes}, F_y, c_x \Uparrow \cdot \vdash \cdot \Uparrow c$$
$$\texttt{lam}(x[\,]) : (a \supset b)_{\boxtimes}, F_y, c_x \Uparrow \cdot \vdash c \Uparrow \cdot$$
$$\texttt{lam}(x[\,]) : (a \supset b)_{\boxtimes}, F_y, c_x \vdash c \Downarrow \qquad \dfrac{\pi}{\texttt{arg}([\,]) : (a \supset b)_{\boxtimes}, F_y, c_x \Downarrow (a \supset b) \supset a \vdash a}$$
$$\texttt{arg}([x[\,]]) : (a \supset b)_{\boxtimes}, F_y, c_x \Downarrow F \vdash a$$
$$\texttt{lam}(y[x[\,]]) : (a \supset b)_{\boxtimes}, F_y, c_x \Uparrow \cdot \vdash \cdot \Uparrow a$$
$$\texttt{lam}(y[x[\,]]) : (a \supset b)_{\boxtimes}, F_y, c_x \Uparrow \cdot \vdash a \Uparrow \cdot \quad \star$$
$$\texttt{lam}(y[x[\,]]) : F_y, c_x \Uparrow (a \supset b) \vdash a \Uparrow \cdot$$
$$\texttt{lam}(y[x[\,]]) : F_y, c_x \Uparrow \cdot \vdash (a \supset b) \supset a \Uparrow \cdot$$
$$\texttt{lam}(\lambda y.y[x[\,]]) : c_x \Uparrow F \vdash (a \supset b) \supset a \Uparrow \cdot$$
$$\texttt{lam}(\lambda y.y[x[\,]]) : c_x \Uparrow \cdot \vdash F \supset ((a \supset b) \supset a) \Uparrow \cdot$$
$$\texttt{lam}(\lambda x \lambda y.y[x[\,]]) : \cdot \Uparrow c \vdash F \supset ((a \supset b) \supset a) \Uparrow \cdot$$
$$\texttt{lam}(\lambda x \lambda y.y[x[\,]]) : \cdot \Uparrow \cdot \vdash c \supset F \supset ((a \supset b) \supset a) \Uparrow \cdot$$

π:

$$\dfrac{\dfrac{\pi'}{\texttt{mimR}\,\boxtimes : (a \supset b)_{\boxtimes}, F_y, c_x \vdash (a \supset b) \Downarrow} \qquad \dfrac{}{\texttt{arg}([\,]) : (a \supset b)_{\boxtimes}, F_y, c_x \Downarrow a \vdash a}}{\texttt{arg}([\,]) : (a \supset b)_{\boxtimes}, F_y, c_x \Downarrow (a \supset b) \supset a \vdash a}$$

Figure 6.6: Derivation of the example 6.2.3

Team $\texttt{lam}$:

$$\begin{array}{l}
\supset_c (\texttt{lam}\ T, \texttt{lam}\ T).\\
\mathbb{S}^l_c(\texttt{lam}\ \lambda X.T, _, \texttt{lam}\ T, (\texttt{id}\ X)).\\
\mathbb{S}^l_c(\texttt{lam}\ (X@L), _, \texttt{lam}\ (X@L), \boxtimes).\star\\
\mathbb{S}^r_c(\texttt{lam}\ T, _, \texttt{lam}\ T).\\
\mathbb{D}^l_e(\texttt{lam}\ (X@L), \texttt{arg}\ L, (\texttt{id}\ X)).\\
\mathbb{R}^r_e(\texttt{lam}\ T, \texttt{lam}\ T).
\end{array}$$

Team $\texttt{arg}$:

$$\begin{array}{l}
\supset_e (\texttt{arg}\ [T|L], \texttt{lam}\ T, \texttt{arg}\ L).\\
\supset_e (\texttt{arg}\ [], (\texttt{mimR}\ \boxtimes), \texttt{arg}\ []).\star\\
\mathbb{I}^l_e(\texttt{arg}\ []).
\end{array}$$

Figure 6.7: Agents for β-normal λ-terms.

Example 6.2.4. *The following term t takes as arguments:*

- *an append function app applicable to two vectors along with their respective lengths and returning a vector of the added length of those vectors,*
- *some encoding len (in λ-calculus) of an integer,*
- *vector v of length len.*

$$t = \lambda app \lambda len \lambda v.app\,[len, v, len, v]$$

The type of term t is:

$$app_\tau \supset \Pi x : nat \supset vec\, x \supset vec(x + x)$$

where nat denotes the type of integer encoding, $n+m$ is a notation for some encoding of the integer addition and the type of the app function, noted app_τ, is:

$$\Pi n : nat.vec\, n \supset \Pi m : nat.vec\, m \supset vec(n + m)$$

$\odot$

Mapping dependent types into *LJF*

The notion of formulas introduced in definition 2.1.1 needs to be extended to account for dependent-types. There, terms are built from function symbols and variables, which do not allow for λ-terms in the predicates. Here, predicate symbols can have (higher-order) λ-terms but the overall system is still first-order, since the *formulas* are only first-order (no quantification over formulas). In addition, the initial rules of the LJF^a system are are applied modulo $\beta\eta$-equivalence. That is to say, the rules:

$$\frac{}{\Gamma \Downarrow A \vdash A'}\ \mathbb{I}^l \qquad \frac{}{\Gamma, A \vdash A' \Downarrow}\ \mathbb{I}^r$$

succeed if A and A' are equivalent modulo $\beta\eta$-equivalence (polarity requirements still apply). This accounts for the so-called conversion rule of dependently typed lambda calculus:

$$\frac{\Gamma \Vdash t : B' \quad B \approx_{\beta\eta} B'}{\Gamma \Vdash t : B}$$

Encoding Π: A dependent type is encoded into an *LJF* formula using the following function $[\,\cdot\,]^{\lambda\Pi}$:

$$[\tau]^{\lambda\Pi} = \tau^- \quad [A \to B]^{\lambda\Pi} = [A]^{\lambda\Pi} \supset [B]^{\lambda\Pi} \quad [\Pi n : A.B]^{\lambda\Pi} = \forall n.[A]^{\lambda\Pi} \supset [B]^{\lambda\Pi}$$

where τ is a base type that is encoded by an atom with a negative polarity in *LJF*. For example, the type of term t in example 6.2.4 is encoded by the following *LJF* formula:

$$F \supset \forall(\lambda x.nat \supset vec\, x \supset (vec(x + x)))$$

where F is:

$$(\forall(\lambda n.nat \supset vec\, n \supset \forall(\lambda m.nat \supset vec\, m \supset vec(n + m))))$$

Dependent types as p.r.i.c.e.

The λ-term used as evidence are in η-long β-normal form, also called *canonical* terms [Harper and Pfenning, 2005]. To accommodate dependent types, one adds the following universal agents to figure 6.7:

$$\forall_c(\texttt{lam}\ T, \lambda x.\texttt{lam}\ T).$$
$$\forall_e(\texttt{lam}\ T, \texttt{lam}\ T, _).$$

$$
\begin{array}{c}
\vdots \\
\texttt{arg}\ \ ([len, v, len, v])\colon (vec\, u)_v, nat_{len}, F_{app} \Downarrow nat \supset vec\, n \supset G \vdash vec(u+u) \\
\hline
\texttt{arg}\ \ ([len, v, len, v])\colon (vec\, u)_v, nat_{len}, F_{app} \Downarrow F \vdash vec(u+u) \quad \star \\
\hline
\texttt{lam}\ \ (app\, [len, v, len, v])\colon (vec\, u)_v, nat_{len}, F_{app} \Uparrow \cdot \vdash \cdot \Uparrow vec(u+u) \\
\hline\hline
\texttt{lam}\ \ (\lambda v.app\, [len, v, len, v])\colon nat_{len}, F_{app} \Uparrow \cdot \vdash vec\, u \supset (vec(u+u))) \Uparrow \cdot \\
\hline
\texttt{lam}\ \ (\lambda len \lambda v.app\, [len, v, len, v])\colon F_{app} \Uparrow nat \vdash vec\, u \supset (vec(u+u))) \Uparrow \cdot \\
\hline
\texttt{lam}\ \ (\lambda len \lambda v.app\, [len, v, len, v])\colon F_{app} \Uparrow \cdot \vdash nat \supset vec\, u \supset (vec(u+u))) \Uparrow \cdot \\
\hline
\texttt{lam}\ \ (\lambda len \lambda v.app\, [len, v, len, v])\colon F_{app} \Uparrow \cdot \vdash \forall(\lambda x.nat \supset vec\, x \supset (vec(x+x))) \Uparrow \cdot \quad \star \\
\hline
\texttt{lam}\ \ (\lambda app \lambda len \lambda v.app\, [len, v, len, v])\colon\ \cdot \Uparrow F \vdash \forall(\lambda x.nat \supset vec\, x \supset (vec(x+x))) \Uparrow \cdot \\
\hline
\texttt{lam}\ \ (\lambda app \lambda len \lambda v.app\, [len, v, len, v])\colon\ \cdot \Uparrow \cdot \vdash F \supset \forall(\lambda x.nat \supset vec\, x \supset (vec(x+x))) \Uparrow \cdot
\end{array}
$$

Here, $G = \forall(\lambda m.nat \supset vec\, m \supset vec(n+m))$

Figure 6.8: Derivation for example 6.2.4

For instance, the case augmenting the end-sequent of a derivation guided by term t in example 6.2.4 is:

$$\texttt{lam}\ \ (\lambda app \lambda len \lambda v.app\, [len, v, len, v])$$

Part of the derivation is shown in figure 6.8. The only novelties are left and right universal introduction rules, marked with $\star$. The rest of the rules remains unchanged.

Conversion rule

The expressive power of dependently typed λ-calculus partly resides in the conversion rule. To illustrate its use, consider the Church encoding of natural numbers:

$$
\begin{aligned}
\bar{0} &= \lambda f.\lambda x.x \\
\bar{1} &= \lambda f.\lambda x.(fx) \\
\bar{2} &= \lambda f.\lambda x.(f(fx)) \\
\bar{3} &= \lambda f.\lambda x.(f(f(fx))) \\
\bar{4} &= \lambda f.\lambda x.(f(f(f(fx))))
\end{aligned}
$$

and addition and multiplication as:

$$
\begin{aligned}
add &= \lambda m.\lambda n.\lambda f.\lambda x.mf(nfx) \\
mult &= \lambda m.\lambda n.\lambda f.m(nf)
\end{aligned}
$$

With this encoding, the return type of term t from example 6.2.4 can be $vec\, d$ where d is either

$$add\, n\, n = \lambda f.\lambda x.nf(nfx)$$

or

$$mult\, \bar{2}\, n = \lambda f.\bar{2}(nf) = \lambda f.(\lambda f.\lambda x.(f(fx)))(nf) \to_{\beta} add\, n\, n$$

but d cannot be $mult\, n\, \bar{2}$, as that term is not $\beta\eta$-equivalent to $add\, n\, n$ given the chosen encoding.

Chapter 7

Reasoning with equality

Equality is central not only to computer science but also mathematics and physics. It is therefore understandable that handling equality in theorem proving has been at the core of an important research effort in the field of formal logics. This chapter introduces a `p.r.i.c.e.` definition for checking proof evidence involving term equalities and, as an example, shows how it can be used in conjunction with other `p.r.i.c.e.` definitions to describe an equality reasoning technique.

7.1 Introduction

There are a myriad of techniques and ideas to deal with equality in theorem proving, including paramodulation, superposition, narrowing, ρ-calculus and E-unification. There are also practical methods to implement them, such as generating a converging term rewriting system as a decision procedure, saturation methods and redundancy elimination. Given that there are so many ways to discover and represent equality proofs, a scheme for checking such proofs must be flexible.

Equality

The question "what is equality" is often answered in different ways. Occasionally, equality is taken as a primitive logical symbol [Andrews, 1972, Girard, 1992, Schroeder-Heister, 1993]. Sometimes it is defined using Leibniz's (higher-order) rule: two terms are equal if they satisfy exactly the same predicates. More commonly, equality is taken to be a non-logical binary predicate symbol that is axiomatized with rules for reflexivity, symmetry, transitivity, and congruence (for predicates and functions). This latter approach to equality is used in this chapter.

Term rewriting

Term Rewriting is a generic label that designates a plethora of methods for replacing terms with other terms that are considered equal and is an effective tool for reasoning with equality. A rewrite rule is a restriction of an equality in that it is used as a *directed* replacement rule (replace the left argument with the right argument). A set of such rules forms a Term Rewriting System (or TRS). Much research in the area of TRS involves proving properties *about* TRSs—such as confluence, termination, completion, and the decidability of certain set of equalities. Instead, the focus here is on "infrastructure": certifying reasoning that takes place *within* a TRS and is merged with logical deduction.

$\rightharpoonup$ indicates that the equality must be used from left to right (*i.e.*,rewrite s into t), the direction $\leftharpoondown$ ind

7.2 Formalizing equality reasoning

There is only one binary predicate symbol defined for this proof format: the equality predicate, noted $=$ and given a negative polarity. In *LKF*, the negation of an equality predicate $a = b$, like the negation of any atomic formula, is noted $\neg(a = b)$ (and the negation flips the polarity of these occurrences to a positive one).

Equality can then be axiomatized for LJF^a through the following indexed formulas:

$$\begin{aligned}
&\langle \texttt{refl}, \forall x.x = x\rangle\\
&\langle \texttt{sym}, \forall x.\forall y.x = y \supset y = x\rangle\\
&\langle \texttt{trans}, \forall z.\forall x.\forall y.x = y \supset y = z \supset x = z\rangle
\end{aligned}$$

and for LKF^a through the following indexed formulas (negated, because *LKF* is one-sided):

$$\begin{aligned}
&\langle \texttt{refl}, \exists x.\neg(x = x)\rangle\\
&\langle \texttt{sym}, \exists x.\exists y.x = y \wedge^+ \neg(y = x)\rangle\\
&\langle \texttt{trans}, \exists z.\exists x.\exists y.x = y \wedge^+ y = z \wedge^+ \neg(x = z)\rangle
\end{aligned}$$

In addition to these rules, the congruence closures for all function symbol allow traversing the term structure. For example, the congruence rule for a binary function symbol f is written:

$$\frac{x = x' \quad y = y'}{f(x,y) = f(x',y')} \tag{7.1}$$

Remark 7.2.1. Equality reasoning also requires congruence rules for predicates. For *LKF* and for a unary predicate p of negative polarity, they can take the form of:

$$\exists s. \exists s'. (p(s) \wedge^{+} (s = s')) \wedge^{+} \neg(p(s')).$$

However, the p.r.i.c.e. defined in this section, formalizing equality reasoning, is intended to be called by other p.r.i.c.e. definitions. This p.r.i.c.e. definition only deals with equality of terms, ignoring everything concerning the predicates, including their congruence formulas. Therefore, any mention of congruence in this section refers to function congruence.

7.2.1 Smallest common unit

The intended purpose of this p.r.i.c.e. is to be modularly used with multiple proof evidence formats involving term equality. To do so, one has to identify the smallest unit common to those formats. In the case of equational reasoning, a *one-step rewrite* can be such a common unit.

Definition 7.2.1. *Given a set of equalities $\mathcal{E}$, a* one-step rewrite*, relates terms t and s when:*

- *t can be written $C[r]$ (where the context $C[\cdot]$ has exactly one hole);*
- *s can be written $C[u]$;*
- *$r = s \in \mathcal{E}$.*

-

Using this smallest unit would not require transitivity. To use a set of equalities as a TRS, symmetry is not used. This leaves reflexivity, which can also be left aside by modifying the conversion rules. Rule 7.1 makes it harder to limit the number of rewrite steps to one. Indeed, both premises can end either in a rewrite step or with a reflexivity rule. Instead, to require each rewrite operation to be done on *exactly* one subterm, congruence rules are written:

$$\frac{x = x'}{f(x,y) = f(x',y)} \quad \frac{y = y'}{f(x,y) = f(x,y')}$$

and are ensured by the following *LKF* formulas:

$$\exists x. \exists y. \exists x'. ((x = x') \wedge^{+} (\neg(f(x,y) = f(x',y)) \vee^{+} (\neg(f(y,x) = f(y,x')).$$

7.2.2 Proof evidence and system's properties

Performing a rewrite step relies on the following information:

- the set of indexes $\mathcal{E}$ of allowed equalities to use
- the set of indexes $\mathcal{C}$ of congruence rules to allow deep rewriting inside the terms
- the search for a subterm can either be bound by the depth of the subterm (a positive integer) or be unbounded (a negative integer) in the case of closed terms
- a direction in which to use an equality $s = t$: $\rightharpoonup$ indicates that the equality must be used from left to right (*i.e.,*rewrite s into t), $\leftharpoondown$ indicates the opposite direction, and $\rightleftharpoons$ indicates that the equality can be used in both directions

This amount of information is rather minimal as it consists only of the set of equality susceptible of being used is given. But one can give further details such as the exact path to a subterm to rewrite. For simplicity, the p.r.i.c.e. defined here follows minimal proof evidence. In doing so, the checking relies heavily on unification, which is an assumed property of the framework (see section 3.1). To benefit from unification, the reconstructed proof is organized in a more precise manner than in chapters 5 and 6. In particular, when a rule generates two premises one can choose which of them is completed first. This ordering allows to benefit from unification by chosing the premise that maximizes the constraints.

Such a change is a further specification that is not motivated by logic. Indeed, from the perspective of poof theory, there is no notion of order or priority between, say, the two premises of a conjunction rule. Nonetheless, The ultimate and unhidden goal of this thesis, as discussed in section 4.2, is to establish a programmable framework. Such a tuning of the checking mechanism, therefore, seems sensible and comparable to the ordering of clauses in a Prolog program.

To do so, new case constructors are defined:

$$\texttt{type}\quad \texttt{infix}\qquad \overleftarrow{c}\,,\,\overrightarrow{c}\qquad \texttt{case} \to \texttt{case} \to \texttt{case}.$$

Agents of single-premised rules are naïvely defined on these constructors. For example, the existential expert:

$$\exists_e((A\overleftarrow{c}B),(A\overleftarrow{c}B),_).\qquad \exists_e((A\overrightarrow{c}B),(A\overrightarrow{c}B),_).$$

while agents of double-premised rules are extended with one argument and their definitions split these constructors in the following way:

$$\wedge_e^+((A \overleftarrow{c} B), A, B, \texttt{left}). \quad \wedge_e^+((A \overrightarrow{c} B), A, B, \texttt{right}).$$

Notice the extra parameter of this agent, of type δ, indicating which of the premises should be accomplished first in order to extract the most information for unification. The type of the $\wedge_e^+$, seen in figure 4.2, is changed to:

$$\texttt{type} \quad \wedge_e^+ \quad \texttt{case} \to \texttt{case} \to \texttt{case} \to \delta \to o.$$

7.2.3 A p.r.i.c.e. definition for one-step rewrite

The definition given here is relative to an *LKF* kernel; an *LJF* version follows the same steps. One defines the following constructors:

$$\texttt{type} \quad \rightharpoonup, \leftharpoondown, \rightleftharpoons \qquad \texttt{orientation}.$$

$$\texttt{type} \quad \texttt{rew} \qquad \mathbb{N} \to \texttt{list}\ \xi \to \texttt{list}\ \xi \to \mathbb{N} \to \texttt{orientation} \to \texttt{case}.$$

where the first argument of `rew`is a counter used to provide fresh indexes and the rest of the arguments follow, in order, the notion of rewrite proof evidence given above ($\mathcal{E}$,$\mathcal{C}$,depth of rewrite and the orientation of equality:$\rightharpoonup$, $\leftharpoondown$ and $\rightleftharpoons$). For convenience, the counter is written as a superscript on `rew`.

The end-sequent of an LKF^a proof of equality is then:

$$\texttt{rew}^0\, \mathcal{E}\, \mathcal{C}\, D\, O\colon\ \vdash \Gamma \Uparrow t\ \texttt{=}\ s$$

The agents are given in order of appearance in a proof. The store rule is augmented with the following clerk:

$$\mathscr{S}_c((\texttt{rew}^N\ \mathcal{E}\ \mathcal{C}\ D\ O), _, (\texttt{rew}^{(N+1)}\ \mathcal{E}\ \mathcal{C}\ D\ O), (\texttt{link}\ \ (N+1)))$$

where the index constructor is declared by:

$$\texttt{type} \quad \texttt{link} \quad \mathbb{N} \to \xi.$$

which uniquely identifies the intermediate equalities. In particular, the formula stored with the current value of the counter N is the last stored formula, subsequently called *last formula*. The next rule is a decide on either of the following:

- a member of $\mathcal{C}$, which succeeds if the depth counter D is either strictly positive (the bound is not reached) or strictly negative (there is no bound) and the two arguments of the last formula share the top function symbol;

- one of the equalities of $\mathcal{E}$, which succeeds if the last formula is an instance of that equality following the orientation O that is the last argument of the `rew` case.

In the first case, the decide expert has the following definition:

$$\mathscr{D}_e((\texttt{rew}^N\ \mathcal{E}\ \mathcal{C}\ D\ O), (\texttt{rew}^N\ \mathcal{E}\ \mathcal{C}\ (D-1)\ O)\overrightarrow{c}(\texttt{initWith (link } N)), I)\ \ \texttt{:-}\ \ (D > 0; D < 0), I \in \mathcal{C}.$$

In the second case, the decide expert has three definitions, depending on the orientation O in which to use the equality:

$$\mathscr{D}_e((\texttt{rew}^N \mathcal{E}\ \mathcal{C}\ D \rightharpoonup), (\texttt{initWith (link } N)), I)\ \ \texttt{:-}\ \ I \in \mathcal{E}.$$
$$\mathscr{D}_e((\texttt{rew}^N \mathcal{E}\ \mathcal{C}\ D \leftharpoondown), (\Psi)\overrightarrow{c}(\texttt{initWith (link } N)), \texttt{sym}).$$
$$\mathscr{D}_e((\texttt{rew}^N \mathcal{E}\ \mathcal{C}\ D \rightleftharpoons), \Xi, I)\ \ \texttt{:-}\ \ \mathscr{D}_e((\texttt{rew}^N \mathcal{E}\ \mathcal{C}\ D \rightharpoonup, \Xi, I); \mathscr{D}_e((\texttt{rew}^N \mathcal{E}\ \mathcal{C}\ D \leftharpoondown), \Xi, I).$$

where Ψ : `tag from` (`oneOf` $\mathcal{E}$(`initWith` [`from`])). After a decide, the existential and positive conjunction rules use the previously seen definitions of their respective agents.

7.3 Paramodulation

Robinson and Wos [1983] introduced paramodulation as a generalization of resolution in order to include equality and isolate the inference apparatus dealing with equality. Despite it being one of the earliest methods of reasoning for such problems, paramodulation is still well suited for various problem domains in group and ring theory.

Definition 7.3.1. *Given clauses A and $(\alpha' = \beta') \vee B$, having no variables in common and such that A contains a term δ with δ and α' having most general common instance α identical to $\alpha'[s_i/u_i]$ and $\delta[t_j/w_j]$, a paramodulation application infers the clause $A' \vee B[s_i/u_i]$, called* paramodulant*, where A' is obtained by replacing in A a single occurrence of α (resulting from an occurrence of δ) by $\beta'[s_i/u_i]$.* •

Example 7.3.1. *From $f(x, g(x)) = e \vee q(x)$ and $p(y, f(g(y), z), z) \vee w(z)$ one can infer $p(y, e, g(g(y))) \vee q(g(y)) \vee w(g(g(y)))$ by paramodulating with $f(x, g(x))$ as α' and $f(g(y), z)$ as δ.* ⊙

Paramodulation is restricted here to unit clauses and the equality predicate $=$ is the only atomic formula and is given a negative polarity.

Example 7.3.2. *The set of containing equalities* 1, 2, 3 *below is refuted through paramodulation:*

1. $\neg(h(g(g(c))) = g(g(g(c))))$.
2. $\forall x.\forall y.h(f(g(x), y)) = g(y)$.
3. $\forall x.f(x, g(x)) = g(x)$.

   $- - - - - - - - - - - - - - - - - - -$
4. $\forall x.h(g(g(x))) = g(g(g(x)))$*(from* 3 *into* 2*).*
5. $false$ *(from* 1 *and* 4*).*

⊙

There are clear similarities with the resolution example 5.3.1. In fact, the `p.r.i.c.e.` definition introduced in section 5.3.1 for a refutation sequence can also be used for paramodulation. Indeed, the *sequence checker* `p.r.i.c.e.` is defined specifically to check a refutation sequence without knowledge of how the individual steps of that sequence are carried out and is used alongside a resolution `p.r.i.c.e.` and a hyperresolution `p.r.i.c.e.`.

In this section, the *sequence checker* `p.r.i.c.e.` is used with a `p.r.i.c.e.` for a paramodulation step to describe the semantics for paramodulation refutations. This work is part of the paper [Chihani et al., 2015].

7.3.1 Describing a paramodulation in *LJF*^{a}

Since the paramodulation refutation sequence is mapped in the same way a resolution refutation is (*i.e.,* through a backbone of cut rule applications) this section focuses on the mapping of an individual paramodulation step, corresponding to the left premise of each cut rule. The mapping of each step forms the prototypical *LKF* derivation shown in figure 7.1.

The proof starts by introducing any universally quantified variables in $r[s']$, which is then stored under index `res` (for *result*). The proof continues by deciding, on the predicate congruence indexed by `pred`. Existentially quantified variables are then replaced by logic variables (to be instantiated later), X for x, X' for x' and as many logic variable as there are variables in $\bar{z}$, some of which appear in r. Then three premises π_1, π_2 and π_3 are completed in this order.

The conclusion of π_1 is a sequent focused on a positive atom, it ends with an initial rule on the `res`-indexed formula and fixes the logic variable X' to s', as well as any logic variable in $r[X']$. These instantiations are passed to the other premises.

$$
\cfrac{
  \cfrac{
    \cfrac{\cfrac{\pi_2}{\vdash \Gamma' \Downarrow r[X]} \quad \cfrac{\pi_3}{\vdash \Gamma' \Downarrow X = X'}}{\vdash \Gamma' \Downarrow r[X] \wedge^+ X = X'} \wedge^+ \quad \cfrac{\pi_1}{\vdash \Gamma' \Downarrow \neg(r[X'])}
  }{\vdash \Gamma' \Downarrow (r[X] \wedge^+ X = X') \wedge^+ \neg(r[X'])} \wedge^+
}{
  \cfrac{\vdash \Gamma' \Downarrow \exists x'.\exists x.\exists \bar{z}.(r[x] \wedge^+ x = x') \wedge^+ \neg(r[x'])}{
    \cfrac{\cfrac{\vdash \Gamma' \Uparrow \cdot}{\vdash \Gamma \Uparrow r[s']} \mathscr{S}}{\vdash \Gamma \Uparrow \forall \bar{z}.r[s']} \forall
  } \mathscr{D}\ on\ \texttt{pred}
} \exists
$$

$\Gamma: \quad \{(\exists x'.\exists x.\exists \bar{z}.(r[x] \wedge^+ x = x') \wedge^+ \neg(r[x']))_{\texttt{pred}}, (\exists \bar{x}.\neg(s = s'))_F, (\exists \bar{z}.\neg(r[s]))_I\}$

$\Gamma': \quad \{\Gamma, (r[s'])_{\texttt{res}}\}$

Here, the equality indexed by F (called the *from* equality) is used to rewrite *into* the formula indexed by I and the predicate r has a negative polarity.

Figure 7.1: Prototypical paramodulation derivation

The conclusion of π_2 is focused on a negative atom, it is released and stored under index `into`. Then a decide is made on the formula indexed by I and the logic variables of $r[X]$ are unified with those of $r[s]$.

The final proof, π_3, is a rewrite of term s into s' using the *from* equality.

Unit equality

For the particular case of unit-equality paramodulation, the predicate symbol r is always the equality predicate. Thus the only predicate congruence rules are:

$$
\langle \texttt{pred}\,1, \exists x'.\exists x.\exists y.x = y \wedge^+ x = x' \wedge^+ \neg(x' = y)\rangle
$$
$$
\langle \texttt{pred}\,2, \exists x'.\exists x.\exists y.y = x \wedge^+ x = x' \wedge^+ \neg(y = x')\rangle
$$

where $r[\cdot]$ is $\square = y$ in the first formula, $y = \square$ in the second formula, and in both cases the extra variables $\bar{z}$ are the single variable y. In what follows, the notation the notation $r[\cdot]$ is used to avoid confusion between the *from* equality (noted as equality) and the *into* equality (noted as a predicate r with a hole).

7.3.2 A p.r.i.c.e. for a paramodulation step

The derivation shown in figure 7.1 can be seen as split in four regions. The proof starts in a region that uses the congruence for predicates and ends with the three premises π_1, π_2 and π_3. Then, one region for each of these proofs.

This first region must keep track (similar to the binary resolution p.r.i.c.e.) of two indexes: I and F. Because paramodulation p.r.i.c.e. is added to the *sequence checker* p.r.i.c.e., some interface between the is used. Recall the cut expert definition seen in the definition of the *sequence checker* p.r.i.c.e. (section 5.3.1):

$$\mathscr{C}_e(\texttt{rlist}\ [\langle\mathcal{S}, K, C_k\rangle|R], \texttt{step}\ \mathcal{S}, \texttt{rlisti}\ K\ R, C_k)$$

where the type of $\mathcal{S}$ is simply fixed at `stuple`. Recall also that this type is given different constructors for resolution and hyperresolution. Here, it is given yet another constructor for paramodulation:

$$\texttt{type}\quad \texttt{pm}\qquad \xi \to \xi \to \texttt{stuple}.$$

where the first argument of `pm` is the *from* equality and the second argument is the *into* clause. The augmented sequent of the left premise of the cut is:

$$\texttt{step}\ (\texttt{pm}\ I\ F):\ \vdash \Gamma \Uparrow \forall \bar{z}.r[s']$$

The checking thus starts with an invertible phase: introducing eigenvariables for each universally quantified variable in the paramodulant clause.

$$\forall_c(\texttt{step}\ (\texttt{pm}\ I\ F), \lambda x.(\texttt{pm}\ I\ F)).$$

Then, it stores the resulting negative atom (recall that the equality predicate symbol was given a negative polarity):

$$\mathscr{S}_c(\texttt{step}\ (\texttt{pm}\ I\ F), _, \texttt{step}\ (\texttt{pm}\ I\ F), \texttt{res}).$$

where `res` is an index constructor.

Once the invertible phase ends, a decide must be done on one of the congruence rules for predicates. Let the indexes of these rules be stored in a list $\mathcal{Q}$. Then the decide expert has the following definition:

$$\mathscr{D}_e(\texttt{step}\ (\texttt{pm}\ I\ F), (\Xi \overleftarrow{c}\ (\texttt{rew}^0[F]\mathcal{C}\ (-1) \rightleftharpoons)) \overrightarrow{c}\ (\texttt{initWith}\ [\texttt{res}]), P)\quad \texttt{:-}\quad P \in \mathcal{Q}.$$

where $\Xi : \texttt{tag into}\ (\texttt{oneOf}\ [I](\texttt{initWith}\ [\texttt{into}]))$ and $\mathcal{C}$ is the set of function congruence rules.

At the boundary of this region are the three conclusions of the proofs π_1, π_2, π_3. They are now augmented to yield the sequents:

$$\frac{\pi_1}{\texttt{initWith [res]}: \vdash \Gamma' \Downarrow \neg(r[X'])}$$

$$\frac{\pi_2}{\texttt{tag}\ I\ \Xi: \vdash \Gamma' \Downarrow r[X]}$$

$$\frac{\pi_3}{\texttt{rew}^0[F]\mathcal{C}\ (-1) \rightleftharpoons: \vdash \Gamma' \Downarrow X\ \texttt{=}\ X'}$$

By establishing an order of verification between these premises (discussed in section 7.2.2), the proof π_1 is completed first using the default `initWith` team presented in section 4.3.1. This instantiates the logic variable X' with the term s', which is propagated to the other premises. The sequent concluding π_2 is solved next through the default teams `tag` and `initWith` defined in sections 4.3.4 and 4.3.1. When π_2 is performed, the logic variable X is constrained to s. The last premise becomes:

$$\frac{\pi_3}{\texttt{rew}^0[F]\mathcal{C}\ (-1) \rightleftharpoons: \vdash \Gamma' \Downarrow s\ \texttt{=}\ s'}$$

with F the index of the rewrite rule to use on subterms of s and s'. This proof is completed following the guidance of the one-step rewrite p.r.i.c.e. shown in section 7.2.3.

Chapter 8

Satellite interests

In this chapter, further exploration of the framework's expressiveness is carried out. After having presented case studies for classical and intuitionistic logics, this chapter proposes elaborate ways to work on p.r.i.c.e. objects.

8.1 Hosting kernels

If they find a parrot who could answer to everything, I would claim it to be an intelligent being without hesitation.

- Denis Diderot *Penses philosophiques*

The possibility to recover classical provability through intuitionistic provability has been known for a long time. To what extent can one make the same claim regarding proofs? In other words, can a classical proof be mapped to an intuitionistic proof with the same structure? And if so, can an intuitionistic checker trick the user into thinking it is a classical checker by following the guidance of a classical proof certificate as closely as a classical checker would?

The independence of proof certificates from the technology that generated them is stated as one of the main goals of the Foundational Proof Certification framework. This chapter answers the above questions, pushing the independence of proof certificates even further. Not only are they independent of the technology that created them, but they can also be, to some extent, independent from the technology that checks them.

8.1.1 Introduction

Three levels of adequacy are identified in the literature when it comes to encoding a source proof system A into a target proof system B (*e.g.,* [Nigam and Miller, 2010] or [Girard, 2006] for a semantically similar classification). The first level is adequacy of provability, where a theorem T is provable in A if and only if its encoding T' is provable in B. The second level is that of (closed) proofs, where every closed proof of T in A corresponds to one and only one proof of T' in B. The last level is that of derivations, or *open* proofs where even incomplete proofs of T in A are in one-to-one correspondence with incomplete proofs of T' in B.

This last level of adequacy is of particular interest in the discipline of proof *checking* as opposed to *theorem proving*. Indeed, classical provability of a theorem T can be established in an intuitionistic prover using a double-negation translation of T. But if one wants to *check* a classical proof certificate using an intuitionistic checker, this checker must be able to *understand* the guiding information enclosed in that certificate. In particular, the goal is then to have an intuitionistic proof checker imitate every step of a classical proof checker. By doing so, not only is the number of kernels to implement reduced, but the proof certificates gain more independence as well.

History

The difference between classical and intuitionistic logics was brought down to the syntactic level in the original sequent calculus papers by Gentzen, where he reduced this difference to a restriction on the number of formulas allowed on the right-hand side of his *LK* and *LJ* sequents (at most one in *LJ*). Later on, this right-hand side cardinality restriction was further reduced to only universal and implication rules, the earliest such systems were introduced in [Kleene, 1952] and in [Maehara, 1954] (in German, described in [Takeuti, 1987]).

Furthermore, several embeddings at the level of provability, in both directions, were proposed in numerous papers. From classical provability to intuitionistic provability, various double-negation translations ([Gödel, 1932, Gentzen, 1936, Kolmogorov, 1925], *etc.*) map classical logic provability directly into intuitionistic logic. From intuitionistic provability to classical provability, embeddings need logical devices that are external to classical logic. For example, modal logic [Gödel, 1933] or linear logic connectives [Girard, 1987].

Embeddings from classical to intuitionistic systems is arguably more direct than the converse (in that it does not require any external artifacts). Therefore, the following sections explore such an embedding, this time at the level of proofs, from

$$
\begin{array}{rcl@{\qquad}rcl}
\lceil t^+ \rceil^+ & = & t & \lceil t^- \rceil^- & = & f \\
\lceil f^+ \rceil^+ & = & f & \lceil f^- \rceil^- & = & t \\
\lceil B \vee^+ C \rceil^+ & = & \lceil B \rceil^+ \vee \lceil C \rceil^+ & \lceil B \vee^- C \rceil^- & = & \lceil B \rceil^- \wedge^+ \lceil C \rceil^- \\
\lceil B \wedge^+ C \rceil^+ & = & \lceil B \rceil^+ \wedge^+ \lceil C \rceil^+ & \lceil B \wedge^- C \rceil^- & = & \lceil B \rceil^- \vee \lceil C \rceil^- \\
\lceil \exists x.A \rceil^+ & = & \exists x.\lceil A \rceil^+ & \lceil \forall x.A \rceil^- & = & \exists x.\lceil A \rceil^- \\
\lceil A^+ \rceil^+ & = & A^+ & \lceil A^- \rceil^- & = & A^+ \\
\lceil N \rceil^+ & = & \lceil N \rceil^- \supset q & \lceil P \rceil^- & = & \lceil P \rceil^+ \supset q
\end{array}
$$

Figure 8.1: The functions $\lceil\cdot\rceil^+$ and $\lceil\cdot\rceil^-$ which map *LKF* formulas onto *LJF* formulas.

LKF to *LJF*.

8.1.2 Mapping *LKF* sequents to *LJF* sequents

Because most double-negation translations are defined on unpolarized syntax, the embedding they produce lacks the desired precision. The Foundational Proof Certification framework, on the other hand, relies on a polarized syntax, therefore a different kind of embedding is needed. The one presented in this section relies on results from Chaudhuri [2010]. Figure 8.1 describes two mappings, $\lceil\cdot\rceil^+$ and $\lceil\cdot\rceil^-$, of *LKF* formulas to *LJF* formulas, where q is a fixed, negative atom that is not allowed in the input *LKF* formulas and all other atoms in the target *LJF* formula are assigned positive polarity (A^+). The embedding, defined through both the mappings $\lceil\cdot\rceil^+$ and $\lceil\cdot\rceil^-$, is noted $\lceil\cdot\rceil^\pm$ and called the *focused embedding*.

Remark 8.1.1. The output formulas of $\lceil\cdot\rceil^\pm$ are either a positive formula or an implication of the form $B \supset q$ for some *positive* formula B (the output of $\lceil\cdot\rceil^+$ on a positive *LKF* formula and of $\lceil\cdot\rceil^-$ on a negative *LKF* formula can only be a positive *LJF* formula). This implies that focused phases on the left-hand side can only introduce the implication (and the negative atom q) while all other focusing rules introduce connectives on the right-hand side. Furthermore, a focused phase on the left contains at most one implication left-introduction rule. Symmetrically, all invertible introduction rules appear on the left-hand side except the implication right-introduction rule. Overall, it is invariably true that the left-hand side storage of *LJF* only contains implications and positive atoms. It is also invariably true that the right-hand side storage can only contain the negative atom q. Indeed, the only negative formulas appearing on the right are implication formulas, and the only implication formulas have q for right subformula.

It is proven by Chaudhuri [2010, Theorem 12] that the focused embedding pre-

serves adequacy at the level of phases between *LKF* and *LJF*. More precisely, Chaudhuri shows that:

- an *LKF*-phase ending with the sequent $\vdash \Theta \Uparrow \Gamma$ corresponds to an LJF-phase ending with the sequent $\lceil\Theta\rceil^{-} \Uparrow \lceil\Gamma\rceil^{-} \vdash q\Uparrow$, and
- an LKF-phase ending with the sequent $\vdash \Theta \Downarrow B$ corresponds to an LJF-phase ending with the sequent $\lceil\Theta\rceil^{-} \vdash \lceil B\rceil^{+} \Downarrow$

where $\lceil\Gamma\rceil^{-}$ is a notation for the set $\{\lceil D\rceil^{-} \mid D \in \Gamma\}$.

Example 8.1.1. *To illustrate this translation, consider the following LKF formula $F = a \vee^+ \neg a$ where a is a positive atom. An LKF proof of this formula is:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a \vee^+ \neg a, \neg a \Downarrow a}\ \mathscr{I}}{\vdash a \vee^+ \neg a, \neg a \Downarrow a \vee^+ \neg a}\ \vee^+}{\vdash a \vee^+ \neg a, \neg a \Uparrow \cdot}\ \mathscr{D}}{\vdash a \vee^+ \neg a \Uparrow \neg a}\ \mathscr{S}}{\vdash a \vee^+ \neg a \Downarrow \neg a}\ \mathscr{R}}{\vdash a \vee^+ \neg a \Downarrow a \vee^+ \neg a}\ \vee^+}{\vdash a \vee^+ \neg a \Uparrow \cdot}\ \mathscr{D}}{\vdash \cdot \Uparrow a \vee^+ \neg a}\ \mathscr{S}$$

which is mapped to the following LJF proof of the translated end-sequent, where $\lceil F\rceil^{-} = (a \vee (a \supset q)) \supset q$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\lceil F\rceil^{-}, a \vdash a \Downarrow}\ \mathbb{I}^r}{\lceil F\rceil^{-}, a \vdash a \vee (a \supset q) \Downarrow}\ \vee \qquad \dfrac{}{\cdots \Downarrow q \vdash q}}{\lceil F\rceil^{-}, a \Downarrow (a \vee (a \supset q)) \supset q \vdash q}\ \supset_l}{\lceil F\rceil^{-}, a \Uparrow \cdot \vdash \cdot \Uparrow q}\ \mathbb{D}^l}{\lceil F\rceil^{-}, a \Uparrow \cdot \vdash q \Uparrow \cdot}\ \mathbb{S}^r}{\lceil F\rceil^{-} \Uparrow a \vdash q \Uparrow \cdot}\ \mathbb{S}^l}{\lceil F\rceil^{-} \Uparrow \cdot \vdash a \supset q \Uparrow \cdot}\ \supset_r}{\lceil F\rceil^{-} \vdash a \supset q \Downarrow}\ \mathbb{R}^r}{\lceil F\rceil^{-} \vdash a \vee (a \supset q) \Downarrow}\ \vee \qquad \dfrac{}{\cdots \Downarrow q \vdash q}}{\lceil F\rceil^{-} \Downarrow (a \vee (a \supset q)) \supset q \vdash q}\ \supset_l}{\lceil F\rceil^{-} \Uparrow \cdot \vdash \cdot \Uparrow q}\ \mathbb{D}^l}{\lceil F\rceil^{-} \Uparrow \cdot \vdash q \Uparrow \cdot}\ \mathbb{S}^r}{\cdot \Uparrow \lceil F\rceil^{-} \vdash q \Uparrow \cdot}\ \mathbb{S}^l$$

$\odot$

Example 8.1.2. *Now consider an LKF proof of another polarization of the excluded middle: $F' = a \vee^- \neg a$ where a is a positive atom:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a, \neg a \Downarrow a}\;\mathscr{I}}{\vdash a, \neg a \Uparrow \cdot}\;\mathscr{D}}{\vdash a \Uparrow \neg a}\;\mathscr{S}}{\vdash \cdot \Uparrow a, \neg a}\;\mathscr{S}}{\vdash \cdot \Uparrow a \vee^- \neg a}\;\vee^-$$

which is mapped to the following LJF proof, where $\lceil F' \rceil^- = a \wedge^+ (a \supset q)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a, a \supset q \vdash a \Downarrow}\;\mathbb{I}^r \quad \dfrac{}{a, a \supset q \Downarrow q \vdash q}\;\mathbb{I}^l}{a, a \supset q \Downarrow a \supset q \vdash q}\;\supset_l}{a, a \supset q \Uparrow \cdot \vdash \cdot \Uparrow q}\;\mathbb{D}^l}{a, a \supset q \Uparrow \cdot \vdash q \Uparrow \cdot}\;\mathbb{S}^r}{a \Uparrow a \supset q \vdash q \Uparrow \cdot}\;\mathbb{S}^l}{\cdot \Uparrow a, a \supset q \vdash q \Uparrow \cdot}\;\mathbb{S}^l}{\cdot \Uparrow a \wedge^+ (a \supset q) \vdash q \Uparrow \cdot}\;\wedge^+$$

$\odot$

These two examples illustrate the following one-to-one mappings of *LKF* rules (on the left-hand side of $\hookrightarrow$) and *LJF* rules (on the right-hand side of $\hookrightarrow$):

$$\mathscr{S} \hookrightarrow \mathbb{S}^l \qquad \mathscr{D} \hookrightarrow \mathbb{D}^l \qquad \vee^+ \hookrightarrow \vee_r \qquad \mathscr{R} \hookrightarrow \mathbb{R}^r \qquad \mathscr{I} \hookrightarrow \mathbb{I}^r \qquad \vee^- \hookrightarrow \wedge^+_l$$

Other one-to-one mappings, not seen in the above derivations, are as follows:

$$\wedge^+ \hookrightarrow \wedge^+_r \qquad \wedge^- \hookrightarrow \vee_l \qquad \forall \hookrightarrow \exists_l \qquad \exists \hookrightarrow \exists_r \qquad f^- \hookrightarrow t_l \qquad t^- \hookrightarrow f_l \qquad t^+ \hookrightarrow t_r$$

In addition to these rules, some *LJF* rules do not have *LKF* counterparts but are introduced by the embedding itself: the left and right $\supset$ rules, the store right-rule $\mathbb{S}^r$, the initial left-rule $\mathbb{I}^l$. Other rules of *LJF*, namely $\mathbb{R}^l$, $\mathbb{D}^r$ and introduction rules for all negative connectives, never appear.

The *LKF* and *LJF* sequent calculi used in this thesis are more deterministic than the ones used by Chaudhuri [2010]. Indeed, both the workbench of *LKF* and the left-hand side workbench of *LJF* are lists and the invertible rules applications are always performed on the left-most formula. The theorem below is a more precise version of Theorem 12 in Chaudhuri's paper.

Theorem 8.1.1. *There is a rule-preserving map of LKF proofs into LJF proofs: that is, for a fixed (negatively polarized) atom q, the following two statements hold:*

$$\begin{array}{lcl} \vdash \Gamma \Uparrow \Theta & \hookrightarrow & \lceil\Gamma\rceil^{-} \Uparrow \lceil\Theta\rceil^{-} \vdash q \Uparrow \cdot \quad \textit{or} \quad \lceil\Gamma\rceil^{-} \Uparrow \lceil\Theta\rceil^{-} \vdash \cdot \Uparrow q \\ \vdash \Gamma \Downarrow B & \hookrightarrow & \lceil\Gamma\rceil^{-} \vdash \lceil B\rceil^{+} \Downarrow \end{array}$$

Proof. The two statements are proven simultaneously and by induction on the derivation. All but the structural rules of release and decide in LKF are mapped to a single proof rule in LJF. An example is the positive disjunction rule:

$$\frac{\vdash \Gamma \Downarrow B_i}{\vdash \Gamma \Downarrow B_1 \vee^{+} B_2}\ \vee^{+} \quad \hookrightarrow \quad \frac{\lceil\Gamma\rceil^{-} \vdash \lceil B_i\rceil^{+} \Downarrow}{\lceil\Gamma\rceil^{-} \vdash \lceil B_1\rceil^{+} \vee \lceil B_2\rceil^{+} \Downarrow}\ \vee_r$$

where $i \in \{1,2\}$, and $\lceil B_1 \vee^{+} B_2\rceil^{+} = \lceil B_1\rceil^{+} \vee \lceil B_2\rceil^{+}$. The negative disjunction rule is also mapped to only one *LJF* rule:

$$\frac{\vdash \Gamma \Uparrow A, B, \Theta}{\vdash \Gamma \Uparrow A \vee^{-} B, \Theta} \quad \hookrightarrow \quad \frac{\lceil\Gamma\rceil^{-} \Uparrow \lceil A\rceil^{-}, \lceil B\rceil^{-}, \lceil\Theta\rceil^{-} \vdash q \Uparrow \cdot}{\lceil\Gamma\rceil^{-} \Uparrow \lceil A \vee^{-} B\rceil^{-}, \lceil\Theta\rceil^{-} \vdash q \Uparrow \cdot}\ \wedge_l^{+}$$

where $\lceil A \vee^{-} B\rceil^{-} = \lceil A\rceil^{-} \wedge^{+} \lceil B\rceil^{-}$.

The case for rules of release and decide of LKF involves more rules (namely, implication $\supset$ and initial $\mathbb{I}$ rules). The mapping of the release rule is:

$$\frac{\vdash \Gamma \Uparrow N}{\vdash \Gamma \Downarrow N}\ \mathscr{R} \quad \hookrightarrow \quad \cfrac{\cfrac{\lceil\Gamma\rceil^{-} \Uparrow \lceil N\rceil^{-} \vdash q \Uparrow \cdot}{\lceil\Gamma\rceil^{-} \Uparrow \cdot \vdash \lceil N\rceil^{+} \Uparrow \cdot}\ \supset_r}{\lceil\Gamma\rceil^{-} \vdash \lceil N\rceil^{+} \Downarrow}\ \mathbb{R}^r$$

where $\lceil N\rceil^{+} = \lceil N\rceil^{-} \supset q$, prompting the $\supset_r$ rule. The decide rule is mapped as:

$$\frac{\vdash P, \Gamma \Downarrow P}{\vdash P, \Gamma \Uparrow \cdot}\ \mathscr{D} \quad \hookrightarrow \quad \cfrac{\cfrac{\lceil\Gamma\rceil^{-} \vdash \lceil P\rceil^{+} \Downarrow \qquad \cfrac{}{\lceil\Gamma\rceil^{-} \Downarrow q \vdash q}\ \mathbb{I}^l}{\lceil\Gamma\rceil^{-}, \lceil P\rceil^{-} \Downarrow \lceil P\rceil^{-} \vdash q}\ \supset_l}{\lceil\Gamma\rceil^{-}, \lceil P\rceil^{-} \Uparrow \cdot \vdash \cdot \Uparrow q}\ \mathbb{D}^l$$

where $\lceil P\rceil^{-} = \lceil P\rceil^{+} \supset q$, prompting the $\supset_l$ rule. □

This embedding is not limited to cut-free *LKF* proofs. Indeed, an *LKF* cut is also mapped to an *LJF* cut. The *LKF* cut, seen in figure 2.4 yields two up-arrow sequents as premises and introduces two formulas, one positive and one negative. Take B to be the positive formula, without loss of generality, then $\neg B$ is necessarily a negative formula (from the de Morgan duality rules). The first premise will immediately go

into a store, the second premise engages in an invertible phase. The LJF cut follows the same structure :

$$\dfrac{\vdash \Gamma \Uparrow \neg B \qquad \dfrac{\vdash \Gamma, B \Uparrow \cdot}{\vdash \Gamma \Uparrow B}}{\vdash \Gamma \Uparrow \cdot}\ \mathscr{C} \quad \hookrightarrow \quad \dfrac{\dfrac{\Gamma \Uparrow \lceil B \rceil^{+} \vdash q \Uparrow \cdot}{\Gamma \Uparrow \cdot \vdash \lceil B \rceil^{-} \Uparrow \cdot}\ \supset_r \qquad \dfrac{\Gamma, \lceil B \rceil^{-} \Uparrow \cdot \vdash \cdot \Uparrow q}{\Gamma \Uparrow \lceil B \rceil^{-} \vdash \cdot \Uparrow q}}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow q}\ \mathbb{C}$$

The proof follows from proposition 8.1.1 bellow:

Proposition 8.1.1. *For any polarized classical formula B, $\lceil \neg B \rceil^{+} = \lceil B \rceil^{-}$ and $\lceil \neg B \rceil^{-} = \lceil B \rceil^{+}$*

Proof. By induction on the structure of B.

For example, if $B = C \wedge^{-} D$ and $\neg B = \neg C \vee^{+} \neg D$, then

$$\lceil B \rceil^{+} = \lceil C \wedge^{-} D \rceil^{-} \supset q = (\lceil C \rceil^{-} \vee \lceil D \rceil^{-}) \supset q = F$$

and by induction:

$$F = (\lceil \neg C \rceil^{+} \vee \lceil \neg D \rceil^{+}) \supset q = \lceil \neg C \vee \neg D \rceil^{+} \supset q = \lceil \neg B \rceil^{+} \supset q = \lceil \neg B \rceil^{-}$$

□

8.1.3 Mapping $\boldsymbol{LKF}^a$ checking to $\boldsymbol{LJF}^a$ checking

The above embedding is at the third level of adequacy from *LKF* to *LJF*. This makes it possible for an LJF^a kernel to check proof certificates written for an LKF^a kernel without it being apparent. This is done through a simple interface between agents of the two augmented systems seen in figure 8.2. Agents augmenting the LKF^a system are preceded with superscript ^{LKF} and agents augmenting the LJF^a system are preceded with superscript ^{LJF}. The other parts of p.r.i.c.e. need not be redefined: the case and index constructors are the same and the polarity assignment is uniquely determined through the $\lceil \cdot \rceil^{\pm}$ above. An added case constructor:

```
type  qc  case.
```

is used in dealing with the negative atom q.

The case for the cut rule is more subtle. The cut expert $\mathscr{C}_e$ provides a formula B, the kernel uses B on the left premise and its negation $\neg B$ on the right premise. While the *LKF* cut rule seen above appears to be mapped to a single *LJF* cut rule,

$$
\begin{array}{lll}
{}^{LJF} \vee_c (\Xi_0, \Xi_1, \Xi_2) & \texttt{:-} & {}^{LKF} \wedge_c^- (\Xi_0, \Xi_1, \Xi_2). \\
{}^{LJF} \wedge_c^+ (\Xi, \Xi') & \texttt{:-} & {}^{LKF} \vee_c^- (\Xi, \Xi'). \\
{}^{LJF} \wedge_e^+ (\Xi_0, \Xi_1, \Xi_2) & \texttt{:-} & {}^{LKF} \wedge_e^+ (\Xi_0, \Xi_1, \Xi_2). \\
{}^{LJF} \vee_e (\Xi, \Xi', C) & \texttt{:-} & {}^{LKF} \vee_e^+ (\Xi, \Xi', C). \\
{}^{LJF}\exists_e(\Xi, \Xi', W) & \texttt{:-} & {}^{LKF}\exists_e(\Xi, \Xi', W). \\
{}^{LJF}\exists_c(\Xi, \Xi') & \texttt{:-} & {}^{LKF}\forall_c(\Xi, \Xi'). \\
{}^{LJF}t_e(\Xi) & \texttt{:-} & {}^{LKF}t_e^+(\Xi). \\
{}^{LJF}t_c(\Xi, \Xi') & \texttt{:-} & {}^{LKF}f_c^-(\Xi, \Xi'). \\
{}^{LJF}\mathbb{S}_c^l(\Xi, F, \Xi', I) & \texttt{:-} & {}^{LKF}\mathscr{S}_c(\Xi, F, \Xi', I). \\
{}^{LJF}\mathbb{D}_e^l(\Xi, \Xi', I) & \texttt{:-} & {}^{LKF}\mathscr{D}_e(\Xi, \Xi', I). \\
{}^{LJF}\mathbb{R}_e^l(\Xi, \Xi') & \texttt{:-} & {}^{LKF}\mathscr{R}_e(\Xi, \Xi'). \\
{}^{LJF}\mathbb{I}_e^r(\Xi, I) & \texttt{:-} & {}^{LKF}\mathscr{I}_e(\Xi, I). \\
{}^{LJF} \supset_c (\Xi, \Xi). & & \\
{}^{LJF}\mathbb{S}_c^r(\Xi, \Xi, _). & & \\
{}^{LJF} \supset_e (\Xi, \Xi, \mathsf{qc}). & & \\
{}^{LJF}\mathbb{I}_e^l(\mathsf{qc}). & &
\end{array}
$$

Figure 8.2: Interfacing LJF^a agents to LKF^a agents

$$\frac{\Xi_1 \vdash \Theta \Uparrow N \quad \Xi_2 \vdash \Theta \Uparrow \neg N \quad cut_e(\Xi, \Xi_1, \Xi_2, N)}{\Xi \vdash \Theta \Uparrow \cdot}$$

$$\hookrightarrow$$

$$\frac{\Xi_1 : \Gamma \Uparrow \vdash \lceil \neg N \rceil^- \Uparrow \quad \Xi_2 : \Gamma \Uparrow \lceil \neg N \rceil^- \vdash \Uparrow R \quad Cut_e(\Xi, \Xi_1, \Xi_2, \lceil \neg N \rceil^-)}{\Xi : \Gamma \Uparrow \vdash \Uparrow R}$$

$$\frac{\Xi_1 \vdash \Theta \Uparrow P \quad \Xi_2 \vdash \Theta \Uparrow \neg P \quad cut_e(\Xi, \Xi_1, \Xi_2, P)}{\Xi \vdash \Theta \Uparrow \cdot}$$

$$\hookrightarrow$$

$$\frac{\Xi_2 : \Gamma \Uparrow \vdash \lceil P \rceil^- \Uparrow \quad \Xi_1 : \Gamma \Uparrow \lceil P \rceil^- \vdash \Uparrow R \quad Cut_e(\Xi, \Xi_1, \Xi_2, \lceil P \rceil^-)}{\Xi : \Gamma \Uparrow \vdash \Uparrow R}$$

Figure 8.3: Cut Embedding

the *augmented* cut rule of LKF^a can be mapped to two LJF^a augmented cut rules, depending on the polarity of B, as seen in figure 8.3. Indeed, the cut formula in the corresponding LJF cut must be an implication (the negative translation $\lceil \cdot \rceil^-$ of a positive formula). If N is a negative formula and P is a positive formula, then interfacing the cut agents of the two systems is:

$$^{LJF}\mathbb{C}_e(\Xi, \Xi_1, \Xi_2, \lceil \neg N \rceil^-) \ \ \texttt{:-} \ \ {}^{LKF}\mathscr{C}_e(\Xi, \Xi_1, \Xi_2, N).$$
$$^{LJF}\mathbb{C}_e(\Xi, \Xi_2, \Xi_1, \lceil P \rceil^-) \ \ \texttt{:-} \ \ {}^{LKF}\mathscr{C}_e(\Xi, \Xi_1, \Xi_2, P).$$

8.2 Transducing proof certificates

Throughout the previous chapters, the trade-off is apparent in several places between the amount of information contained in a proof certificate and the necessity of search to reconstruct a proof. This section discusses the varying amount of details in proof certificates by focusing in particular on both possible extremes: on the one hand the least information that must be given by a proof certificate to avoid non-termination; on the other hand the sufficient amount of information that makes a proof certificate detailed enough to be functionally checkable, *i.e.,*non-reliant on proof search. This section then discusses ways of obtaining these two extreme proof certificates from other proof certificates: obtaining the detailed proof certificate is done through an *elaborating transducer* while obtaining the least detailed proof certificate is done

through a *forgetful transducer*. The transducers considered here are used on LKF^a proof certificates, but the same points hold for LJF^a proof certificates.

8.2.1 Elaborating transducer

The term *transduction* comes from automata theory where it designates a state machine with two tapes: an input and an output tape. The term is used here for a generic p.r.i.c.e., used to elaborate any proof certificate into a fully detailed proof certificate. More precisely, because the elaborating transduction is used to supply all details of a proof, three p.r.i.c.e. definitions are discussed in this section:

- an input p.r.i.c.e. previously defined, as usual, for a given format and used to guide the kernel to a proof,
- an output p.r.i.c.e., defined as a standard p.r.i.c.e. for fully functional proof certificates and
- the transduction p.r.i.c.e., that uses the input p.r.i.c.e. to obtain a proof and constructs the output p.r.i.c.e..

The input certificate does not influence the definition of the transduction and is, therefore, not the main topic. The output certificate will have the case constructors shown in figure 8.4. They record all information in a tree skeleton that will be isomorphic to the derivation generated by the proof checking process.

The polarity assignment is fixed by the input certificate and the transducer and of the output p.r.i.c.e. comply with it. The agents of the output p.r.i.c.e. are straightforwardly defined to unpack the input case and pass on the continuation case. For instance, the decide expert is defined as:

$$\mathscr{D}_e((\dddot{\mathscr{D}}\ I\ \Xi), \Xi, I)$$

The index of the output (and the transducer) p.r.i.c.e. cannot depend solely on the index of the input p.r.i.c.e.. Indeed, if the former is to be completely defined so as to leave no room for backtracking, the indexes have to be functional (*i.e.,*no two formulas in the storage share the same index). To this end, the indexes of the are formed between the indexes given by the original certificate and an integer that must be distinct for each stored formula in order for the resulting elaborated p.r.i.c.e. to be functional.

$$\texttt{type pid} \quad (\mathbb{N} * \xi) \rightarrow \xi.$$

type	$\dddot{\wedge}^{-}$	case $\rightarrow$ case $\rightarrow$ case.
type	$\dddot{\vee}^{-}$	case $\rightarrow$ case.
type	$\dddot{\wedge}^{+}$	case $\rightarrow$ case $\rightarrow$ case.
type	$\dddot{\vee}^{+}$	$\delta \rightarrow$ case $\rightarrow$ case.
type	$\dddot{\exists}$	$\iota \rightarrow$ case $\rightarrow$ case.
type	$\dddot{\forall}$	$(\iota \rightarrow$ case$) \rightarrow$ case.
type	$\dddot{t}^{+}$	case.
type	$\dddot{t}^{-}$	case $\rightarrow$ case.
type	$\dddot{\mathscr{S}}$	$\xi \rightarrow$ case $\rightarrow$ case.
type	$\dddot{\mathscr{D}}$	$\xi \rightarrow$ case $\rightarrow$ case.
type	$\dddot{\mathscr{R}}$	case $\rightarrow$ case.
type	$\dddot{\mathscr{I}}$	$\xi \rightarrow$ case.
type	$\dddot{\mathscr{C}}$	fm $\rightarrow$ case $\rightarrow$ case $\rightarrow$ case.

Figure 8.4: The case constructors for the elaborating transducer

To complete the transducer $\texttt{p.r.i.c.e.}$ definition (the polarity assignment and the indexing are described above), it remains to define the case constructor and the agents. The only case constructor is defined as follows:

$$\texttt{type}\quad\texttt{infix}\quad\blacktriangleright\qquad\texttt{case}\to\texttt{case}\to\mathbb{N}\to\texttt{case}.$$

where the first parameter is the input case which is used to guide the kernel towards a proof, the second parameter is the output case which records all steps taken by the kernel, and the third parameter is a counter used to generate fresh indexes to store formulas.

The team of agents defined on the $\blacktriangleright$ case is given in figure 8.5. To save space, the counter c is written above the left arrow $\blacktriangleright$, as such: $\Xi_i \overset{c}{\blacktriangleright} \Xi_o$. It is only used in the store clerk definition (marked by *).

8.2.2 Forgetful transducer

As mentioned in remark 4.1.2, the only rule susceptible of causing non-termination is the decide rule. The least information to give is thus a bound on the proof search in the form of a decide depth (see in definition 2.2.2). A decide depth can be enforced in many ways: an integer counter decremented at each decide and prevented from reaching zero, a list of indexes that are removed one by one every time a decide is done on them, or any other data structure from which one can obtain a finitely decreasing measure. One can even use the following `case` constructor as case:

$$\texttt{type}\quad\texttt{dd}\quad\mathbb{N}\to\texttt{case}.$$

and define the decide expert to be

$$\mathscr{D}_e(\texttt{dd}\,C, \texttt{dd}\,(C-1), _)\quad\texttt{:-}\quad C > 0.$$

while all other agent definitions are naïve.

Following the same constructor as for the elaborating transducer, one can design a *forgetful* transducer. This transducer also acts on a pair of cases and only record the conjunctions structure and the decide rules. Only the following case constructors are used:

$$\texttt{type}\quad\texttt{infix}\quad\triangleright\qquad\texttt{case}\to\texttt{case}\to\mathbb{N}\to\texttt{case}.$$

which is the case on which the forgetful transducing team is defined,

$$\texttt{type}\quad\clubsuit\quad\texttt{case}.$$

$$\wedge_c^- \,(\Xi \overset{c}{\blacktriangleright} (\overset{\cdots}{\wedge}{}^{-}\; \Xi_1'\, \Xi_2'), \Xi_1 \overset{c}{\blacktriangleright} \Xi_1', \Xi_2 \overset{c}{\blacktriangleright} \Xi_2') \quad \texttt{:-} \quad \wedge_c^- \,(\Xi, \Xi_1, \Xi_2).$$

$$\vee_c^- \,(\Xi \overset{c}{\blacktriangleright} (\overset{\cdots}{\vee}{}^{-}\; \Xi'), \Xi_o \overset{c}{\blacktriangleright} \Xi') \quad \texttt{:-} \quad \vee_c^- \,(\Xi, \Xi_o).$$

$$\wedge_e^+ \,(\Xi \overset{c}{\blacktriangleright} (\overset{\cdots}{\wedge}{}^{+}\; \Xi_1'\, \Xi_2'), \Xi_1 \overset{c}{\blacktriangleright} \Xi_1', \Xi_2 \overset{c}{\blacktriangleright} \Xi_2') \quad \texttt{:-} \quad \wedge_e^+ \,(\Xi, \Xi_1, \Xi_2).$$

$$\vee_e^+ \,(\Xi \overset{c}{\blacktriangleright} (\overset{\cdots}{\vee}{}^{+}\; D\, \Xi'), \Xi_o \overset{c}{\blacktriangleright} \Xi', D) \quad \texttt{:-} \quad \vee_e^+ \,(\Xi, \Xi_o, D).$$

$$\exists_e(\Xi \overset{c}{\blacktriangleright} (\overset{\cdots}{\exists}\; W\, \Xi'), \Xi_o \overset{c}{\blacktriangleright} \Xi', W) \quad \texttt{:-} \quad \exists_e(\Xi, \Xi_o, W).$$

$$\forall_c(\Xi \overset{c}{\blacktriangleright} (\overset{\cdots}{\forall}\, \Xi'), \lambda y.(\Xi_o\, y) \overset{c}{\blacktriangleright} (\Xi'\, y)) \quad \texttt{:-} \quad \forall_c(\Xi, \Xi_o).$$

$$t_e^+(\Xi \overset{c}{\blacktriangleright} \overset{\cdots}{t}{}^{+}\,) \quad \texttt{:-} \quad t_e^+(\Xi).$$

$$f_c^-(\Xi \overset{c}{\blacktriangleright} (\overset{\cdots}{f}{}^{-}\, \Xi'), \Xi_o \overset{c}{\blacktriangleright} \Xi') \quad \texttt{:-} \quad f_c^-(\Xi, \Xi_o).$$

$$\mathscr{R}_e(\Xi \overset{c}{\blacktriangleright} (\overset{\cdots}{\mathscr{R}}\, \Xi'), \Xi_o \overset{c}{\blacktriangleright} \Xi') \quad \texttt{:-} \quad \mathscr{R}_e(\Xi, \Xi_o).$$

$$\mathscr{C}_e(\Xi \overset{c}{\blacktriangleright} (\overset{\cdots}{\mathscr{C}}\, F\, \Xi_1'\, \Xi_2'), \Xi_1 \overset{c}{\blacktriangleright} \Xi_1', \Xi_2 \overset{c}{\blacktriangleright} \Xi_2', F) \quad \texttt{:-} \quad \mathscr{C}_e(\Xi, \Xi_1, \Xi_2, F).$$

$$\mathscr{S}_c(\Xi \overset{c}{\blacktriangleright} (\overset{\cdots}{\mathscr{S}}\, \texttt{pid}\langle c, I\rangle\, \Xi'), F, \Xi_o \overset{c+1}{\blacktriangleright} \Xi', \texttt{pid}\langle c, I\rangle) \quad \texttt{:-} \quad \mathscr{S}_c(\Xi, F, \Xi_o, I).^\star$$

$$\mathscr{D}_e(\Xi \overset{c}{\blacktriangleright} (\overset{\cdots}{\mathscr{D}}\, \texttt{pid}\langle J, I\rangle\, \Xi'), \Xi_o \overset{c}{\blacktriangleright} \Xi', \texttt{pid}\langle J, I\rangle) \quad \texttt{:-} \quad \mathscr{D}_e(\Xi, \Xi_o, I).$$

$$\mathscr{I}_e(\Xi \overset{c}{\blacktriangleright} (\overset{\cdots}{\mathscr{I}}\, \texttt{pid}\langle J, I\rangle), \texttt{pid}\langle J, I\rangle) \quad \texttt{:-} \quad \mathscr{I}_e(\Xi, I).$$

Figure 8.5: Team of agents for the elaborating transduction

$$
\begin{array}{l}
\wedge_c^- (\Xi \rhd (\ddot{\wedge}\ \Xi_1'\, \Xi_2'), \Xi_1 \rhd \Xi_1', \Xi_2 \rhd \Xi_2') \quad \texttt{:-} \quad \wedge_c^- (\Xi, \Xi_1, \Xi_2).^\star \\
\vee_c^- (\Xi \rhd \Xi', \Xi_o \rhd \Xi') \quad \texttt{:-} \quad \vee_c^- (\Xi, \Xi_o). \\
\wedge_e^+ (\Xi \rhd (\ddot{\wedge}\ \Xi_1'\, \Xi_2'), \Xi_1 \rhd \Xi_1', \Xi_2 \rhd \Xi_2') \quad \texttt{:-} \quad \wedge_e^+ (\Xi, \Xi_1, \Xi_2).^\star \\
\vee_e^+ (\Xi \rhd \Xi', \Xi_o \rhd \Xi', D) \quad \texttt{:-} \quad \vee_e^+ (\Xi, \Xi_o, D). \\
\exists_e(\Xi \rhd \Xi', \Xi_o \rhd \Xi', W) \quad \texttt{:-} \quad \exists_e(\Xi, \Xi_o, W). \\
\forall_c(\Xi \rhd \Xi', \lambda y.(\Xi_o\, y) \rhd \Xi') \quad \texttt{:-} \quad \forall_c(\Xi, \Xi_o). \\
t_e^+(\Xi \rhd \clubsuit) \quad \texttt{:-} \quad t_e^+(\Xi).^\star \\
f_c^-(\Xi \rhd \Xi', \Xi_o \rhd \Xi') \quad \texttt{:-} \quad f_c^-(\Xi, \Xi_o). \\
\mathscr{R}_e(\Xi \rhd \Xi', \Xi_o \rhd \Xi') \quad \texttt{:-} \quad \mathscr{R}_e(\Xi, \Xi_o). \\
\mathscr{S}_c(\Xi \rhd \Xi', F, \Xi_o \rhd \Xi', I) \quad \texttt{:-} \quad \mathscr{S}_c(\Xi, F, \Xi_o, I). \\
\mathscr{D}_e(\Xi \rhd (\nabla \Xi'), \Xi_o \rhd \Xi', I) \quad \texttt{:-} \quad \mathscr{D}_e(\Xi, \Xi_o, I).^\star \\
\mathscr{I}_e(\Xi \rhd \clubsuit, I) \quad \texttt{:-} \quad \mathscr{I}_e(\Xi, I).
\end{array}
$$

Figure 8.6: Team of agents for the forgetful transduction

which is used as a leaf case at the initial,

$$\texttt{type} \quad \ddot{\wedge} \quad \texttt{case} \to \texttt{case} \to \texttt{case}.$$

and

$$\texttt{type} \quad \nabla \quad \texttt{case} \to \texttt{case}.$$

to mark the decide rule locations. No particular index is needed, it is left unspecified. The team of agents is shown in figure 8.6. Notice that the cut expert is excluded from the team. The agents marked with a star $\star$ are the only non-naïve ones: the ones that finish the proof fix the case to the leaf $\clubsuit$, the decide expert records that there has been a decide, and the conjunctions agents keep the structure of the tree. After the transduction, a simple program can count the number of ∇ in the resulting case tree and create a `dd` case.

Customized transducers

The amount of recorded information can be adjusted to one's needs. For example, if one wants only to keep only the instantiation terms, the forgetful `p.r.i.c.e.` can be

modified to fit this goal. Notice that the output of a forgetful transducer will, most likely, fail to be fully functional.

8.3 Cooperating proof certificates

The fact that the framework ensures soundness regardless of how the semantics is defined allows for great flexibility in the design of p.r.i.c.e.. One can even define a (meta) p.r.i.c.e. that links together multiple p.r.i.c.e. definitions. This way, one can link a possibly non-terminating decision procedure with a p.r.i.c.e. that has a decide depth for only information. One can even link two (or more) incomplete p.r.i.c.e. definitions and use their cooperation to guide proof checking or to generate a complete proof certificate using a transducer p.r.i.c.e..

Consider the following case constructor:

$$\texttt{type} \quad \texttt{concur} \quad \texttt{case} \to \texttt{case} \to \texttt{case}.$$

This case holds a pair of cases, each of them presumably defined on its own team of agents. The purpose of the `concur` team is to make sure that *both* teams defined on the cases listed in the argument of the `concur` case *agree* on at least one set of guidance to give to the kernel, thus on one reconstructed proof. To do so, every agent from the `concur` team calls on its counterparts defined on both cases listed as arguments of the `concur` case. For example, a disjunction expert is defined as:

$$\vee_e^+(\texttt{concur}\,\Xi_1\,\Xi_2, \texttt{concur}\,\Xi_1'\,\Xi_2', D) \quad \texttt{:-} \quad \vee_e^+\,(\Xi_1, \Xi_1', D), \vee_e^+(\Xi_2, \Xi_2', D).$$

The index constructor must be formed of the pair of indexes given by the two proof certificates. It is defined as:

$$\texttt{type} \quad \texttt{pid} \quad (\xi * \xi) \to \xi.$$

and the store clerk is defined as:

$$\mathscr{S}_c(\texttt{concur}\,\Xi_1\,\Xi_2, F, \texttt{concur}\,\Xi_1'\,\Xi_2', \langle I_1, I_2\rangle) \quad \texttt{:-} \quad \mathscr{S}_c(\Xi_1, F, \Xi_1', I_1), \mathscr{S}_c(\Xi_2, F, \Xi_2', I_2).$$

The rest of the team of agents is given figure 8.7.

The `concur` team can check what can be seen as trace equivalence between different definitions of the semantics of the same language, or even semantic comparison of different languages for particular proof evidence. One can also compare more than two p.r.i.c.e. definitions by using pairs of pairs, *e.g.,* $(\texttt{concur}(\texttt{concur}(\Xi_1\,\Xi_2))\,\Xi_3)$. Finally, the `concur` p.r.i.c.e. can be used alongside the transducer to generate a complete proof certificate from possibly incomplete ones.

$$\wedge_c^- \,(\texttt{concur}\,\Xi_1\,\Xi_2, \texttt{concur}\,\Xi_1'\,\Xi_2', \texttt{concur}\,\Xi_1''\,\Xi_2'') \quad \texttt{:-} \quad \wedge_c^- \,(\Xi_1, \Xi_1', \Xi_1''), \wedge_c^-(\Xi_2, \Xi_2', \Xi_2'').$$
$$\vee_c^- \,(\texttt{concur}\,\Xi_1\,\Xi_2, \texttt{concur}\,\Xi_1'\,\Xi_2') \quad \texttt{:-} \quad \vee_c^- \,(\Xi_1, \Xi_1'), \vee_c^-(\Xi_2, \Xi_2').$$
$$\wedge_e^+ \,(\texttt{concur}\,\Xi_1\,\Xi_2, \texttt{concur}\,\Xi_1'\,\Xi_2', \texttt{concur}\,\Xi_1''\,\Xi_2'') \quad \texttt{:-} \quad \wedge_e^+ \,(\Xi_1, \Xi_1', \Xi_1''), \wedge_e^+(\Xi_2, \Xi_2', \Xi_2'').$$
$$\vee_e^+ \,(\texttt{concur}\,\Xi_1\,\Xi_2, \texttt{concur}\,\Xi_1'\,\Xi_2', D) \quad \texttt{:-} \quad \vee_e^+ \,(\Xi_1, \Xi_1', D), \vee_e^+(\Xi_2, \Xi_2', D).$$
$$\exists_e(\texttt{concur}\,\Xi_1\,\Xi_2, \texttt{concur}\,\Xi_1'\,\Xi_2', W) \quad \texttt{:-} \quad \exists_e(\Xi_1, \Xi_1', W), \exists_e(\Xi_2, \Xi_2', W).$$
$$\forall_c(\texttt{concur}\,\Xi_1\,\Xi_2, \lambda y.\texttt{concur}\,(\Xi_1' y)\,(\Xi_2' y)) \quad \texttt{:-} \quad \forall_c(\Xi_1, \Xi_1'), \forall_c(\Xi_2, \Xi_2').$$
$$t_e^+(\texttt{concur}\,\Xi_1\,\Xi_2) \quad \texttt{:-} \quad t_e^+(\Xi_1), t_e^+(\Xi_2).$$
$$f_c^-(\texttt{concur}\,\Xi_1\,\Xi_2, \texttt{concur}\,\Xi_1'\,\Xi_2') \quad \texttt{:-} \quad f_c^-(\Xi_1, \Xi_1'), f_c^-(\Xi_2, \Xi_2').$$
$$\mathscr{R}_e(\texttt{concur}\,\Xi_1\,\Xi_2, \texttt{concur}\,\Xi_1'\,\Xi_2') \quad \texttt{:-} \quad \mathscr{R}_e(\Xi_1, \Xi_1'), \mathscr{R}_e(\Xi_2, \Xi_2').$$
$$\mathscr{C}_e(\texttt{concur}\,\Xi_1\,\Xi_2, \texttt{concur}\,\Xi_1'\,\Xi_2', \texttt{concur}\,\Xi_1''\,\Xi_2'', F) \quad \texttt{:-} \quad \mathscr{C}_e(\Xi_1, \Xi_1', \Xi_1'', F), \mathscr{C}_e(\Xi_2, \Xi_2', \Xi_2'', F).$$
$$\mathscr{S}_c(\texttt{concur}\,\Xi_1\,\Xi_2, F, \texttt{concur}\,\Xi_1'\,\Xi_2', \texttt{pid}\langle I_1, I_2\rangle) \quad \texttt{:-} \quad \mathscr{S}_c(\Xi_1, F, \Xi_1', I_1), \mathscr{S}_c(\Xi_2, F, \Xi_2', I_2).$$
$$\mathscr{D}_e(\texttt{concur}\,\Xi_1\,\Xi_2, \texttt{concur}\,\Xi_1'\,\Xi_2', \texttt{pid}\langle I_1, I_2\rangle) \quad \texttt{:-} \quad \mathscr{D}_e(\Xi_1, \Xi_1', I_1), \mathscr{D}_e(\Xi_2, \Xi_2', I_2).$$
$$\mathscr{I}_e(\texttt{concur}\,\Xi_1\,\Xi_2, \texttt{pid}\langle I_1, I_2\rangle) \quad \texttt{:-} \quad \mathscr{I}_e(\Xi_1, I_1), \mathscr{I}_e(\Xi_2, I_2).$$

Figure 8.7: Team of agents for the cooperating p.r.i.c.e.

Chapter 9

Conclusion, related and future work

One of the objectives of this thesis is to promote the idea that "feature zero" of all provers should be the ability to communicate independently checkable outputs. Obstacles to this feature include the need to translate into some theory-motivated language and the size of the outputs. Using the Foundational Proof Certification framework, the outputs are described rather than translated and parts of them can be omitted, allowing for more compact objects.

By focusing on the semantics definition of proof languages, rather than the proof objects, a broad spectrum of proof evidence formats (of which some were shown in this thesis) can be formally defined independent of the technology that produced them, proof languages can know the same mathematical definition as that brought to programming languages by structured operational semantics. A relational setting facilitates the support for modularity.

The Foundational Proof Certification framework is now being generalized to proofs involving inductive and co-inductive definitions [Heath and Miller, 2015]. With that extension, it should be possible to check proof evidence coming from model checkers[1] and inductive theorem provers. Energy is also spent in extending the Foundational Proof Certification framework to modal logics, starting by defining a focused setting in these logics[2]

Proofs freed from the technologies that produced them take us one step closer to the formal eternity that proof theorists have been seeking for a century.

---

[1] `http://slimmer.gforge.inria.fr/bedwyr/pcmc/`

[2] Submitted draft of the paper "Focused labeled proof systems for modal logic" by Miller and Volpe: `http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/focused-modal.pdf`

9.1 Future work

Being the first effort in the multi-year project ProofCert, this thesis offers many directions in which to take the next steps. For example, to what extent can one handle partial proofs or offer a satisfactory treatment of counterexamples? [Miller, 2014, Section 4.1].

However, it may be premature to tackle these longer-term goals of the ProofCert project. Instead, this section focuses on the immediate next steps to be taken.

Completeness proof of embedding and assisting semantics specification

Recall the embeddings of expansion trees in sequent calculus (sections 5.4.2 and 5.4.2). The argument for the embeddings has the following form "if a node N in the expansion tree appears with a formula F in the sequent then the rule R is applied. The resulting premises are formed with the subformulas of F and the subnodes of N and are embedded in the same way". Compare this to the definition of agents (in the order in which they appear in a sequentialization). These definitions have the form "the agent relative to the rule R is defined on the case C, containing the node N, gives back some output cases with subnodes of N that are used by the next agent to be called".

One notices the similarity between the two processes: the one describing the embedding of a proof format in focused sequent calculus, and the one describing the definition of agents that guide the kernel towards proofs of that form. The same observation holds for the rest of the case studies in this thesis. It seems that, if one proves the completeness of an embedding of a proof format in focused sequent calculus, then one can extract a relational specification in the same way an ML program can be extracted from Coq proofs.

In light of this, even without enforcing the rigors of a completeness proof, one should be able to design a "semantics specifying assistant". Such a tool could be used by someone who is unfamiliar with focusing or logic programming. It should be possible to supply several useful options, such as a polarity inference mechanism that assigns polarity to a connective depending on how the user choses to treat that connective. It can also have an indicator of the current size of the generated `p.r.i.c.e.`, and a measure on choice points generated so that one can try to reduce proof search.

More independence

Section 8.1.3 showed that, while a logic (classical or intuitionistic) is chosen for checking a proof certificate, the particular kernel (based on LKF^a or on LJF^a with

the $\lceil \cdot \rceil^{\pm}$ embedding) needs not be specified. Similar hosting of kernels were tried (though not formally proved) for LKF^a and LJF^a in fragments of an augmented version of *LKU*. One can also explore the *LKF* fragment of *LKU* inside the *LJF* fragment of *LKU* using an embedding similar to $\lceil \cdot \rceil^{\pm}$. The feasability of this is not controversial, the goal of these investigations is to identify exactly what is the minimal common notion of relational specification. In particular, can the same p.r.i.c.e. written for an LKF^a kernel be used, with no change, in kernels based on *LJF*, the *LKF* and *LJF* fragments of *LKU*, or even some two-sided *LKF* kernel.

More implementations

At this stage, several kernels (based on *LKU*, *LKF* and *LJF*) exist in the Teyjus[3] implementation of λProlog[4] language. An implementation in Bedwyr[5] also exists in the work by Heath and Miller [2015], this time based on focused linear logic with fixed points. The Foundational Proof Certification framework was implemented again (in the span of a week-end[6]) in Ocaml[7] and used in the work by Brock-Nannestad and Chaudhuri [2015].

By minimizing the knowledge base needed to reimplement the framework (section 3.2), one hopes that more and more implementations will be made, bringing new ideas to improve the framework. Comparing different implementations can also help find the best technology when the time comes to have an official release of a proof checker.

Libraries

When describing the G4ip decision procedure [Dyckhoff, 1992] using the Foundational Proof Certification framework, an initial investigation in the use of lemmas (not included in this manuscript) was carried out. A new type of formulas, called *generic*, was introduced.

Detailing the use of lemmas is an indispensable step for any real-world application of this framework. Several questions remain unanswered: how can the libraries be browsed? In theory, a URL can serve as an index, for example, and instead of deciding on a formula in the storage, one can decide on a previously certified formula from some trusted library. What exactly should be stored in the libraries? Should

[3] `http://teyjus.cs.umn.edu/`
[4] `http://www.lix.polytechnique.fr/~dale/lProlog/`
[5] `http://slimmer.gforge.inria.fr/bedwyr/`
[6] Conversation with the implementers.
[7] `http://caml.inria.fr/ocaml/`

the theorems be polarized or unpolarized (this would require adding mechanisms to assign polarities). How can the safety of the libraries be guaranteed?

Interaction

The kernel guidance protocol defined through the agents can also be used to interact with a user, or even with an outside program. To what extant can a proof certificate interactively gain information from a user? Can someone rely on focusing to design a similar framework for theorem proving instead of certificate checking?

Interactive theorem proving based on a focused framework is not, strictly speaking, in the scope of this thesis. However, by investigating such possibilities, ideas may appear that can also be applied to proof-certificate checking.

More formats

Handling more proof formats is a natural continuation of this thesis. Some formats are still challenging. For example, the SAT solving community uses three formats routinely: DRAT, DRUP and TRACECHECK. While the TRACECHECK format, which essentially describes a resolution refutation, is naturally mapped to sequent calculus, DRAT and DRUP are not so straightforwardly mapped. Nonetheless, further investigation may lead to a support for these formats, with the use of cut rules and more sophisticated definitions of agents (so far, the agents were defined using very simple, often atomic, Horn clauses).

These difficulties require a clear definition of what constitutes acceptable proof evidence. One such definition was suggested in a recent paper [Chihani et al., 2015].

Deployment

Recently, the organizers of the CADE ATP System Competition (or CASC[8]) changed the design and procedures to "explicitly encourage" systems to produce solutions in TPTP format, which, in the sense of this thesis, is a step forward in advocating for explicitly and independently proof checking.

This competition (currently lacking foundational checking) can be a good place to start a concrete application of the ideas in this thesis. Part of the output of one participating prover (E prover[9]) was addressed [Chihani et al., 2015].

[8] `http://www.cs.miami.edu/~tptp/CASC/`
[9] `http://www.eprover.org/`

9.2 Related work

Interfacing provers, sharing proofs and certifying theorems are the goals of many efforts. Some of them, centered around proof assistants and/or fully automated provers, are briefly presented in this section.

Logosphere

Logosphere[10] has the goal of sharing development of digital libraries of formal proofs between different proof assistants in a foundational logic-independant framework, built on Twelf. Unlike this thesis, the role of the "message" aspect of the proofs is central to Logosphere as it aims at sharing *semantic* mathematical logic. Additionally, the actual proofs of theorems are not intended to be translated in the context of this thesis.

Dedukti

Dedukti is a proof checker based on the $\lambda\Pi$-calculus modulo rewriting [Boespflug et al., 2012]. Cousineau and Dowek [2007] showed that the $\lambda\Pi$-calculus modulo rewriting can encode all pure type systems. Dedukti can check proofs both from proof assistants (*e.g.,* Coq and HOL) and from automated provers (*e.g.,* Zenon and iProver), by first translating them in the $\lambda\Pi$-calculus modulo. Several specialized translators were built since the beginning of the project including Holide for HOL Light, Coqine for Coq, Zenonide for Zenon, Focalide for FoCaLiZe and a plugin to iProver[11].

It is premature to compare the proof translation effort in the setting of Dedukti and the proof language description possibilities of the Foundational Proof Certification framework. However, one lasting difference is the fourth desideratum of the ProofCert project, namely proof reconstruction capabilities.

Other differences are foundational: where Dedukti relies on type theory, a functional setting and intuitionistic logic, ProofCert relies on proof theory, a relational setting, and both classical and intuitionistic logics. (Dedukti is able to check classical proofs with the use of axioms).

[10] Logosphere's URL (logosphere.org) redirects to the Delphin [Poswolsky and Schürmann, 2008] project's website, the language that was successfully used to express translations between various logics, in the context of Logosphere.

[11] All of which are present at the following page `https://www.rocq.inria.fr/deducteam/software.html`

TPTP and TSTP

The TPTP (Thousands of Problems for Theorem Provers) library is a browsable repository of test problems varying in size and complexity. The main goal of the TPTP library is to support the evaluation of automated theorem provers.

TSTP (Thousands of Solutions from Theorem Provers) is the solution library and is "the flip-side of the TPTP"[Sutcliffe, 2007]. It is intended as a useful resource, storing solutions in a standard format so that users can browse and understand them with the goal of improving their own prover. A checking mechanism, called GDV, applies structural and semantics verification. However, this checker relies on automated theorem provers, those same tools whose output this thesis proposes to check.

LF and LFSC

Although the first logical framework was de Bruijn's Automath [de Bruijn, 1970], the term LF usually designates the Edinburgh Logical framework. In their papers, Harper et al. [1987] describe LF as a suitable basis for "logic-independent proof development environment". The overall aims of LF and of Automath are similar, and the former can be seen as "carrying forward the aims" of the latter [Harper et al., 1987]. Both of them are based on type theory and λ-calculus.

LFSC is an extension of LF with *computational side conditions*, allowing "some parts of a proof to be established by computation" [Stump et al., 2013]. Although LFSC is mainly used as a proof checker for SMT solvers, the same ideas can be applied to other logics. The undeniable gain in efficiency is the main incentive behind this effort, but it comes at the cost of allowing possibly non-checked decision procedure in the "proof systems trusted computing base". Using an unverified decision procedure, said Milner [MacKenzie, 2001, p.295], is "like selling your soul to the Devil – you get this enormous power, but what have you lost? You've lost proof, in some sense". However, the functional programs of LFSC are usually small and readable and, if efficiency is paramount, perhaps one can formally verify them. In any case, for the purpose of presenting a framework, it would have been premature for this thesis to sacrifice soundness.

But soundness may be only one of the differences. Consider this (shortened) quote of Stump [2008]: "To encode binary propositional resolution with factoring in pure LF, we must insist that each resolution inference comes with a proof of a side condition showing that C_1 and C_2 resolve as just described to give the resolvent, which the inference proves. That proof may be a trace of the computation of the resolvent, or perhaps evidence based on a more declarative view of the relationship

between the resolvent and C_1 and C_2. But there is no obvious way to reduce its size from $\mathrm{O}(|C_1| + |C_2|)$ to a constant. And hence, the size of resolution proofs will be completely dominated by the size of the proofs of their side conditions".

By using the Foundational Proof Certification framework, however, the size of the proof evidence for a binary resolution inference *can* be reduced to a constant (modulo the size of the resolvent clause). The size of resolution step to check is two integer numbers (indexes of the clauses), and the size of the reconstructed proof (that does not need to be communicated) is an *LKF* proof with a maximal decide depth of 3. This can scale easily to hyperresolution, where the size of the proof evidence for the resolution step is as many integer numbers as there are clauses to resolve.

Higher-order proof translation

Sultana [2015] investigated different approaches to proof translation for higher-order logic. His thesis offers survey of many translation efforts and compares them along three axes. *Importer* translators, located close to the source prover, *exporter* translators, close to the target prover, and *transducer* translator which is agnostic with regard to the internal states of the target and the source provers. He also proposes a promising compiler-like framework for translation and applies it to some provers.

His thesis is rather implementation-oriented, technology-dependent and focuses on higher-order logic, whereas the present thesis is still at the theoretical stage, independent of technology and focuses on first-order logics.

Bibliography

Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.

Peter B. Andrews. General models, descriptions, and choice in type theory. *Journal of Symbolic Logic*, 37(2):385–394, 1972.

Peter B. Andrews. Theorem proving via general matings. *J. ACM*, 28(2):193–214, 1981. doi: 10.1145/322248.322249.

Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Kluwer Academic Publishers, second edition, 2002.

Robert L. Ashenhurst. Acm forum. In *Letters to the editors*, volume 22, pages 621–630, New York, NY, USA, November 1979. ACM. doi: 10.1145/359168.359177. URL `http://doi.acm.org/10.1145/359168.359177`.

Andrea Asperti. Proof, message and certificate. In Johan Jeuring, John A. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge, editors, *Intelligent Computer Mathematics - Proceedings of AISC, DML, and MKM 2012*, volume 7362 of *LNCS*, pages 17–31. Springer, 2012. doi: 10.1007/978-3-642-31374-5.

David Baelde. Least and greatest fixed points in linear logic. *ACM Trans. on Computational Logic*, 13(1), April 2012. doi: 10.1145/2071368.2071370. URL `http://tocl.acm.org/accepted/427baelde.pdf`.

H. Barendregt and F. Wiedijk. The challenge of computer mathematics. *Transactions A of the Royal Society*, 363(1835):2351–2375, October 2005.

Henk Barendregt. The impact of the lambda calculus in logic and computer science. *Bulletin of Symbolic Logic*, 3(2):181–215, 1997.

Henk P. Barendregt. Introduction to generalized type systems. *Journal of Functional Programming*, 1(2):125–154, April 1991.

Jon Barwise. Mathematical proofs of computer system correctness. In *Notices of the American Mathematical Society, 36*, 1989.

Christoph Benzmüller and Dale Miller. Automation of higher-order logic. In J. Siekmann, editor, *Logic and Computation*, volume 9 of *Handbook of the History of Logic*, pages 215–254. North Holland, 2014. ISBN 978-0-444-51624-4.

W. Bibel. *Automated Theorem Proving*. Vieweg Verlag, Braunschweig, second edition edition, 1987.

Mathieu Boespflug, Quentin Carbonneaux, and Olivier Hermant. The λΠ-calculus modulo as a universal proof language. In David Pichardie and Tjark Weber, editors, *Proceedings of PxTP2012: Proof Exchange for Theorem Proving*, pages 28–43, 2012.

Taus Brock-Nannestad and Kaustuv Chaudhuri. Disproving using the inverse method by iterated refinement of finite approximations. In Hans de Nivelle, editor, *Proceedings of the 24th Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, volume 9323 of *LNAI*, Wrocław, Poland, September 2015. Springer. URL `http://chaudhuri.info/papers/draft15saturate.pdf`.

Kaustuv Chaudhuri. Classical and intuitionistic subexponential logics are equally expressive. In Anuj Dawar and Helmut Veith, editors, *CSL 2010: Computer Science Logic*, volume 6247 of *LNCS*, pages 185–199, Brno, Czech Republic, August 2010. Springer. doi: 10.1007/978-3-642-15205-4_17. URL `http://hal.archives-ouvertes.fr/inria-00534865/en/`.

Kaustuv Chaudhuri, Dale Miller, and Alexis Saurin. Canonical sequent proofs via multi-focusing. In G. Ausiello, J. Karhumäki, G. Mauri, and L. Ong, editors, *Fifth International Conference on Theoretical Computer Science*, volume 273 of *IFIP*, pages 383–396. Springer, September 2008.

Kaustuv Chaudhuri, Stefan Hetzl, and Dale Miller. A multi-focused proof system isomorphic to expansion proofs. *J. of Logic and Computation*, June 2014. doi: 10.1093/logcom/exu030. URL `http://hal.inria.fr/hal-00937056`.

Zakaria Chihani, Dale Miller, and Fabien Renaud. Foundational proof certificates in first-order logic. In Maria Paola Bonacina, editor, *CADE 24: Conference on Automated Deduction 2013*, number 7898 in LNAI, pages 162–177, 2013a.

Zakaria Chihani, Dale Miller, and Fabien Renaud. Checking foundational proof certificates for first-order logic (extended abstract). In J. C. Blanchette and J. Urban, editors, *Third International Workshop on Proof Exchange for Theorem Proving (PxTP 2013)*, volume 14 of *EPiC Series*, pages 58–66. EasyChair, 2013b.

Zakaria Chihani, Tomer Libal, and Giselle Reis. The proof certifier checkers. In *TABLEAUX 24: Automated Reasoning with Analytic Tableaux and Related Methods*, number 9323 in LNAI, 2015.

Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:354–363, 1936.

Alonzo Church. A formulation of the Simple Theory of Types. *J. of Symbolic Logic*, 5:56–68, 1940.

Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4583 of *LNCS*, pages 102–117. Springer, 2007.

Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960. ISSN 0004-5411. doi: 10.1145/321033.321034. URL `http://doi.acm.org/10.1145/321033.321034`.

N. G. de Bruijn. The mathematical language AUTOMATH, its usage, and some of its extensions. In *Symposium on Automatic Demonstration*, pages 29–61. Lecture Notes in Mathematics, 125, Springer, 1970.

Nicolaas Govert de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with an application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34(5):381–392, 1972.

R. Dyckhoff and S. Lengrand. LJQ: a strongly focused calculus for intuitionistic logic. In A. Beckmann and *et al.*, editors, *Computability in Europe 2006*, volume 3988 of *LNCS*, pages 173–185. Springer, 2006.

Roy Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *J. of Symbolic Logic*, 57(3):795–807, September 1992.

Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, 1990.

Gerhard Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1935. Translation of articles that appeared in 1934-35. Collected papers appeared in 1969.

Gerhard Gentzen. Die widerspruchfreiheit der reinen zahlentheorie. *Mathematische Annalen*, 112:493–565, 1936. Reprinted in English translation as "The consistency of Elementary Number Theory" in The collected papers of Gerhard Gentzen, M. E. Szabo, ed.

Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

Jean-Yves Girard. A fixpoint theorem in linear logic. An email posting to the mailing list linear@cs.stanford.edu, February 1992.

Jean-Yves Girard. *Le Point Aveugle: Cours de logique: Tome 1, Vers la perfection*. Hermann, 2006.

Kurt Gödel. Zur intuitionistischen arithmetik und zahlentheorie. *Ergebnisse eines Mathematischen Kolloquiums*, pages 34–38, 1932. English translation in *The Undecidable* (M. Davis, ed.) 1965, 75-81.

Kurt Gödel. Eine interpretation des intuitionistischen aussagenkalkuls. *Ergebnisse eines Mathematischen Kolloquiums.*, 4:39–40, 1933. Available in "Kurt Gödel: Collected Works. Volume 1" edited by S. Feferman and et al.

Robert Harper and Frank Pfenning. On equivalence and canonical forms in the LF type theory. *ACM Trans. Comput. Log.*, 6(1):61–101, 2005. doi: 10.1145/1042038.1042041. URL `http://doi.acm.org/10.1145/1042038.1042041`.

Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. In *2nd Symp. on Logic in Computer Science*, pages 194–204, Ithaca, NY, June 1987.

Quentin Heath and Dale Miller. A framework for proof certificates in finite state exploration. In Cezary Kaliszyk and Andrei Paskevich, editors, Proceedings Fourth Workshop on *Proof eXchange for Theorem Proving,* Berlin, Germany, August 2-3, 2015, volume 186 of *Electronic Proceedings in Theoretical Computer Science*, pages 11–26. Open Publishing Association, 2015. doi: 10.4204/EPTCS.186.4.

Hugo Herbelin. A λ-calculus structure isomorphic to gentzen-style sequent calculus structure. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer Berlin Heidelberg, 1995a. ISBN 978-3-540-60017-6. doi: 10.1007/BFb0022247. URL http://dx.doi.org/10.1007/BFb0022247.

Hugo Herbelin. *Séquents qu'on calcule: de l'interprétation du calcul des séquents comme calcul de lambda-termes et comme calcul de stratégies gagnantes*. PhD thesis, Université Paris 7, 1995b.

Jacques Herbrand. *Recherches sur la Théorie de la Démonstration*. PhD thesis, University of Paris, 1930.

Stefan Hetzl and Daniel Weller. Expansion trees with cut. *CoRR*, abs/1308.0428, 2013.

Alfred Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16:14–21, 3 1951. ISSN 1943-5886. doi: 10.2307/2268661. URL http://journals.cambridge.org/article_S0022481200102385.

Stephen Cole Kleene. *Introduction to Metamathematics*. North-Holland, Amsterdam, 1952.

Andrei Nikolaevich Kolmogorov. On the principle of the excluded middle. *Matematicheskii sbornik*, 32:646–667, 1925. English translation by Jean van Heijenoort in *From Frege to Gödel*.

Boris Konev and Alexei Lisitsa. A sat attack on the erds discrepancy conjecture. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 219–226. Springer International Publishing, 2014. ISBN 978-3-319-09283-6. doi: 10.1007/978-3-319-09284-3_17. URL http://dx.doi.org/10.1007/978-3-319-09284-3_17.

Konstantin Korovin. Inst-gen a modular approach to instantiation-based automated reasoning. In Andrei Voronkov and Christoph Weidenbach, editors, *Programming Logics*, volume 7797 of *Lecture Notes in Computer Science*, pages 239–270. Springer Berlin Heidelberg, 2013.

S.G. Krantz. *The Proof is in the Pudding: The Changing Nature of Mathematical Proof*. SpringerLink : Bücher. Springer, 2011. ISBN 9780387487441. URL http://books.google.fr/books?id=mMZBtxVZiQoC.

Olivier Laurent. *Etude de la polarisation en logique*. PhD thesis, Université Aix-Marseille II, March 2002.

Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009. doi: 10.1016/j.tcs.2009.07.041.

Chuck Liang and Dale Miller. A focused approach to combining logics. *Annals of Pure and Applied Logic*, 162(9):679–697, 2011. doi: 10.1016/j.apal.2011.01.012.

Dana Mackenzie. What in the name of euclid is going on here? *Science*, 307 (5714):1402–1403, 2005. doi: 10.1126/science.307.5714.1402a. URL http://www.sciencemag.org/content/307/5714/1402.1.short.

Donald MacKenzie. *Mechanizing Proof*. MIT Press, 2001.

S. Maehara. Eine darstellung der intuitionistischen logik in der klassischen. *Nagoya Mathematical Journal*, pages 45–64, 1954.

Ursula Martin. Stumbling around in the dark: Lessons from everyday mathematics. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, volume 9195 of *Lecture Notes in Computer Science*, pages 29–51. Springer International Publishing, 2015. ISBN 978-3-319-21400-9. doi: 10.1007/978-3-319-21401-6_2. URL http://dx.doi.org/10.1007/978-3-319-21401-6_2.

Mathematicians. The qed manifesto. In *Proceedings of the 12th International Conference on Automated Deduction*, CADE-12, pages 238–251, London, UK, UK, 1994. Springer-Verlag. ISBN 3-540-58156-1. URL http://dl.acm.org/citation.cfm?id=648231.752823.

Dale Miller. A compact representation of proofs. *Studia Logica*, 46(4):347–370, 1987.

Dale Miller. Abstract syntax for variable binders: An overview. In John Lloyd and *et al.*, editors, *CL 2000: Computational Logic*, number 1861 in LNAI, pages 239–253. Springer, 2000. URL http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/cl2000.pdf.

Dale Miller. Proofcert: Broad spectrum proof certificates. An ERC Advanced Grant funded for the five years 2012-2016, February 2011. URL http://www.lix.polytechnique.fr/Labo/Dale.Miller/ProofCert.pdf.

Dale Miller. Communicating and trusting proofs: The case for broad spectrum proof certificates. In *Logic, Methodology, and Philosophy of Science. Proceedings of the Fourteenth International Congress*, 2014.

Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

Richard A. De Millo, Richard J. Lipton, and Alan J. Perlis. Social processes and proofs of theorems and programs. *Communications of the Association of Computing Machinery*, 22(5):271–280, May 1979.

Sara Negri and Jan von Plato. *Structural Proof Theory*. Cambridge University Press, 2001.

Vivek Nigam and Dale Miller. A framework for proof systems. *J. of Automated Reasoning*, 45(2):157–188, 2010. URL `http://springerlink.com/content/m12014474287n423/`.

David A. Plaisted. History and prospects for first-order automated deduction. In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, pages 3–28, 2015. doi: 10.1007/978-3-319-21401-6_1. URL `http://dx.doi.org/10.1007/978-3-319-21401-6_1`.

Gordon Plotkin. A structural approach to operational semantics. DAIMI FN-19, Aarhus University, Aarhus, Denmark, September 1981.

Adam Poswolsky and Carsten Schürmann. System description: Delphin - A functional programming language for deductive systems. In A. Abel and C. Urban, editors, *International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2008)*, volume 228, pages 113–120, 2008.

Dag Prawitz. *Natural Deduction*. Almqvist & Wiksell, Uppsala, 1965.

Michael O. Rabin. Probabilistic algorithms. *Algorithms and Complexity: New Directions and Recent Results*, pages 21–40, September 1976.

G Robinson and L Wos. Paramodulation and theorem-proving in first-order theories with equality. In *Automation of Reasoning*, pages 298–313. Springer, 1983.

J. A. Robinson. Automatic deduction with hyper-resolution. *International journal of computer mathematics*, 1:227–234, 1965a. ISSN 1943-5886.

J. A. Robinson. A machine-oriented logic based on the resolution principle. *JACM*, 12:23–41, January 1965b.

Andrew L. Russell. *Open Standards and the Digital Age: History, Ideology, and Networks*. Cambridge University Press, New York, NY, USA, 2014. ISBN 1107612047, 9781107612044.

Peter Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *8th Symp. on Logic in Computer Science*, pages 222–232. IEEE Computer Society Press, IEEE, June 1993.

Robert J. Simmons. Structural focalization. *ACM Trans. Comput. Log.*, 15(3):21, 2014. doi: 10.1145/2629678. URL `http://doi.acm.org/10.1145/2629678`.

Aaron Stump. Proof checking technology for satisfiability modulo theories. In A. Abel and C. Urban, editors, *Logical Frameworks and Meta-Languages: Theory and Practice*, 2008.

Aaron Stump, Duckki Oe, Andrew Reynolds, Liana Hadarean, and Cesare Tinelli. Smt proof checking using a logical framework. *Formal Methods in System Design*, 42(1):91–118, 2013. ISSN 0925-9856. doi: 10.1007/s10703-012-0163-3. URL `http://dx.doi.org/10.1007/s10703-012-0163-3`.

Nikolai Sultana. Higher-order proof translation. Technical Report UCAM-CL-TR-867, University of Cambridge, Computer Laboratory, April 2015. URL `http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-867.pdf`.

Geoff Sutcliffe. Tptp, tstp, casc, etc. In Volker Diekert, MikhailV. Volkov, and Andrei Voronkov, editors, *Computer Science Theory and Applications*, volume 4649 of *Lecture Notes in Computer Science*, pages 6–22. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-74509-9. doi: 10.1007/978-3-540-74510-5_4. URL `http://dx.doi.org/10.1007/978-3-540-74510-5_4`.

Gaisi Takeuti. *Proof Theory*. North Holland, 2nd edition, 1987.

A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 2 edition, 2000.

Hao Wang. Proving theorems by pattern recognition i. *Commun. ACM*, 3(4):220–234, April 1960. ISSN 0001-0782. doi: 10.1145/367177.367224. URL http://doi.acm.org/10.1145/367177.367224.

Freek Wiedijk. The qed manifesto revisited. *Studies in Logic, Grammar and Rhetoric*, 10(23):121–133, 2007.

Doron Zeilberger and GeorgeE. Andrews. Theorems for a price: tomorrows semi-rigorous mathematical culture. *The Mathematical Intelligencer*, 16(4):11–18, 1994. ISSN 0343-6993. doi: 10.1007/BF03024696. URL http://dx.doi.org/10.1007/BF03024696.