Deep Inference for Graphical Theorem Proving

Pablo Donato

March 9, 2024

Abstract

Proof assistants are software systems that allow for the precise checking of mathematical reasoning. They can be general purpose (like Coq, Lean, Isabelle...) or more specialized like EasyCrypt. They enable a level of accuracy which certifies that no error can occur, have given birth to a wide range of practical and theoretical works, and have been used in a wide range of applications. But they remain difficult to use.

We propose a new paradigm of formal proof construction through actions performed in a graphical user interface, to enable a more comfortable and intuitive use. Our paradigm builds upon direct manipulation principles, combining both old (Proof-by-Pointing) and new (Proof-by-Linking) interaction techniques that exploit recent advances in deep inference proof theory. We implement this paradigm in a prototype of graphical user interface called Actema, using modern web-based technologies. We also design a generic protocol for plugging Actema on any proof system that supports first-order intuitionistic logic. This protocol is deployed inside the Coq proof assistant through the coq-actema plugin, offering users the ability to integrate graphical proofs into existing textual developments.

Then, driven by the will to improve the various interaction techniques of Actema in a unified formalism, we explore a series of deep inference proof systems that give more structure to the notion of logical goal. These systems share the ability to represent goals in two alternative ways: either textually through a standard inductive syntax, or graphically through a metaphorical notation well-suited to direct manipulation.

The first family of systems, called bubble calculi, is an extension of the theory of nested sequents, that we reframe as local rewriting systems with a graphical and topological interpretation. Bubble calculi enable an efficient sharing of contexts between subgoals, making them well-suited to the factorization of both forward and backward reasoning steps in proofs. The second system, called flower calculus, is an intuitionistic refinement of C.S. Peirce's theory of existential graphs, understood as a system for interactive, goal-directed proof building. It provides more iconic and economical means of reasoning than bubble calculi, by exposing a small number of expressive rules that apply to the goals themselves, removing the need for logical connectives. Both types of systems are shown to be analytic and fully invertible, making them amenable to proof automation techniques.

We finally go back to practical experimentation by designing and implementing the Flower Prover, another web-based prototype of GUI for interactive proof building based on the flower calculus. An innovative feature of the Flower Prover is that it works well on modern mobile devices, thanks to its responsive layout and first-class support for touch interactions.

Contents

Contents iii

1. Introduction 1
1.1. Proof theory . . . 2
1.2. Proof assistants . . . 10
1.3. This thesis . . . 15
1.4. Related works . . . 22

Symbolic Manipulations 24

2. Proof-by-Action 25
2.1. Logical setting . . . 25
2.2. A first example . . . 26
2.3. Proof steps through clicks . . . 28
2.4. A simple example involving equality . . . 30
2.5. Drag-and-dropping through connectives . . . 31
2.6. Related works . . . 35

3. Subformula Linking 38
3.1. Linkages . . . 38
3.2. Validity . . . 40
3.3. Describing DnD actions . . . 43
3.4. Soundness . . . 44
3.5. Productivity . . . 47
3.6. Focusing . . . 51
3.7. Completeness . . . 52
3.8. Related works . . . 58

4. Proof-by-Action in Practice 60
4.1. Forward reasoning . . . 61
4.2. Sets and functions . . . 65
4.3. Peano arithmetic . . . 71

5. Parallel Conclusions and Classical Logic 75
5.1. Backtracking . . . 76
5.2. Implementation in theorem provers . . . 76
5.3. Click actions . . . 77
5.4. DnD actions . . . 78
5.5. Metatheory of parallel reasoning . . . 79

6. Integration in a Proof Assistant 82
6.1. Actema . . . 83
6.2. Why a plugin? . . . 84
6.3. The coq-actema system . . . 84
6.4. Interaction protocol . . . 87
6.5. Compiling actions . . . 94

6.6. Future works . . . 97

Iconic Manipulations 102

7. Asymmetric Bubble Calculus 103
7.1. The chemical metaphor . . . 103
7.2. Bubbles and solutions . . . 104
7.3. Asymmetric calculus . . . 106
7.4. Back to Proof-by-Action . . . 113

8. Symmetric Bubble Calculi 116
8.1. Non-determinism and iconicity . . . 117
8.2. Symmetric interpretation . . . 118
8.3. Coloring bubbles . . . 121
8.4. Designing for properties . . . 124
8.5. Symmetric calculus . . . 127
8.6. Soundness . . . 133
8.7. Completeness . . . 143
8.8. Invertible calculus . . . 148

9. Existential Graphs 155
9.1. Alpha graphs . . . 156
9.2. Illative transformations . . . 158
9.3. Graphs as multisets . . . 161
9.4. Illative atomicity . . . 163
9.5. Soundness . . . 167
9.6. Completeness . . . 169
9.7. Beta graphs . . . 176
9.8. Gardens . . . 183

10. Flower Calculus 187
10.1. Intuitionistic existential graphs . . . 188
10.2. Flowers . . . 194
10.3. Calculus . . . 198
10.4. Kripke semantics . . . 209
10.5. Soundness . . . 211
10.6. Completeness . . . 217
10.7. Automated proof search . . . 223
10.8. The Flower Prover . . . 231
10.9. Conclusion . . . 240

Appendix 247

A. Symmetric Bubble Calculi 248
A.1. Soundness . . . 248
A.2. Completeness . . . 255

Bibliography 260

Concept Index 275

Acronym Index **280**

Proof System Index **281**

Software System Index **282**

List of Figures

1.1. Natural deduction calculus NJ for intuitionistic logic 6
1.2. Rule of indirect proof in natural deduction 6
1.4. Sequent calculus LJ for intuitionistic logic 7
1.3. Proof of the law of non-contradiction in natural deduction 7
1.5. Proof of the law of non-contradiction in sequent calculus 8
1.6. Dependency graph between chapters 21

2.1. A partial screenshot showing a goal in the Actema prototype 26
2.2. Variants of some sequent calculus rules 29
2.3. Proving $1 + 1 = 2$ in Peano arithmetic 31
2.4. Linking rules 37
2.5. Unit elimination rules 37

3.1. Release rules 40
3.2. Linkage formation rules 53
3.3. Resource rules 53
3.4. Duplicating linkage formation rules 53
3.5. Reflexivity rule for $=$ 54
3.6. Non-analytic reflexivity rules 54

4.1. The beginning of an example due to Edukera 61
4.2. Coq proof script formalizing Edukera's riddle 64
4.3. Preliminary definitions in Coq of an exercise on abstract functions 69
4.4. Solution in Coq to the first question of an exercise on abstract functions 70
4.5. Solution in Coq to the second question of an exercise on abstract functions 70
4.6. Proof of commutativity of addition on natural numbers in Coq 74
5.1. Multiplicative right introduction rules for disjunction and absurdity 75
5.2. Proof of the excluded middle in LK 75

5.3. Multi-conclusion right introduction rules for implication 77
5.4. Multi-conclusion instantiation rules for quantifiers 78
5.5. Parallel linking rules 79
5.6. Alternating structure between reasoning modes 79

6.1. Architecture of the coq-actema system 85
6.2. Graphical layout of the coq-actema system 86
6.3. Sequence diagram of coq-actema's interaction protocol — non-interactive mode 88
6.4. Sequence diagram of coq-actema's interaction protocol — breaking out of the interaction loop 89
6.5. Sequence diagram of coq-actema's interaction protocol — applying an action 90
6.6. ATD definitions for first-order formulas and environments 92
6.7. ATD definitions for goals 93
6.8. ATD definitions for actions 95
6.9. Soundness theorems of DnD fixpoints in Coq 97

7.1. Context-scoping in bubbles as topological constraints 105
7.2. Sequent-style presentation of the asymmetric bubble calculus BJ 107
7.3. Example of sequent-style proof in BJ 108

7.4. Graphical presentation of the asymmetric bubble calculus BJ . . . 111
7.5. Example of graphical proof in BJ . . . 112
7.6. Linkage creation rules in BJ . . . 113

8.1. Multi-conclusion right introduction rule for conjunction . . . 118
8.2. Distributing conclusions in bubbles . . . 118
8.3. Classical multi-conclusion version of ⊃+ . . . 121
8.4. $\mathbb{F}$-rules for red bubbles . . . 121
8.5. Proof attempts for Grishin (a) and Grishin (b) . . . 122
8.6. $\mathbb{F}$-rules for blue bubbles . . . 123
8.7. $\mathbb{H}$-rules for exclusion ⊂ . . . 123
8.8. Sequent-style presentation of system B . . . 128
8.9. Graphical presentation of system B . . . 129
8.10. Porosity of bubbles in system B . . . 131
8.11. A proof of Uustalu's formula in system B . . . 132
8.12. Relationship between the various algebras interpreting system B . . . 133
8.13. Rules of the deep nested sequent system DBiInt . . . 145
8.14. Proof of DNE in system B . . . 146
8.15. Rules for the invertible bubble calculus $\mathsf{B}_{\mathsf{inv}}$ . . . 149
8.16. Mapping of formulas to equivalent solutions . . . 154
8.17. Left introduction rule for ⊃ in JN . . . 154

9.1. Peirce's notation for emphasizing negative areas . . . 158
9.2. Drawing negative areas literally in negative . . . 158
9.3. A derivation of Peirce's law in Alpha . . . 160
9.4. Inductive presentation of the rules of Alpha . . . 162
9.5. Graphical representation of the four conditions of local justification . . . 165
9.6. The illatively atomic system $\mathsf{Alpha}^{\mathsf{a}}$ . . . 166
9.7. Inference rules of SKS . . . 171
9.8. Universal quantification $\forall x.P(x)$ in Beta . . . 178
9.9. Implication $A \supset B$ in Alpha . . . 179
9.10. Decomposing lines of identity . . . 179
9.11. Building a two-ended wire from two teridentities . . . 179
9.12. A proof of a famous syllogism in Beta . . . 182
9.13. Using variables in EGs . . . 183
9.14. A depiction of Peirce's Bridge for lines of identity . . . 184
9.15. Inductive presentation of the rules of Beta . . . 185
9.16. A proof in the inductive syntax of Beta . . . 185

10.1. Peirce's scroll . . . 189
10.2. Peirce's scroll with a blank antecedant . . . 189
10.3. The rule of Blank Antecedant . . . 189
10.4. Currying as scroll nesting . . . 189
10.5. Intuitionistic proof of currying . . . 190
10.6. Intuitionistic proof of uncurrying . . . 190
10.7. A curl with five loops . . . 191
10.8. Continuity, disjunction and implication in IEGs . . . 191
10.9. Formula interpretation of the n-ary scroll . . . 192
10.10. Turning a 5-ary scroll inside-out . . . 194
10.11. Nested flowers . . . 194
10.12. Pollination in flowers . . . 199

10.13. Converse of epis rule 199
10.14. Rules of the flower calculus 202
10.15. Cross-reproduction rule 205
10.16. Graphical presentation of the natural rules 206
10.17. Graphical presentation of the cultural rules 207
10.18. A natural proof in the flower calculus 208
10.19. The syntax-semantics mirror 210
10.20. Life traces for *modus ponens* (left) and identity expansion of disjunction (right) 227
10.21. Translation $(-)^\bullet$ of formulas into bouquets 228
10.22. Life trace for doubly-negated double-negation elimination law 228
10.23. Testing dataset of tautologies from `Edukera` 230
10.24. Proof mode (left) and Edit mode (right) of the `Flower Prover` 233
10.25. A sequence of Proof actions in the `Flower Prover` 239
10.26. (De)iteration rules for petals 240
10.27. Simulating the srep rule by iterating petals 240

List of Tables

6.1. Protocol for applying an action in Actema 94

10.1. Notational conventions for meta-variables 196
10.2. Fragments of intuitionistic logic as cardinality constraints on flowers 201
10.3. Graphical actions of the `Flower Prover` 238

Introduction 1.

1.1 **Proof theory** 2
1.1.1 Mathematical logic 2
1.1.2 Structural proof theory . 5
1.2 **Proof assistants** 10
1.2.1 Logical frameworks . . . 11
1.2.2 Interfaces 12
1.3 **This thesis** 15
1.3.1 Research goals 15
1.3.2 Contributions 18
1.3.3 How to read 20
1.4 **Related works** 22

> The ultimate meaning of logic is this ability to manipulate.
>
> **Jean-Yves Girard**, *The blind spot*, 2011

Proof assistants (PAs) — also called *interactive theorem provers* (ITPs) — are software systems that allow to both create and check the correctness of mathematical proofs. They are based on the idea that mathematical knowledge can be represented unambiguously inside *proof formalisms* — also called *proof systems*, where the truth of a statement can be reduced to the mechanical application of symbolic manipulation rules. For instance, consider the equation

$$4x + 6x = (12 - 2)x$$

While any mathematician would immediately recognize it as true, a middle school student learning algebra would have to carry manually some computations to convince herself (and her teacher) of its validity. A first step might consist in applying the distributivity of multiplication over addition on the left-hand side of the equation, yielding the new equation

$$(4 + 6)x = (12 - 2)x$$

Then, computing the sum on the left-hand side and the difference on the right-hand side gives the final equation

$$10x = 10x$$

which is trivially true. This is a very simple example, but it already shows the two main aspects of proof formalisms: on the one hand, they allow to *represent* mathematical statements in a formal language, here that of equations between linear univariate polynomials; on the other hand, they allow to *manipulate* this representation in order to prove the statements, here through *rewriting rules* that transform a valid equation into another valid equation.

Algebra lends itself particularly well to formalization, as it is arguably the very study of the rules governing symbolic manipulations in mathematics. It also heavily relies on computations, which explains why it was the target of the first, and to this day most popular application of computers to mathematics: computer algebra systems.

However in this thesis, we are interested in improving the usability of proof assistants, which have a much broader scope than computer algebra systems: their ambition is to enable the formalization on computers of virtually *any* kind of mathematics. Ultimately, the dream is to provide a platform that helps humans in creating *new* mathematics: both novel solutions and proofs to existing problems, and brand new theories involving new types of mathematical objects. This seemingly disproportionate ambition is not entirely utopic: it is based on the great discoveries of 19$^{\text{th}}$ and 20$^{\text{th}}$ century mathematicians and logicians, in the broad research area now known as *mathematical logic*.

1.1. Proof theory

1.1.1. Mathematical logic

Universal language At the dawn of the 20$^{\text{th}}$ century, some mathematicians started to realize that it may be possible to formalize not only specific branches of mathematics like algebra with their own language, but the *whole* of mathematics in a single, universal language. This idea was first intuited in the 17$^{\text{th}}$ century by Leibniz with his dream of a *characteristica universalis*, an ideal language in which all propositions — mathematical propositions, but also scientific propositions about the real world, and even metaphysical propositions — could be expressed and understood unambiguously by every human. Also, Leibniz introduced the concept of a *calculus ratiocinator*, a systematic method for determining the truth of any proposition expressed in the *characteristica universalis*, providing a definitive and objective way to settle any argument through simple calculations[1].

1: Leibniz himself might have been inspired by his predecessor Galileo, who famously declared that "the universe [...] is written in the language of mathematics" [84].

Predicate logic and set theory The possibility of a universal language for mathematics became credible at the dusk of the 19$^{\text{th}}$ century, thanks to the works of logicians like Boole, Frege and Peirce on one hand [24, 80, 199], and mathematicians like Cantor and Dedekind on the other hand [133]. The former laid the groundwork for a formal account of deduction that greatly improved on Aristotle's syllogistic, by inventing notations and rules that can express reasoning about not only *properties* of individuals, but also *relations* between them. The latter invented *set theory*, which provided the first setting where a general notion of *function* or mapping could be rigorously defined, a notion that became increasingly central in modern mathematics.

[24]: Boole (1854), *The Laws of Thought*
[80]: Frege (1879), *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*
[199]: Peirce (1885), 'On the Algebra of Logic'
[133]: Johnson (1972), 'The Genesis and Development of Set Theory'

Foundations This formed the basis for a unification of many branches of mathematics on the same *foundation*: it was realized that with enough effort, every mathematical structure could be encoded with the sets of Cantor, and all the laws governing sets could be expressed with a finite number of axioms expressed in *predicate logic*, i.e. the language and calculus of relations devised by 19$^{\text{th}}$ century logicians. This crystallized into two famous axiomatic systems for set theory: the *Principia Mathematica* of Russell and Whitehead [249]; and Zermelo-Fraenkel set theory (ZF, or ZFC with the axiom of choice), which is the most popular foundation nowadays because of its greater simplicity.

[249]: Whitehead et al. (1925), *Principia Mathematica*

Truth and proofs An *axiomatic system* specifies the formal language in which statements about mathematical objects are expressed, as well as a collection of such statements — the *axioms* — that are taken to be true from the outset, without further justification. One does not even need to speak about *truth* to define the system: although it can be a guiding intuition when designing the system, the fact that axioms denote true properties of abstract objects in some "mathematical universe" is a particular philosophical stance (platonism), which has nothing to do with concrete reasoning on the formal representation.

Traditionally, the branch of mathematical logic that tries to model the "semantic" content of axioms through the notion of truth is called *model theory*. In this thesis, we are concerned with the construction of formal proofs that derive the consequences of axioms by pure "syntactic" manipulation, through the application of so-called *inference rules*. Accordingly, the branch of mathematical logic studying this activity is called *proof theory*. We will still do a bit of model theory in a few places (Section 8.6, Section 10.6), but only as a means to justify the properties of our syntax. Thus to avoid any unnecessary philosophical commitment, we will only consider axioms of a given system as ordinary *assumptions* that can be used in the course of reasoning, without according any particular status to their truth. This is very much in line with the *formalist* school of thought in philosophy of mathematics, represented by the great mathematician and main instigator of proof theory David Hilbert.

> The real focus throughout this thesis is on the *inference rules* used to build (correct) proofs from axioms/assumptions. Those form the theoretical basis for both the *interactive creation*, and the *automatic checking* of formal proofs in proof assistants. The branch of proof theory concerned with the study of inference rules is called *structural proof theory*.

Axiomatic systems In the very beginnings of proof theory in the 1920s, under the influence of Hilbert, the axiomatic method was predominant, and thus proof systems of this era — now called *Hilbert systems* — featured very few inference rules. Almost all logical reasoning principles were encoded as *axiom schemas* involving generic *propositional variables*. For instance, the famous *law of excluded middle* (LEM), that states that every proposition is either true or false, is expressed formally by the schema

$$A \vee \neg A$$

Here, $\vee$ and $\neg$ are symbols denoting the logical connectives of *disjunction* ("or") and *negation* ("not"), and A is a propositional variable that can be substituted with any concrete *formula* built from *atomic propositions* and logical connectives. An atomic proposition is typically a property of a mathematical object, that does not involve any logical connective. An example of *instance* of this schema would be the proposition "n is prime or n is not prime" with n some natural number, which can be written formally as

$$\mathrm{prime}(n) \vee \neg\mathrm{prime}(n)$$

Another related principle is the *law of non-contradiction*, which states that no proposition can be both true and false at the same time. It is expressed by the schema

$$\neg(A \wedge \neg A)$$

where $\land$ is the symbol denoting *conjunction* (“and”).

Intuitionistic logic One motivating factor in the development of a new foundation for mathematics was the discovery of strange theorems that defy intuition, like the existence of the Weierstraß function which is continuous everywhere but differentiable nowhere [248], or the Banach-Tarski paradox which asserts that a ball can be decomposed and reassembled into two exact copies of itself [13]. Some mathematicians like Brouwer and Weyl rejected the truth of such theorems, on the basis that their proofs rely on reasoning principles that are not *constructive*[2]. In particular, these principles allow to prove the existence of objects satisfying certain properties without ever providing a *witness*, i.e. a concrete object that satisfies the properties in question. This marked the birth of *constructivism* in philosophy of mathematics, whose most famous incarnation is Brouwer's *intuitionism*.

[248]: Weierstraß (1895), ‘Über continuirliche functionen eines reellen arguments, die für keinen werth des letzteren einen bestimmten differentialquotienten besitzen’

[13]: Banach et al. (1924), ‘Sur la décomposition des ensembles de points en parties respectivement congruentes’

2: This is true for the Banach-Tarski paradox, which relies crucially on non-constructive definitions of the concepts of partitions and equivalence classes [180]. But for the Weierstraß function, it is possible to give it a constructive definition with some efforts [16].

The original intuitionism of Brouwer was strongly opposed to any attempt at formalizing mathematics, standing against both Frege and Russell's logicism that saw mathematics as a mere branch of logic, and Hilbert's formalism that reduced mathematics to a game of symbol manipulation. However, this did not prevent Heyting, one of Brouwer's students, from developing an axiomatic system in the style of Hilbert and Frege, in an attempt to capture formally the objections of Brouwer towards *classical* logic — i.e. the logic developed by 19^{th} century logicians that was at the heart of the new set-theoretical foundations. Heyting's system captures what is now called *intuitionistic* logic, which can be succinctly summarized as being exactly classical logic, but *without* the law of excluded middle. Thus intuitionistic logic is a generalization of classical logic, where propositions cannot be assigned a truth value *a priori*: they are only considered true if they can be proved with *direct*, constructive evidence.

To this day, there is no consensus among mathematicians as to which logic — intuitionistic or classical — is the right one to found mathematics upon. Since intuitionistic logic is more restrictive than classical logic, some fundamental theorems of classical mathematics do not hold anymore, requiring in the worst cases to recreate entire branches of mathematics from scratch, like in *constructive analysis*. This explains why a large majority of mathematicians still work in classical logic, and are often even unaware of the existence of constructive mathematics.

> To account for this diversity, in this thesis we design proof systems that support *both* classical and intuitionistic reasoning. Because every theorem of intuitionistic logic is also a theorem of classical logic (but not the converse), we will often focus first on the intuitionistic “kernel” of our systems, designing the classical part as an extension of the former.

1.1.2. Structural proof theory

Inference rules In Hilbert systems, the only inference rule is that of *modus ponens*, which is expressed formally with the following figure:

$$\frac{A \quad A \supset B}{B}\,\mathsf{mp}$$

Like axioms, it is a *schema* that involves generic propositional variables A and B, which may be *instantiated* with arbitrary formulas. It can be read from top to bottom as follows: for any propositions A and B, if we have a proof of A and a proof of $A \supset B$, i.e. a proof that A implies B, then we can immediately derive a proof of B by virtue of the rule, here designated by the abbreviated name mp. This reading of the rule corresponds to a form of *forward* reasoning: starting from the known *premises* that A and $A \supset B$ are true, it *necessarily* follows that the *conclusion* B is true.

Conversely, one can also have a bottom-up reading of the rule: to build a proof of any proposition B, one way to proceed is to come up with another proposition A such that both A and $A \supset B$ are provable. This reading corresponds to a form of *backward* reasoning: we start from the conclusion B that we want to reach, also called the *goal*, and try to find *subgoals* A and $A \supset B$ that are provable, and hopefully simpler to prove; then the rule guarantees that proving these subgoals is *sufficient* to ensure the truth of the original goal.

Forward reasoning is typically how mathematicians write (informal) proofs on paper, for the *presentation* of their proofs to other mathematicians. Indeed, it is more natural for humans to follow an argument by starting from its premises, because the latter will always contain all the information required to deduce the conclusion, the argument only serving as a means to explicate how this information is combined. On the other hand, backward reasoning is more natural during the *construction* phase of a proof, because the information required to reach the conclusion (e.g. the proposition A in the mp rule) is not yet known.

Natural deduction Axiomatic systems can be relatively concise, in that many logics can be expressed in them with a small number of axioms. In return, they produce very long and verbose formal proofs that are hard for humans to follow, and almost impossible to come up with in most cases. In a series of seminars started in 1926, the Polish logician Łukasiewicz became one of the first to advocate for a more *natural* approach in proof theory, that models more closely the way mathematicians actually reason [132]. A few years later, in a dissertation delivered to the faculty of mathematical sciences of the University of Göttingen [88], the German logician Gerhard Gentzen proposed independently his famous calculus of *natural deduction*.

[132]: Jaśkowski (1934), 'On the Rules of Suppositions in Formal Logic'

[88]: Gentzen (1935), 'Untersuchungen über das logische Schließen. I'

This formalism follows the opposite approach to Hilbert systems: it features as few axioms as possible, favoring the use of *inference rules* to model the forms of reasoning found in mathematical practice. Those are divided into two categories: *introduction rules* *define* the meaning of logical connectives, by prescribing how to prove complex formulas from

$$\frac{\bot}{A}\ \bot\mathsf{e} \qquad [1]\ \frac{\begin{array}{c}\Gamma,[A]^1\\ \vdots\\ B\end{array}}{A\supset B}\ \supset\mathsf{i} \qquad \frac{A \quad A\supset B}{B}\ \supset\mathsf{e}$$

$$\frac{A \quad B}{A\wedge B}\ \wedge\mathsf{i} \qquad \frac{A\wedge B}{A}\ \wedge\mathsf{e}_1 \qquad \frac{A\wedge B}{B}\ \wedge\mathsf{e}_2$$

$$\frac{A}{A\vee B}\ \vee\mathsf{i}_1 \qquad \frac{B}{A\vee B}\ \vee\mathsf{i}_2 \qquad [1,2]\ \frac{A\vee B \quad \begin{array}{c}\Gamma,[A]^1\\ \vdots\\ C\end{array} \quad \begin{array}{c}\Delta,[B]^2\\ \vdots\\ C\end{array}}{C}\ \vee\mathsf{e}$$

$$\frac{\begin{array}{c}\Gamma\\ \vdots\\ A\end{array}}{\forall x.A}\ \forall\mathsf{i} \qquad \frac{\forall x.A}{A\{t/x\}}\ \forall\mathsf{e} \qquad \frac{A\{t/x\}}{\exists x.A}\ \exists\mathsf{i} \qquad \frac{\exists x.A \quad \begin{array}{c}\Gamma,A\\ \vdots\\ C\end{array}}{C}\ \exists\mathsf{e}$$

In the rules $\forall\mathsf{i}$ and $\exists\mathsf{e}$, x must not occur free in Γ and C.

Figure 1.1.: Natural deduction calculus NJ for intuitionistic logic

proofs of their components. Dually, *elimination rules* explain how to *use* complex formulas, by giving a canonical way to derive new conclusions from them. Figure 1.1 shows the complete set of natural deduction rules for all connectives and quantifiers in intuitionistic logic, that was introduced by Gentzen under the name NJ[3] [88]. The most simple example can be found in the rules for the conjunction connective $\wedge$: the introduction rule $\wedge\mathsf{i}$ allows to build a proof of $A\wedge B$ by combining a proof of A and a proof of B; while the elimination rules $\wedge\mathsf{e}_1$ and $\wedge\mathsf{e}_2$ allow to derive proofs of A and B from a proof of $A\wedge B$.

3: Gentzen simultaneously introduced a natural deduction calculus named NK for classical logic, which is just NJ with an additional rule modelling the principle of *indirect proof* — i.e. the possibility to prove any proposition A by deriving a contradiction from its negation $\neg A$ (Figure 1.2). Indeed, this principle can be expressed by the formula $\neg\neg A\supset A$ which is strictly equivalent to the law of excluded middle, in the sense that $\neg\neg A\supset A$ is (intuitionistically) provable if and only if $A\vee\neg A$ is.

$$[1]\ \frac{\begin{array}{c}\Gamma,[\neg A]^1\\ \vdots\\ \bot\end{array}}{A}\ \mathsf{ip}$$

Figure 1.2.: Rule of indirect proof in natural deduction

Remark 1.1.1 Note that the rules for negation $\neg$ are not present in Figure 1.1: indeed, it is customary in intuitionistic logic to define negation by $\neg A \triangleq A\supset\bot$, identifying the negation of any proposition A with its implying of a contradiction. Thus the rules for negation are subsumed by those for implication $\supset$ and absurdity $\bot$.

All logical reasoning principles that were *axiomatized* in Hilbert systems can be *derived* in natural deduction. For example, Figure 1.3 shows a proof of the law of non-contradiction, built by *composing* instances of rules from Figure 1.1. The composition of a rule instance $\mathsf{r1}$ with another rule instance $\mathsf{r2}$ simply consists in using the conclusion of $\mathsf{r1}$ as one of the premisses of $\mathsf{r2}$.

Identity

$$\frac{}{A \Rightarrow A}\,\mathsf{ax} \qquad \frac{\Gamma \Rightarrow A \quad \Delta, A \Rightarrow C}{\Gamma, \Delta \Rightarrow C}\,\mathsf{cut}$$

Structural

$$\frac{\Gamma, A, B, \Delta \Rightarrow C}{\Gamma, B, A, \Delta \Rightarrow C}\,\mathsf{xL} \qquad \frac{\Gamma, A, A \Rightarrow C}{\Gamma, A \Rightarrow C}\,\mathsf{cL} \qquad \frac{\Gamma \Rightarrow C}{\Gamma, A \Rightarrow C}\,\mathsf{wL}$$

Logical

$$\frac{}{\Gamma, \bot \Rightarrow C}\,\bot\mathsf{L} \qquad \frac{\Gamma \Rightarrow A \quad \Delta, B \Rightarrow C}{\Gamma, \Delta, A \supset B \Rightarrow C}\,\supset\!\mathsf{L} \qquad \frac{\Gamma, A \Rightarrow B}{\Gamma \Rightarrow A \supset B}\,\supset\!\mathsf{R}$$

$$\frac{\Gamma, A \Rightarrow C}{\Gamma, A \wedge B \Rightarrow C}\,\wedge\mathsf{L}_1 \qquad \frac{\Gamma, B \Rightarrow C}{\Gamma, A \wedge B \Rightarrow C}\,\wedge\mathsf{L}_2 \qquad \frac{\Gamma \Rightarrow A \quad \Delta \Rightarrow B}{\Gamma, \Delta \Rightarrow A \wedge B}\,\wedge\mathsf{R}$$

$$\frac{\Gamma, A \Rightarrow C \quad \Delta, B \Rightarrow C}{\Gamma, \Delta, A \vee B \Rightarrow C}\,\vee\mathsf{L} \qquad \frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A \vee B}\,\vee\mathsf{R}_1 \qquad \frac{\Gamma \Rightarrow B}{\Gamma \Rightarrow A \vee B}\,\vee\mathsf{R}_2$$

$$\frac{\Gamma, A\{t/x\} \Rightarrow C}{\Gamma, \forall x.A \Rightarrow C}\,\forall\mathsf{L} \qquad \frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow \forall x.A}\,\forall\mathsf{R} \qquad \frac{\Gamma, A \Rightarrow C}{\Gamma, \exists x.A \Rightarrow C}\,\exists\mathsf{L} \qquad \frac{\Gamma \Rightarrow A\{t/x\}}{\Gamma \Rightarrow \exists x.A}\,\exists\mathsf{R}$$

In the rules $\forall\mathsf{R}$ and $\exists\mathsf{L}$, x must not occur free in Γ and C.

Figure 1.4.: Sequent calculus LJ for intuitionistic logic

Sequent calculus In addition to the constructivists' objections, some doubts were raised by the discovery of fatal flaws in early attempts at defining foundational axiomatic systems, the most famous one being the *antinomy* of Russell's paradox caused by the unrestricted axiom of comprehension in naive set theory. In order to restore absolute trust in the foundations of (classical) mathematics, Hilbert proposed in the early 1920s to prove mathematically the *consistency* of the axiomatic system for arithmetic introduced in 1889 by Peano[4], i.e. that no contradiction can be derived from Peano's axioms. Indeed, he believed that every mathematical truth could be derived from the principles of arithmetic, thus reducing the problem of the consistency of mathematics to that of arithmetic. Moreover, Hilbert's program was to be carried by *finitist* means, without resorting to any reasoning principle involving infinite collections — which were at the heart of the controversy started by constructivists. This was the initial impulse for developing proof theory, since it provided a mathematical definition of "mathematical proofs" as *finite* sequences of symbols satisfying certain properties.

Gentzen's work on natural deduction was an integral part of this program, as an attempt to render the *metamathematics* of proof theory more structured and elegant. However, he could not devise any argument for consistency in this framework. He thus set out to devise a new formalism that would be a reformulation of natural deduction with better math-

$$\dfrac{\dfrac{[A \wedge \neg A]^1}{A}\,\wedge\mathsf{e}_1 \qquad \dfrac{[A \wedge \neg A]^1}{\neg A}\,\wedge\mathsf{e}_2}{[1]\;\dfrac{\bot}{\neg(A \wedge \neg A)}\,\supset\!\mathsf{i}}\,\supset\!\mathsf{e}$$

Figure 1.3.: Proof of the law of non-contradiction in natural deduction

4: A more involved axiomatic system was proposed one year earlier by Dedekind [58]. Less known is that Peirce had already published in 1881 an equivalent axiomatization of natural numbers [198].

ematical properties, such as *symmetries*. This gave us *sequent calculus*, which is widely regarded as the cornerstone for most developments in proof theory to this day. Gentzen introduced simultaneously *two* sequent calculi: one for classical logic called LK, and one for intuitionistic logic called LJ. Here we focus on the intuitionistic system LJ, whose rules are shown in Figure 1.4.

Sequent calculus is based on the observation that some rules in natural deduction depend crucially on the use of *hypotheses* that appear "higher" or earlier in the proof. The prototypical example is the introduction rule $\supset$i for implication: to prove $A \supset B$, it suffices to prove B under the assumption that A is true. Then the hypothesis A is *discharged* by the rule (bracket notation in Figure 1.1), meaning that the conclusion $A \supset B$ holds *unconditionally*, without the assumption. In sequent calculus, this relation of provability of a conclusion C under a collection/*context* of hypotheses Γ is captured by the expression $\Gamma \Rightarrow C$, called a *sequent*. The introduction rule $\supset$i is then expressed by the so-called *right introduction rule* $\supset$R, which keeps track of the full context of hypotheses by having sequents as premiss and conclusion, instead of just formulas. Right introduction rules for other connectives are also obtained straightforwardly from the corresponding introduction rules in natural deduction, by simply making the contexts Γ and Δ always explicit.

Following the original presentation of Gentzen, contexts are taken to be *lists* of formulas ("Sequenz" in German), i.e. ordered collections where repetitions are allowed. Still, we really want to see them as *sets* of formulas, since it is implicit in mathematical practice that:

1. the *order* in which hypotheses are listed does not matter;
2. hypotheses may be used *more than once* in a proof (as in Figure 1.3).

These two conventions are respectively captured by the *structural rules* xL of *exchange* and cL of *contraction* in Figure 1.4. A third structural rule, wL for *weakening*, accounts for the presence of unused assumptions in some proofs, by allowing the introduction of new hypotheses at will (with a top-down reading of rules).

The main difference between sequent calculus and natural deduction lies in its splitting of elimination rules into two parts: *left introduction rules*, and the *cut* rule cut. As their name indicates, left introduction rules serve a purpose symmetric to right introduction rules: while the latter define how to introduce a connective in the conclusion of a sequent, the former define how to introduce a connective in one of its hypotheses. Then, the only way to *use* such an hypothesis A is through the cut rule, which erases A from the context in the conclusion of the rule, by justifying it with the proof of A given as premiss. The cut rule can also be seen as a generalization of the *modus ponens* rule of Hilbert systems, replacing the logical connective $\supset$ of implication by the "structural connective" $\Rightarrow$ of sequents. The remaining *axiom* rule ax is in a sense dual to the cut rule: while the latter allows justifying a hypothesis by an identical conclusion, the ax rule allows justifying a conclusion by an identical hypothesis[5]. Figure 1.5 shows a proof of the law of non-contradiction in LJ.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \Rightarrow A}\,\mathsf{ax} \qquad \dfrac{}{\bot \Rightarrow \bot}\,\mathsf{ax}}{A, \neg A \Rightarrow \bot}\,{\supset}\mathsf{L}}{A, A \wedge \neg A \Rightarrow \bot}\,{\wedge}\mathsf{L}_2}{A \wedge \neg A, A \wedge \neg A \Rightarrow \bot}\,{\wedge}\mathsf{L}_1}{A \wedge \neg A \Rightarrow \bot}\,\mathsf{cL}}{\Rightarrow \neg(A \wedge \neg A)}\,{\supset}\mathsf{R}$$

Figure 1.5.: Proof of the law of non-contradiction in sequent calculus

5: It is not clear where the terminology "axiom rule" comes from. It might be because hypotheses can be considered as axioms in the sense we introduced earlier, i.e. statements that we *assume* to be true. In any case, it would be a misattribution of category to consider the ax rule itself as an axiom, and some authors like Girard call it the *identity rule* or id rule to avoid the confusion — which also coincides with the concept of *identity morphism* in category theory.

Gentzen managed to prove a powerful result called alternatively *Haupstatz*, *fundamental theorem* (of proof theory), or *cut-elimination*: every provable formula in sequent calculus has a *cut-free* proof, i.e. a proof that does not make use of the cut rule. Intuitively, it can be understood as a formal justification for the possibility to *inline* proofs of lemmas, by seeing an instance of cut on A as a way to invoke the lemma A without duplicating its proof. Moreover, Gentzen's proof of the Haupstatz is itself constructive: it describes an *algorithm* for transforming every sequent calculus proof into a cut-free one. Thus the cut rule is said to be *admissible*, in the sense that any provable sequent can be proved without it.

An important consequence of cut-elimination, which was the original motivation of Gentzen, is the consistency of the logic (intuitionistic predicate logic in the case of LJ). This stems from the fact that all rules apart from the cut rule satisfy the *subformula property*: every formula A appearing in the premisses is a *subformula* of some formula B in the conclusion, i.e. A already occurs inside B. Thus there cannot exist a proof of the absurd sequent $\Rightarrow \bot$, since the only formula that is a subformula of $\bot$ is $\bot$ itself, and there is no rule instance with $\Rightarrow \bot$ as conclusion that only contains $\bot$ in its premisses. The subformula property is the first occurrence of the concept of *analyticity* in proof theory, and can be seen as a technical realization of the philosophical notion of analyticity first applied to propositions by Kant, and later to proofs in mathematics by Bolzano [219].

[219]: Sebestik (1992), *Logique et mathématiques chez Bernard Bolzano*

Unfortunately, the proof of cut-elimination for the sequent calculus incorporating Peano's axioms, found by Gentzen a few years after proving cut-elimination for LJ [89], is not finitist: it makes use of a transfinite induction up to the ordinal ϵ_0. But the very ideas of cut-elimination and analyticity will have far-reaching applications in proof theory and beyond, including many of the results presented in this thesis.

[89]: Gentzen (1936), 'Die Widerspruchsfreiheit der reinen Zahlentheorie'

Deep inference Many years after Gentzen's seminal work, at the advent of the 21th century, Alessio Guglielmi introduced a new methodology for designing proof formalisms called *deep inference* [110]. The idea was to overcome some limitations of Gentzen formalisms while preserving their good properties, by allowing inference rules to be applied *anywhere* inside formulas, instead of only at the top-level of sequents[6]. The first deep inference system was the *calculus of structures* (CoS), which can be succinctly described as a *rewriting system* on formulas. For instance, the following *switch rule*, when read bottom-up (i.e. in backward mode), indicates that the formula $A \vee (B \wedge C)$ may be rewritten into $(A \vee B) \wedge C$:

[110]: Guglielmi (1999), *A Calculus of Order and Interaction*

6: In fact, Schütte had already proposed a deep inference system as early as 1977 [218], as did Peirce one century before him with his *entitative graphs* (as we will see in Chapter 9). But the idea did not generate much interest at the times.

$$\frac{S\boxed{(A \vee B) \wedge C}}{S\boxed{A \vee (B \wedge C)}}\ \mathsf{s}$$

Importantly, the rule can be applied in any *context* $S\square$. A *context* $S\square$ is simply a formula containing a single occurrence of a special subformula $\square$ called its *hole*, which can be *filled* (i.e. substituted) with any formula A to give a new formula $S\boxed{A}$. This notion of context serves two purposes:

- it formalizes the ability of rewriting rules to be applied at an arbitrary *depth* inside expressions, while retaining all the information available in the surrounding context;

- it generalizes the contexts Γ, Δ of sequent calculus, by giving them the full structure of formulas instead of just flat lists of formulas. Indeed, a sequent $\Gamma \Rightarrow C$ can be interpreted as the formula $\bigwedge \Gamma \supset C$, where $\bigwedge \Gamma$ denotes the conjunction of all the formulas in Γ.

Then, a proof of a formula A in the calculus of structures is not a *tree* of rule instances as in natural deduction and sequent calculus, but a *sequence* of rewritings $A \rightarrow^* \top$ that reduces A to the trivially true goal $\top$.

The calculus of structures, as a rewriting system, is closer to the equational reasoning that mathematicians are accustomed to in algebra. The main difference is that most rules (including the switch rule s) can only be applied in a *single* direction, because the premiss and conclusion are not *equivalent*[7]. When the premiss and conclusion of a rule are equivalent, we say that the rule is *invertible*.

7: In the standard system SKS from where the s rule originates, every context is *positive*, in the sense that the hole $\square$ never occurs under a negation $\neg$ or left-hand side of an implication $\supset$; otherwise the rule would need to be restricted to positive contexts to stay valid. See Section 9.6 for more details on SKS.

> *All* the proof formalisms designed in this thesis are deep inference rewriting systems in the style of the calculus of structures.

1.2. Proof assistants

Proof assistants are a direct application of proof theory, exploiting the ability of programming languages to represent and manipulate arbitrary data structures to give a concrete implementation of proof formalisms. Crucially, they open the possibility to *automate* the construction and verification of formal proofs, by acting on two fronts:

- on the **human** side, the design of *high-level interfaces* for representing and manipulating statements and proofs can bridge the gap between the low-level and very detailed proofs of formal logic, and the informal proofs of mathematicians;
- on the **machine** side, the design of *algorithms* that both find proofs of given statements and ensure their correctness can — to some extent[8] — relieve mathematicians from the burdens of proof-writing and proof-checking.

8: For instance, it is well-known that the problem of provability in predicate logic is *undecidable*.

Thus the advent of computers gave a new purpose to proof theory, going beyond its foundational role with the hope to support and change the everyday practice of mathematicians.

> In this thesis, we are concerned mostly with the *human* side of the equation: we aim to provide smoother means for the user to communicate her intent to the proof assistant, and conversely for the proof assistant to communicate its results and suggestions on how to solve problems.

1.2.1. Logical frameworks

Type theory The ancestor of all proof assistants was the `Automath` project, initiated by Nicolaas Govert de Bruijn as soon as 1967. Citing Geuvers [90]:

[90]: Geuvers (2009), 'Proof assistants'

> [One] aim of the project was to develop a mathematical language in which all of mathematics can be expressed accurately, in the sense that linguistic correctness implies mathematical correctness. This language should be computer checkable and it should be helpful in improving the reliability of mathematical results.

Thus the design of `Automath` was focused on the automatic *verification* of proofs through *linguistic* means. It introduced many fundamental ideas that are still at work in many modern proof assistants, the most prominent being the use of a *type theory* to encode formal proofs.

Contrary to predicate logic in traditional proof theory, type theories break the syntactic hierarchy imposed upon mathematical objects, propositions and proofs, by giving them a uniform representation as so-called *terms* that can be assigned a *type*. The assertion that a term t has type T is usually written with the expression $t : T$, which has come to be called a typing *judgment* after Martin-Löf. For instance, the judgment $3 : \mathbb{N}$ states that the term 3 has the type $\mathbb{N}$ of natural numbers, and $1 + 1 = 2 : \mathsf{Prop}$ states that the term $1 + 1 = 2$ has the type Prop of propositions.

First-order vs. higher-order Almost all type theories are *higher-order*: they give a first-class status to functions and predicates, by allowing them to take other functions and predicates as arguments. This is because they are based on the *λ-calculus* of Alonzo Church [47], an intensional theory of higher-order functions that is now considered to be the first functional programming language in history. For instance in *simply-typed* λ-calculus [48], one may type the *sum operator* over sequences of natural numbers with the judgment $\lambda u.\lambda n. \sum_{i=1}^{n} u\, i : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$, where $\lambda u.\lambda n. \sum_{i=1}^{n} u\, i$ is a *λ-term* encoding the higher-order function that takes a sequence represented as a function $u : \mathbb{N} \to \mathbb{N}$ and a bound $n : \mathbb{N}$, and returns the sum of each of u's values at index $1 \leq i \leq n$ encoded as the function application $u\, i$.

[47]: Church (1932), 'A Set of Postulates for the Foundation of Logic'

[48]: Church (1940), 'A Formulation of the Simple Theory of Types'

By contrast, the predicate logic developed in the 19th century and studied in traditional proof theory is *first-order*: functions and predicates can only take so-called *first-order individuals* as arguments, which usually model "non-functional" mathematical objects like numbers and sets.

> In this thesis, we exclusively study proof formalisms for *first-order* predicate logic ("FOL" hereafter).

We identified a few reasons for working in FOL:

- it is a standard and well-understood setting that has received a lot of attention, allowing us to exploit various existing works from the structural proof theory literature, and even from some overlooked theories of 19^{th} century logicians;

- by contrast, type theory is a quite recent subject[9], which explains why there is still a great diversity of type theories that differ in subtle and often incompatible ways;
- FOL is a common kernel of virtually every type theory, making our work directly applicable to all present and future proof assistants;
- it is also a simpler setting, that is powerful enough to study the essential features of logical reasoning, without the idiosyncracies of type theories aimed at capturing the full complexity of mathematics.

9: Almost 100 years younger than FOL if we ignore Russell's theory of types (1902), that was based on a set theory encoded in FOL.

Curry-Howard correspondence De Bruijn came up with the revolutionary idea that propositions could themselves be seen as types, by having judgments such as $t : 1 + 1 = 2$ where t is a *proof term* representing a proof of the proposition $1 + 1 = 2$. This *propositions-as-types* principle was rediscovered independently by Howard in 1978 [123], and developed further into a *proofs-as-programs* correspondence — also called *Curry-Howard correspondence* or *Curry-Howard isomorphism*[10] — where λ-terms in the simply-typed λ-calculus are put in one-to-one correspondence with proofs in the implicational fragment of the natural deduction system NJ of Gentzen.

[123]: Howard (1980), 'The Formulae-as-Types Notion of Construction'

10: In fact, Curry had already noticed in 1958 a similar connection between the types of combinators in his *combinatory logic*, and the axioms of Hilbert systems for implication [54] — hence the mention of Curry.

Thus the core of type theory is *intuitionistic* in nature, and the Curry-Howard correspondence has fostered many fruitful interactions between computer science, logic and constructive mathematics, with proof assistants acting as a crucial tool and source of investigations. One influential development in this direction has been the *intuitionistic type theory* of Martin-Löf (MLTT), which formed the basis for the implementation of many proof assistants like `NuPrl`, `Alf`, and most recently `Agda` [90]. A system closely related to MLTT, the *calculus of inductive constructions* (CoIC), is also implemented in two leading proof assistants: `Coq` [238] and `Lean` [178]. Following the proofs-as-programs correspondence, these systems support the creation of both proofs and programs that manipulate and compute mathematical objects, by compiling everything down to typed terms.

[238]: The Coq Development Team (2022), *The Coq Proof Assistant*

[178]: Moura et al. (2021), 'The Lean 4 Theorem Prover and Programming Language'

1.2.2. Interfaces

Elaboration Type theories are the logical foundation for the *kernel* of proof assistants, i.e. the part of the system that is responsible for checking formal proofs expressed in a *terse*, *machine-oriented* format. But it quickly became clear that this was not enough to make proof assistants a viable alternative to paper proofs: de Bruijn estimates that it takes a time factor of 20 to translate a paper proof into a formalized proof in `Automath` [56]. This factor has been estimated to be shrinkable to 4 in the `Mizar` proof assistant [250], thanks to the design of high-level *languages* for representing mathematical statements and proofs, that sit on top of the core logical theory[11]. The process of compiling a high-level proof text into a low-level proof term is called *elaboration*.

[56]: de Bruijn (1980), 'A survey of the project Automath'

[250]: Wiedijk (2000), *The De Bruijn Factor*

11: In the case of `Mizar`, a typed set theory based on FOL.

Statement languages Any proof assistant must provide the two following features in its statement language:

Logical primitives Naturally, one needs a way to write propositions formed with logical connectives and quantifiers. This is usually done in symbolic form, either with a custom ASCII notation, or with Unicode characters corresponding to the standard symbols in more modern proof assistants.

Mathematical notations In addition to the logical primitives, one needs to be able to express mathematical objects and operations in the domain of interest, e.g. numbers and arithmetic operators. Contrary to the logical language that can be hardcoded once and for all in the proof assistant, the mathematical language needs to be *extensible* by the user, so that custom notations can be defined for new mathematical objects.

> In this thesis, we focus exclusively on the *logical primitives*, because they are found universally in all types of mathematical reasoning.

We leave aside the question of providing domain-specific languages for particular branches of mathematics, which is nonetheless as much important. It has been tackled extensively in Ayers' thesis [11], and more specifically in his framework `ProofWidgets` for user-extensible, interactive graphical notations in the `Lean` proof assistant[12] [12]. De Moura and Ullrich have also designed a powerful macro system for `Lean` 4, that supports the elaboration of abstract notations into terms of the underlying type theory [243].

[11]: Ayers (2021), 'A Tool for Producing Verified, Explainable Proofs.'

12: For a similar approach to `ProofWidgets` in the context of functional programming, see [189].

[12]: Ayers et al. (2021), 'A Graphical User Interface Framework for Formal Verification'

[243]: Ullrich et al. (2022), 'Beyond Notations'

Proof languages Once one disposes of a convenient way to state mathematical propositions, comes the question of how to efficiently write *proofs* of these propositions. There have been broadly two approaches in the design of high-level proof languages:

Imperative proof languages, like imperative programming languages, offer a set of *commands* or instructions than can be given to the computer to modify some state stored in memory. The latter is called the *proof state*, and corresponds to the *partial* proof that is built by the system incrementally through the execution of commands. In the dominant paradigm, these commands are provided by the user in text form; since Robin Milner and the LCF theorem prover [175], they are called *tactics*. Proof files are literally *proof scripts*, that is the sequence of tactics typed in by the user.

[175]: Milner (1984), 'The use of machines to assist in rigorous proof'

Contrary to imperative programming languages, the main execution paradigm for proof scripts is *interactive*: the user triggers commands one at a time, so that she can visualize the intermediate proof states and determine the next steps to take. In most tactics-based ITPs, only the *statement* part of the proof state is shown, in a so-called *proof view* or *goal view*. This corresponds to the goals that the user needs to prove, and each goal is presented in the form of a sequent $\Gamma \Rightarrow C$, where C

is the conclusion that must be reached under the assumptions in Γ. Tactics generally apply to one goal at a time, and the user can choose which goal to *focus* at any point during the interaction. When the set of goals becomes empty, we say that the initial goal or conjecture has been *solved*.

Remark 1.2.1 The transformations performed by tactics can be more or less sophisticated. But, fundamentally, one finds elementary commands that correspond roughly to the inference rules, generally of natural deduction or sequent calculus. For instance, a goal $\Gamma \Rightarrow A \vee B$ (resp. $\Gamma \Rightarrow A \wedge B$) can be turned into either a goal $\Gamma \Rightarrow A$ or a goal $\Gamma \Rightarrow B$ (resp. into two goals $\Gamma \Rightarrow A$ and $\Gamma \Rightarrow B$), corresponding to the rules $\vee R_1$ and $\vee R_2$ (resp. $\wedge R$) of `LJ` in their backward reading (Figure 1.4).

`Coq` and `Lean` are examples of state-of-the-art proof assistants in the imperative paradigm.

Declarative proof languages follow a different approach, by aiming to provide an *explicit*, *self-contained* description of the proof. Although a goal view is still available to guide the user during the proof construction process, the finished proof must be readable *statically* by a human, without relying on the ITP to compute and display intermediate proof states. Consequently, declarative proofs are more verbose and take longer to type than their imperative homologues. But since they contain more information, they have the advantage of being more *robust* to slight changes to definitions or to the statement of the theorem being proved, and are generally easier to *debug*. They can also be put in correspondence with some proof-theoretical formalisms, usually Fitch-style natural deduction [90].

`Agda`, `Mizar` and `Isabelle` (with its `Isar` proof language [185]) are examples of state-of-the-art proof assistants in the declarative paradigm.

[185]: Nipkow (2002), 'Structured Proofs in Isar/HOL'

In this thesis, we focus mostly on exploring new modalities of interaction in the *imperative* paradigm. Only in Subsection 10.8.2 do we sketch a possible escape from the imperative/declarative dichotomy.

Remark 1.2.2 In some rare cases, an additional *natural language* layer is added on top of the proof language, to be as close as possible to informal proofs. With the recent development of large language models like GPT, such natural language translations are becoming increasingly convincing (see e.g. Patrick Massot's work in `Lean` [166]).

1.3. This thesis

1.3.1. Research goals

Universal user interface Many kinds of logical manipulations such as discharging assumptions, instantiating quantifiers, and composing lemmas, are conceptually universal; yet, a user wishing to carry out such manipulations in a particular proof assistant must express the wish in terms of the specific proof language of the system. A *universal user interface* would instead allow performing such manipulations directly on the *goal* itself, using physical, reversible, and incremental *actions* with immediate feedback. A sequence of user actions should then be representable in terms of any particular proof language.

Direct manipulation This approach to user interfaces has been termed *direct manipulation* by Shneiderman in the 1980s [222]. It is now at the heart of virtually every modern user interface and is arguably the major factor in the *personal computing revolution.* Indeed, it has opened the use of computing devices to a much wider audience, by allowing users to interact with them in an intuitive way that resembles interactions with physical objects in the real world. According to Shneiderman, this contrasts with the *command-line* paradigm, where users need to memorize a complex command language that often varies from one software to another within the *same* domain. This induces an unnecessary cognitive burden for newcomers, especially those unfamiliar with textual interfaces, which constitute the majority of users of computing devices nowadays; to the point that the hurdle is too great to overcome for most potential users.

[222]: Shneiderman (1983), 'Direct Manipulation: A Step Beyond Programming Languages'

Unfortunately, the user interfaces of state-of-the-art proof assistants are largely stuck in the pre-80s era of command-line interfaces, making them reserved to an audience of highly motivated, computer-savvy individuals. One could argue that like programming languages, this is due to their inherent *abstraction* capabilities, that can only be captured through the symbolic power of language; hence that this state of fact is unavoidable, and can only be solved through the addition (or improvement) of computer science curricula in primary and secondary education. We do not agree with this conception: we believe that the current state of user interfaces in ITPs is one of the major obstacles to their wider adoption in the mathematical community, both by professional researchers, teachers, and novice students alike.

> Our first main working hypothesis is that the direct manipulation paradigm is not only possible, but also *crucial* for building formal proofs in ITPs, if they are to become a viable alternative to paper proofs in mathematics.

In fact, following the proofs-as-programs correspondence, we also believe that this applies (to some extent) to *programming*, and that it is only a matter of time before direct manipulation becomes viable for building (general-purpose) programs, in the spirit of *visual programming languages.*

We do not explore this direction in this thesis, but it is one of our hopes that some of the techniques we develop will apply to programming as well; and one of the reasons why we chose to focus so much on *intuitionistic* logic.

Graphical deep inference A first attempt to design direct manipulation principles for interactive proof building was made in the 90s by the team of Gilles Kahn at Inria, where they coined the *Proof-by-Pointing* (PbP) paradigm [18]. The idea was to synthesize complex tactics from the simple act of *pointing* at specific locations inside expressions occurring in the goal, typically with a mouse cursor. More recently [37], Kaustuv Chaudhuri proposed a variation on this idea called *subformula linking* (SFL) or *Proof-by-Linking* (PbL), where instead of selecting expressions in isolation, one can *link* two of them together to make them interact.

[18]: Bertot et al. (1994), 'Proof by pointing'

[37]: Chaudhuri (2013), 'Subformula Linking as an Interaction Method'

In both cases, the expressions considered were logical formulas and the associated actions chains of inferences in FOL. Importantly, both paradigms rely on the use of *deep inference*, since the user can point at subformulas that occur at an arbitrary depth inside the goal. This is in contrast with the basic commands found in the proof languages of ITPs, where the user can only designate formulas appearing at the top-level of sequents[13]. While the semantics of PbP actions is still based on the *shallow* inference rules of sequent calculus, PbL fully embraces the deep inference paradigm by relying on CoS-style rewriting rules.

13: A notable exception is the `rewrite` tactic of imperative proof languages, where the user can specify *patterns* to designate particular occurrences of a term t that are to be rewritten into an equal term u, usually thanks to an assumed equation $t = u$. But coming up with such patterns is a lot slower than directly *pointing* at the locations of interest onscreen.

> All the works presented in this thesis can be seen as a direct continuation of the research programme initiated by PbP and PbL. The aim is to *replace* textual proof languages with *gestural actions* performed directly upon *goals* in a *graphical user interface* (GUI). To be as general as possible, we call such a paradigm *Proof-by-Action* (PbA).

Proof exploration Note that we believe such a replacement to be useful mostly during the *construction* or *writing* phase of proofs. Quoting Shneiderman [222]:

> The pleasure in using these systems stems from the capacity to manipulate the object of interest directly and to generate multiple alternatives rapidly.

Thus in the context of ITPs, the major advantage of a (well-designed) GUI in the PbA paradigm would be to enable the *rapid exploration* of multiple paths towards the construction of a complete proof. But once a proof has been found, the *dynamic* sequence of actions that led to it could be "compiled" into a *static*, textual representation of the proof in the favorite proof language of the user, to facilitate the *reading* phase.

As for the *modification* phase of proofs, the PbA paradigm requires a way to navigate and edit directly a recorded sequence of actions, and possibly a mechanism for mapping parts of a static proof object to the actions that generated them.

In this thesis, we leave the question of proof evolution in PbA — i.e. the design of interfaces for the reading and modification phases, that support a smooth interaction with the writing phase — for future work. A more detailed discussion can be found in Subsection 6.6.4.

Iconicity Contrary to a common misconception among logicians, Leibniz did not conceive of his *characteristica universalis* as a symbolic language, but rather as an *iconic* one [52, Chpt. 3]:

[52]: Couturat (1901), *La Logique de Leibniz*

> The true "real characteristic" [...] would express the composition of concepts by the combination of signs representing their simple elements, such that the correspondence between composite ideas and their symbols would be natural and no longer conventional. [...] This shows that the real characteristic was for him an ideography, that is, a system of signs that directly represent things (or, rather, ideas) and not words.

This is to be compared to Frege's *Begriffsschrift*, a graphical, two dimensional language and calculus of "pure thought", whose name has repeatedly been translated as *ideography* [81, 82].

[81]: Frege (1952), *Translations From the Philosophical Writings of Gottlob Frege*
[82]: Frege (1999), *L'idéographie*

Following the seminal work of Charles Sanders Peirce in *semiotics*[14], we define an *iconic* language as one whose signs are mainly *icons*, i.e. signs that *resemble* or share qualities with the objects they denote. This is to be contrasted with *symbolic* languages where most signs are just *symbols*, i.e. signs that *conventionally* denote their objects. In his systematic usage of *triads* of concepts, Peirce identified a third kind of sign, *indexes* [9]:

14: Semiotics is the systematic study of sign processes and the communication of meaning.

[9]: Atkin (2023), 'Peirce's Theory of Signs'

> if the constraints of successful signification require that the sign reflect qualitative features of the object, then the sign is an icon. If the constraints of successful signification require that the sign utilize some existential or physical connection between it and its object, then the sign is an index. And finally, if successful signification of the object requires that the sign utilize some convention, habit, or social rule or law that connects it with its object, then the sign is a symbol.

He even went further by analyzing icons into another trichotomy[15]: *images* that depend on simple quality; *diagrams*, who share *structural* relations among their constituents that are analogous to that of their object; and *metaphors*, that denote features of their object by relating them to features of another object [149].

15: Actually he only applied this trichotomy to pure icons devoid of any indexical elements, that he called *hypoicons*.

[149]: Legg (2008), 'The Problem of the Essential Icon'

Interestingly, Peirce held that mathematics relies mostly on *diagrammatic* thinking — observation of, and experimentation on, diagrams [122, Chpt. 6]. This agrees with the contemporary practice of mathematics: indeed, there is an increasing number of areas in mathematics — the most prominent one being *category theory* — where the heart of a proof lies in the dynamical construction of a *diagram* capturing the structure of interest in given mathematical objects. The natural language proof text is often just a means to explicit the meaning or intuition behind the dia-

[122]: Hookway (1985), *Peirce*

grammatic manipulations, or simply a retranscription of the commentary that the mathematician would give when unfolding the construction on a blackboard.

If one views logic as one particular type of mathematical reasoning — albeit one that is omnipresent in all branches of mathematics, then it is only natural to expect that some diagrammatic system should exist for it, that can express in the most natural way most (if not all) logical arguments. This is the *iconicity* thesis of Peirce, which motivated his inquiry into what is arguably the first diagrammatic proof system in history: the *existential graphs* (EGs). This will be the subject of Chapter 9, and the basis for the development of a *metaphorical* proof system in Chapter 10.

> Our second main working hypothesis exploited in the second part of this thesis, is that iconic representations of logical *statements*, and the proofs that result from their manipulation, can play a crucial role in the design of intuitive proof building interfaces.

1.3.2. Contributions

This thesis proposes several contributions toward the research goals highlighted above.

Symbolic manipulations In the first part of this thesis, we substantiate the PbA paradigm in the context of traditional representations of goals, by presenting a number of techniques based on the direct manipulation of *symbolic* formulas in sequents.

We start in Chapter 2 with an introduction to PbP and PbL, by describing how to reason with logical connectives, quantifiers and equality through *click* and *drag-and-drop* (DnD) actions in a prototype of GUI called `Actema`. In particular, DnD actions can be seen as a generalization of both the `apply` and `rewrite` tactics of imperative proof languages.

In Chapter 3, we ground the semantics of DnD actions in deep inference proof theory, by designing an intuitionistic variant of the CoS for subformula linking introduced by Chaudhuri in [37]. Our approach differs from Chaudhuri's mainly through our notion of *valid linkage*, which filters out unproductive DnD actions by restricting them to *unifiable* subformulas.

In Chapter 4, we present more advanced techniques in the PbA paradigm, that handle pervasive forms of reasoning in mathematical practice such as the use of *definitions*, reasoning by *induction*, and the *simplification* of expressions through automatic computation. This is illustrated through a few case studies of basic logical and mathematical problems.

In Chapter 5, we investigate an extension of PbA to sequents with *multiple* alternative conclusions, as opposed to the *single*-conclusion sequents found in the interface of almost every ITP. We argue that the use of direct manipulation greatly facilitates the management of multiple conclusions, and introduce a so-called *parallel* interaction operator to model reasoning in classical logic that involves the interaction of two conclusions.

Lastly in Chapter 6, we present `coq-actema`, a plugin that integrates the `Actema` web application as an interactive proof view in `Coq`. We focus on the architecture and interaction protocols that connect the different components of the system, and give an overview of the elaboration/compilation strategy that turns graphical actions performed in `Actema` into `Coq` proof terms. We also discuss current shortcomings of our approach and future directions for improvement, in particular concerning the question of proof evolution.

Iconic manipulations In the second part of this thesis, we explore a series of deep inference proof systems that give more structure to the notion of logical *goal*. These systems share the ability to represent goals in two alternative ways: either *textually* through a standard inductive syntax, or *graphically* through a metaphorical notation well-suited to direct manipulation. The first can be used as a machine representation in the backend of an ITP, and the latter as the substrate for GUIs in the frontend.

Bubble calculi In the first two chapters, we introduce a family of systems called *bubble calculi*. They are an extension of the theory of *nested sequents* first introduced by Brünnler [29], that we reframe as local rewriting systems with a graphical and topological interpretation. Bubble calculi enable an efficient sharing of contexts between subgoals, making them well-suited to the factorization of both forward and backward reasoning steps in proofs.

[29]: Brünnler (2009), 'Deep sequent systems for modal logic'

Chapter 7 presents the *asymmetric* bubble calculus BJ for intuitionistic logic, modelled after the *asymmetric* sequents of the intuitionistic sequent calculus LJ of Gentzen. It introduces the metaphor of *bubbles* as a way to iconically represent the separation and sharing of contexts between different subgoals.

Chapter 8 refines BJ into a more general and symmetric calculus for classical logic called system B, where bubbles can be *polarized* in addition to formulas. Intuitionistic, dual-intuitionistic and bi-intuitionistic logic can be recovered as fragments of system B, by forbidding certain inference rules that characterize the *porosity* of bubbles. We also devise a fully invertible variant of system B, that we conjecture to be complete.

Existential graphs In the last two chapters, we study two systems based on the existential graphs of Peirce, that allow us to achieve *full iconicity*: every logical construction has an associated icon, and thus there is no use anymore for the connectives and quantifiers of symbolic formulas. Hopefully, this shall remove a first barrier in the learning of formal logic, which lies in the *arbitrary* correspondence between symbols and their meaning.

In Chapter 9, we give a complete review of Peirce's original systems of EGs for propositional and first-order classical logic, which have been consistently neglected in the proof theory literature[16]. We propose in particular a novel inductive characterization of the syntax of EGs, as well as the first identification of an *analytic* fragment of the system for propositional logic that is complete for provability.

16: One reason might be that EGs have been invented at the end of the 19th century, *before* the birth of proof theory as a discipline in the 1920s under the impulse of Hilbert.

Finally, we introduce in Chapter 10 the *flower calculus*, an intuitionistic variant of EGs where statements are represented metaphorically as *flowers*. We partition the system into a *natural* fragment where every rule is both analytic and invertible, and a *cultural* fragment where every rule is non-invertible. We prove that the cultural fragment is admissible thanks to a completeness proof for the natural fragment with respect to Kripke semantics. We exploit these meta-theoretical results to design the `Flower Prover`, a prototype of GUI in the PbA paradigm that aims to unify the concepts of *goal* and *theory* in a *modal interface*: goals correspond to flowers manipulated with natural rules in Proof mode; while theories correspond to the same flowers manipulated with cultural rules in Edit mode. The `Flower Prover` is also the first *mobile-friendly* interface for ITPs that we know of.

Note

Most of the content of Chapter 2 and Chapter 3 has been previously published in [67], and the `coq-actema` system described in Chapter 6 is under active development by us and Benjamin Werner [68]. A shortened version of Chapter 9 and Chapter 10 has been submitted for publication at FSCD 2024 [66]. All other chapters present completely original and personal work.

1.3.3. How to read

Reading order The ordering of chapters in this thesis is mostly *chronological*, reflecting the order in which the ideas were developed. For readers interested in all of the contributions, we thus advise reading all chapters in order.

Still, the investigations into iconic manipulations in the second part started as an offshoot of those on symbolic manipulations in the first part, and were carried mostly in parallel. Although we sometimes reference ideas from chapters in the first part, the second part can thus be read mostly independently from the first one.

In all cases, the reader should start with Chapter 2, which gives a taste of the PbA paradigm explored in all other chapters.

Figure 1.6 shows the precise graph of dependencies between chapters. Four independent paths can be followed:

The applied road (④ → ⑥) If you want to see to what extent the PbA paradigm can currently be applied for practical theorem proving in real proof assistants, this is the right path for you.

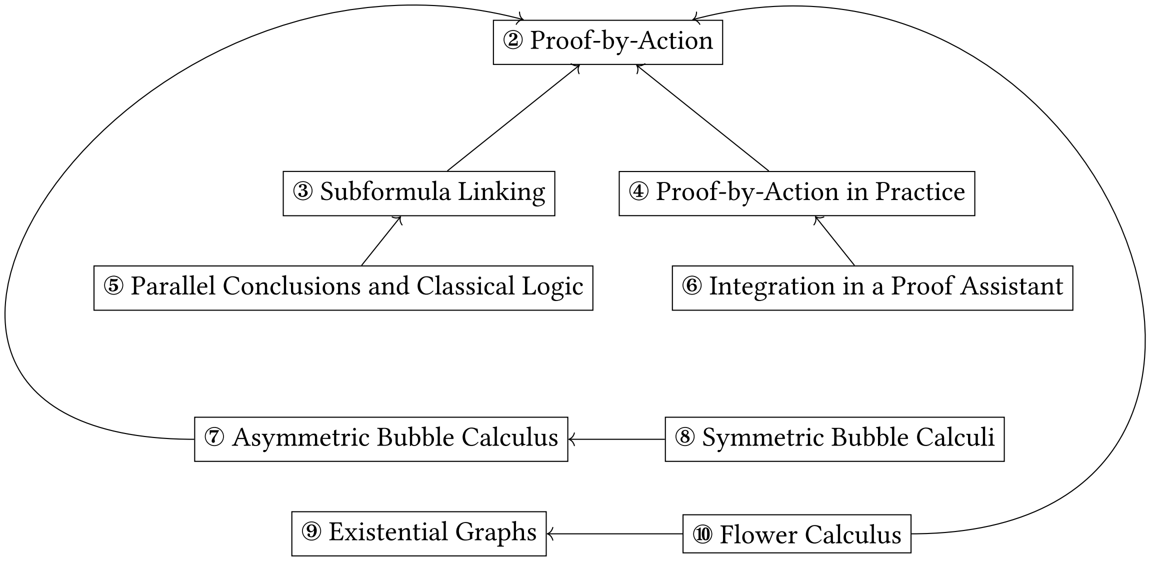


Figure 1.6.: Dependency graph between chapters

Proof theory of SFL (③ → ⑤) This path is for readers only interested in the proof theory of subformula linking, which is the foundation for the semantics of DnD actions on symbolic formulas.

Bubble calculi (⑦ → ⑧) This path is for readers only interested in the proof theory and potential applications of bubble calculi.

Flower calculus (⑨ → ⑩) This path is for readers only interested in the proof theory of the flower calculus, and its applications to automated and interactive theorem proving.

A last option is to read only Chapter 9, skipping even Chapter 2. This might be of interest to people looking for an introduction to the existential graphs of Peirce.

Color Some parts of this document make a heavy, *semantic* use of colors. Although all important concepts still have a textual, color-independent presentation, it is recommended to print this document with a decent amount of color levels.

Digression

We will sometimes develop ideas loosely related to the main text in *digression* boxes such as this one: at least on first reading, they can be safely ignored. We distinguish them from normal side notes, which are usually shorter and more relevant to the matter at hand.

Hyperlinks We tend to cross-reference many ideas from different chapters with the help of *hyperlinks*. In particular, we use the `knowledge` package from Thomas Colcombet to hyperlink occurrences of concepts and notations to the place where they are introduced. We thus recommend the usage of a PDF reader that supports at least hyperlink *jumping*, and if possible hyperlink *preview* for a more comfortable reading experience.

1.4. Related works

Window inference Other researchers have stressed the importance of being able to reason *deep* inside formulas to provide intuitive proof steps. The first and biggest line of research supporting this idea is probably that of *window inference*, which started in 1993 with the seminal article of P.J. Robinson and J. Staples [216], and slowly became out of fashion during the 2000s. This is well expressed in the following quote from one of its main contributors, Serge Autexier [10, p. 184–187]:

[216]: Robinson et al. (1993), 'Formalizing a Hierarchical Structure of Practical Mathematical Reasoning'

[10]: Autexier (2004), 'Hierarchical Contextual Reasoning'

> We believe it is an essential feature of a calculus for intuitive reasoning to support the transformation of parts of a formula without actually being forced to decompose the formula. In that respect the inference rules of Schütte's proof theory are a clear contribution. [...] One motivation for the development of the CORE proof theory was to overcome the need for formula decomposition as enforced by sequent and natural deduction calculi in order to support an intuitive reasoning style.

Thus we are not the first to attempt to design new proof systems based on deep inference principles, with the explicit objective of improving the usability of ITPs. However, we believe our approach is unique in that it emphasizes two aspects:

- the use of *direct manipulation* on goals to perform proof steps (although some pointing interactions were already at work in window inference-based systems);
- in the second part of this thesis, the use of *iconic* representations for the proof state, that stray away from traditional symbolic formulas.

Digression

Note that during most of the period when window inference was developed, the terminology of "deep inference" had not been introduced yet. Indeed, the first article on the subject appeared in 1999 [110], with very different motivations in mind: namely, the development of a proof-theoretical approach unifying concurrent and sequential computation, resulting in the calculus of structures for the logic BV. However, some proof systems based on deep inference principles already existed and inspired researchers in window inference, as witnessed by the reference to Schütte's proof theory in the above quote.

Ayers' thesis More recently, Ayers described in his thesis a new tool for producing verifiable and explainable (formal) proofs, including both theoretical discussions of novel concepts and designs for components of proof assistants, and practical implementations of software evaluated through user studies [11]. Notable contributions from our point of view are:

[11]: Ayers (2021), 'A Tool for Producing Verified, Explainable Proofs.'

- his `Box` development calculus, which introduces a unified `Box` data structure representing at the same time goals and partial proofs, with the aim to offer more "human-like" interfaces for both the *construction* and the *presentation* of proofs;
- and his `ProofWidgets` framework, that allows to extend the `Lean` proof assistant with new interactive and domain-specific notations for mathematical objects, thus offering a form of *end-user* programming.

The `Box` data structure easily lends itself to visualization in a two dimensional graphical notation, while `ProofWidgets` promises great capabilities for proofs by both direct and iconic manipulation.

However, the work of Ayers focuses mainly on designing a general framework that can integrate modern interfaces for proofs in the `Lean` proof assistant, while we focus on exploring various proof calculi that provide the foundations for such interfaces at the purely logical level, independently of any particular proof assistant. Thus we believe that our work is quite complementary to Ayers': it emphasizes different aspects while sharing a common vision for the future of proof assistants, where modern graphical interfaces play a crucial role in improving the interaction between the user and the computer.

Symbolic Manipulations

Proof-by-Action 2.

> When it is obvious that the goals cannot be reached, don't adjust the goals, adjust the action steps.
>
> **Confucius**

2.1 Logical setting 25
2.2 A first example 26
2.2.1 Layout 26
2.2.2 Two kinds of actions . . . 27
2.2.3 Modelling the mechanism 27
2.3 Proof steps through clicks 28
2.4 A simple example involving equality . . . 30
2.5 Drag-and-dropping through connectives . . 31
2.5.1 Conjunction and disjunction 31
2.5.2 Implication 32
2.5.3 Quantifiers 33
2.5.4 Dependency between variables 34
2.5.5 Conditional rewriting . . 35
2.6 Related works 35

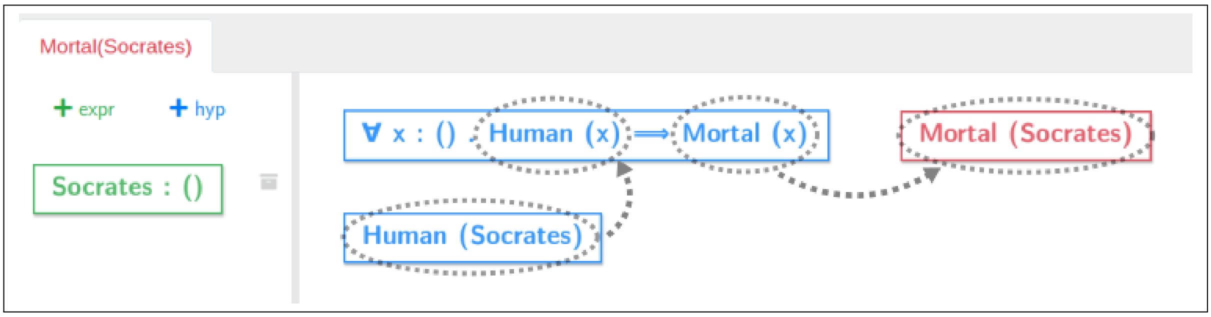
In this chapter, we focus on how both *click* and *drag-and-drop* (DnD) actions upon the formulas of a sequent can implement proof construction operations corresponding to the core logic; that is how they deal with logical connectives, quantifiers and equality. We present these core principles through various illustrations and examples in first-order logic. The technical and proof-theoretical foundations for the semantics of DnD actions will be investigated more thoroughly in Chapter 3. More advanced features of the Proof-by-Action paradigm that go beyond the core logic are illustrated in Chapter 4, and Chapter 6 explains how it can be integrated in a mainstream proof assistant.

We have started to implement the paradigm in a prototype named `Actema` (for "**Act**ive math**ema**tics") running through a web `HTML`/`JavaScript` interface. At the time of writing, a standalone version of the prototype (i.e. which does not use an existing theorem prover as its backend) is publicly available online [69]. This possibility to experiment in practice, even though yet on a small scale, gave valuable feedback for crafting the way DnD actions are to be translated into proof construction steps in an intuitive and practical way. A description of the overall architecture and implementation design of `Actema` will be provided in Chapter 6.

The chapter is organized as follows: Section 2.1 briefly outlines its logical setting. Section 2.2 describes the basic features of a graphical proof interface based on our principles, and illustrates them with a famous syllogism from Aristotle. Section 2.3 shows how it can integrate Proof-by-Pointing capabilities through click actions. The two following sections explain, through further examples, how drag-and-drop actions work; first for so-called *rewrite* actions involving equalities, then for actions involving logical connectives and quantifiers. We end in Section 2.6 with a discussion of related works.

2.1. Logical setting

Any proof system must implement a given logical formalism. What we describe here ought to be applied to a wide range of formalisms, but in this chapter we focus mainly on the core of intuitionistic FOL with equality. This allows us to consider sequents where hypotheses are unordered which, in turn, simplifies the technical presentation. We will thus write $\Gamma, A \Rightarrow C$ for a sequent where A is among the hypotheses.



The conclusion is red on the right, the two hypotheses blue on the left. The gray dotted arrows have been added to show the two possible actions.

Figure 2.1.: A partial screenshot showing a goal in the `Actema` prototype

We use and do not recall the usual definitions of terms and propositions in FOL. We assume a first-order language (function and predicate symbols) is given. Provability is defined over sequents $\Gamma \Rightarrow C$ by the usual inference rules of natural deduction (Figure 1.1) and/or sequent calculus (Figure 1.4).

Equality is treated in a common way: $=$ is a binary predicate symbol written in the usual infix notation, together with the reflexivity axiom $\forall x.x = x$ and the Leibniz scheme, stating that for any proposition A one has

$$\forall x.\forall y.x = y \wedge A \supset A\{x/y\}.$$

We will not consider, on paper, the details of variable renaming in substitutions, implicitly applying the so-called *Barendregt convention*, that bound and free variables are distinct and that a variable is bound at most once.

Extending this work to simple extensions of FOL, like multi-sorted predicate calculus is straightforward (and actually done in the `Actema` prototype). Some interesting points may show up when considering how to apply this work to more complex formalisms like type theories. We will not explore these questions here.

2.2. A first example

2.2.1. Layout

One advantage of the PbA paradigm, is that it allows a very lean visual layout of the proof state. There is no need to name hypotheses. In the prototype we also dispense with a text buffer, since proofs are solely built through graphical actions.

Figure 2.1 shows the layout of the system using the ancient example from Aristotle. A goal appears as a set of *items* whose nature is defined by their respective colors[1]:

1: We are well aware that, in later implementations, this color-based distinction ought to be complemented by some other visual distinction, at least for users with impaired color vision. But in the present description we stick to the red/blue denomination, as it is conveniently concise.

- a *red* item which is the proposition to be proved, that is the *conclusion*;
- *blue* items, which are the local *hypotheses*;
- *green* items, which are the declared (first-order) *objects*.

The items are what the user can act upon: either by *clicking* on them, or by *moving* them. Each item can be positioned freely on a so-called *proof canvas*, which is depicted by the white background in Figure 2.1.

Often in the course of a proof, one will want to add new items: either a new conjecture (blue item), or a new object (green item) that would be helpful to solve the current goal. These can be done respectively with the blue `+hyp` and the green `+expr` buttons, placed in the top-left corner of the screen in Figure 2.1. When clicked, they prompt the user for the statement of the conjecture, or the name and expression defining the object[2]. The `+hyp` button will also create a new subgoal requiring to prove the conjecture within the current context.

Finally, note that each goal is displayed in its own *tab*, whose title is the statement of the goal's conclusion.

2: For now we ask the user to input textual data, in an idiosyncratic syntax specific to the logic of `Actema`. A desirable feature would be to provide some elaborate input mechanism tailored to the type of object the user wants to create. This would obviously require some extensibility to new domain-specific input interfaces: typically one could imagine plugging a tool like GeoGebra to construct geometrical figures [98], or a categorical diagram editor like YADE [145]. The `ProofWidgets` framework is particularly well-suited for this task [12].

2.2.2. Two kinds of actions

In this example, there are two possible actions.

- A first one is to bring together by DnD the conclusion $\mathsf{Mortal}(\mathsf{Socrates})$ with the conclusion of the first hypothesis $\mathsf{Mortal}(x)$. This will transform the goal by changing the conclusion to $\mathsf{Human}(\mathsf{Socrates})$.
- A second possibility is to combine the two hypotheses; more precisely to bring together the item $\mathsf{Human}(\mathsf{Socrates})$ with the premise $\mathsf{Human}(x)$ of the first hypothesis. This will yield a new hypothesis $\mathsf{Mortal}(\mathsf{Socrates})$.

The first case is what we call a *backward* step where the conclusion is modified by using a hypothesis. The second case is a *forward* step where two known facts are combined to deduce a new fact, that is an additional blue item.

In both cases, the proof can then be finished invoking the logical axiom rule. In practice this means bringing together the blue hypothesis $\mathsf{Human}(\mathsf{Socrates})$ (resp. the new blue fact $\mathsf{Mortal}(\mathsf{Socrates})$) with the identical red conclusion.

2.2.3. Modelling the mechanism

A backward step involves a hypothesis, here $\forall x.\mathsf{Human}(x) \supset \mathsf{Mortal}(x)$ and the conclusion, here $\mathsf{Mortal}(\mathsf{Socrates})$. Furthermore, the action actually links together two *subterms* of each of these items; this is written by squaring these subterms. The symbol $\circleddash$, called an *interaction operator*, is meant to describe the result of the interaction. Internally, the behavior of this operator is defined by a set of rewriting rules of the form $A \circleddash B \to C$ given in Figure 2.4. Here is the sequence of rewrites corresponding to the example[3]:

3: Note that $\circleddash$ has lower precedence than all logical connectives.

$$
\begin{array}{lll}
 & \forall x.\mathsf{Human}(x) \supset \boxed{\mathsf{Mortal}(x)} \circleddash \boxed{\mathsf{Mortal}(\mathsf{Socrates})} & \\
\to & \mathsf{Human}(\mathsf{Socrates}) \supset \boxed{\mathsf{Mortal}(\mathsf{Socrates})} \circleddash \boxed{\mathsf{Mortal}(\mathsf{Socrates})} & \mathsf{L}\forall\mathsf{i} \\
\to & \mathsf{Human}(\mathsf{Socrates}) \wedge \left(\boxed{\mathsf{Mortal}(\mathsf{Socrates})} \circleddash \boxed{\mathsf{Mortal}(\mathsf{Socrates})}\right) & \mathsf{L}{\supset}_2 \\
\to & \mathsf{Human}(\mathsf{Socrates}) \wedge \top & \mathsf{id} \\
\to & \mathsf{Human}(\mathsf{Socrates}) & \mathsf{neur}
\end{array}
$$

Notice that:

- These elementary rewrites are not visible for the user. What she sees is the final result of the action, that is the last expression of the rewrite sequence.
- The definitions of the rewriting rules in Figure 2.4 do not involve squared subterms. The information of which subterms are squared is only used by the system to decide which rules to apply in which order.
- The last step applies the neur rule defined in Figure 2.5.

In general, the action solves the goal when the interaction ends with the trivially true proposition $\top$. The base case being the action corresponding to the axiom/identity rule id: $A \circleddash A \to \top$.

A forward step, on the other hand, involves two (subterms of two) hypotheses. The interaction operator between two hypotheses is written $\circledast$. In the example above, the detail of the interaction is:

$$
\begin{array}{lll}
 & \forall x.\boxed{\mathsf{Human}(x)} \supset \mathsf{Mortal}(x) \circledast \boxed{\mathsf{Human}(\mathsf{Socrates})} & \\
\to & \boxed{\mathsf{Human}(\mathsf{Socrates})} \supset \mathsf{Mortal}(\mathsf{Socrates}) \circledast \boxed{\mathsf{Human}(\mathsf{Socrates})} & \mathsf{F}\forall\mathsf{i} \\
\to & \left(\boxed{\mathsf{Human}(\mathsf{Socrates})} \circleddash \boxed{\mathsf{Human}(\mathsf{Socrates})}\right) \supset \mathsf{Mortal}(\mathsf{Socrates}) & \mathsf{F}{\supset}_1 \\
\to & \top \supset \mathsf{Mortal}(\mathsf{Socrates}) & \mathsf{id} \\
\to & \mathsf{Mortal}(\mathsf{Socrates}) & \mathsf{neul}
\end{array}
$$

The final result is the new hypothesis. We come back to the study of the rewriting rules of $\circleddash$ and $\circledast$ in Chapter 3.

2.3. Proof steps through clicks

Drag-and-drop actions involve two items. Some proof steps involve only one item; they can be associated to the action of clicking on this item. The general scheme is that clicking on a connective or quantifier allows to "break" or destruct this connective. The results of clicks are not very surprising, but this feature is necessary to complement drag-and-drop actions.

- Clicking on a blue conjunction $A \wedge B$ transforms the item into two separate blue items A and B.

- Clicking on a red conjunction $A \wedge B$ splits the goal into two subgoals, whose conclusions are respectively A and B.
- Clicking on a blue disjunction $A \vee B$ splits the goal into two subgoals of same conclusion, with A (resp. B) added as a new hypothesis.
- Clicking on the left (resp. right)-hand subterm of a red disjunction $A \vee B$ replaces this red conclusion by A (resp. B).
- Clicking on a red implication $A \supset B$ breaks it into a new red conclusion B and a new blue hypothesis A.
- Clicking on a red universal quantifier $\forall x.A$ introduces a new object x and the conclusion becomes A.
- Clicking on a blue existential $\exists x.A$ introduces a new object x together with a blue hypothesis A.
- Clicking on a red equality $t = t$ solves the goal immediately.

One can see that these actions correspond essentially to the right introduction rules of the head connective for the conclusion, and either the elimination rule from NJ or the left introduction rule from LJ for hypotheses. The exact mapping between click actions and inference rules is given in Table 2.1. A few remarks are in order:

- In the current implementation of `Actema`, clicking on a blue item $A \supset B$ will work only if the conclusion is B, replacing the latter with A. An alternative is to use the $\supset$L rule from sequent calculus, which is applicable in every context.
- There is no action mapped to red $\bot$ items, simply because $\bot$ does not have any introduction rule.
- There is currently no action mapped to blue $\forall$ and red $\exists$ items. The reason is that one needs additional information about the *witness* to be used when instantiating with the $\forall$L or $\exists$R rule. This could be provided with further input (e.g. from a dialog box), but this would need a change in the communication protocol between the frontend and backend of `Actema`. Instead we decompose this in two steps: first the user can add the witness as a new object by using the `+expr` button; then she can drag the corresponding green item and drop it on the quantified item to instantiate it. It is also possible to select with the mouse an arbitrary subexpression occurring in any item of the current goal, and then drag-and-drop the item holding the selected subexpression instead.

Table 2.1.: Mapping of click actions to inference rules

Head connective	Red item	Blue item
$\top$	$\top$R	$\top$L
$\bot$	$\varnothing$	$\bot$L
$\wedge$	$\wedge$R	$\wedge$L
$\vee$	$\vee\mathrm{R}_1$, $\vee\mathrm{R}_2$	$\vee$L
$\supset$	$\supset$R	$\supset$e
$\forall$	$\forall$R	$\varnothing$
$\exists$	$\varnothing$	$\exists$L

$$\frac{}{\Gamma \Rightarrow \top}\,\top\mathrm{R} \qquad \frac{\Gamma \Rightarrow C}{\Gamma, \top \Rightarrow C}\,\top\mathrm{L}$$

$$\frac{\Gamma, A, B \Rightarrow C}{\Gamma, A \wedge B \Rightarrow C}\,\wedge\mathrm{L}$$

Figure 2.2.: Variants of some sequent calculus rules

Remark 2.3.1 (Click completeness) From the previous remarks and the completeness of the cut-free sequent calculus LJ, it follows that click actions, when combined with new object declarations and DnD instantiations, provide a sufficient set of interactions to prove any true formula of (intuitionistic) first-order logic.

It is possible to associate some more complex effects to click actions performed on locations deeper under connectives. This is the essence of

Proof-by-Pointing, and [18] provides ample description. Since we here focus more on drag-and-drop actions, we do not detail further more advanced PbP features. The version of `Actema` presented in Chapter 6 provides an implementation of PbP available as a contextual menu action.

[18]: Bertot et al. (1994), 'Proof by pointing'

2.4. A simple example involving equality

Most interactive theorem provers expose a *rewrite* tactic that allows the use of equality hypotheses, that is known equations of the form $t = u$, in order to replace some occurrences of t by u (or symmetrically, occurrences of u by t). This substitution can be performed in the conclusion or in hypotheses. Specifying the occurrences to be replaced with textual commands can be quite tedious, since it involves either dealing with some form of naming/numbering to designate locations of subterms, or writing manually patterns which duplicate parts of the structure of terms.

In our setting we can provide this replacement operation through drag-and-drop. The user points at the occurrence(s) of t to be replaced, and then brings them to the corresponding side of the equality.

Figure 2.3 shows a very elementary example where one wants to prove $1 + 1 = 2$ in the setting of Peano arithmetic. For any number n, we write $n{\oplus}1$ to denote the application of the successor function to n; closed terms are directly written in decimal notation. The proof goes as follows[4]:

4: Note that we use the symbol $\equiv$ to denote *syntactic equality* of formulas at the meta-level, by contrast to the symbol $=$ denoting equality at the object-level.

- We link the left-hand side $x + y{\oplus}1$ of the second addition axiom with $1 + 1$ in the conclusion, which has the effect of rewriting $1 + 1$ into $(1 + 0){\oplus}1$:

$$\begin{array}{lll}
 & \forall x.\forall y.\boxed{x + y{\oplus}1} = (x + y){\oplus}1 \circleddash \boxed{1 + 1} = 2 & \\
\rightarrow & \forall y.\boxed{1 + y{\oplus}1} = (1 + y){\oplus}1 \circleddash \boxed{1 + 1} = 2 & \mathsf{L}\forall\mathsf{i} \\
\rightarrow & \boxed{1 + 0{\oplus}1} = (1 + 0){\oplus}1 \circleddash \boxed{1 + 1} = 2 & \mathsf{L}\forall\mathsf{i} \\
\equiv & \boxed{1 + 1} = (1 + 0){\oplus}1 \circleddash \boxed{1 + 1} = 2 & \\
\rightarrow & (1 + 0){\oplus}1 = 2 & \mathsf{L}{=}_1
\end{array}$$

- We link the right-hand side $x + 0$ of the first addition axiom with $1 + 0$ in the conclusion, which rewrites $1 + 0$ into 1:

$$\begin{array}{lll}
 & \forall x.x = \boxed{x + 0} \circleddash \left(\boxed{1 + 0}\right){\oplus}1 = 2 & \\
\rightarrow & 1 = \boxed{1 + 0} \circleddash \left(\boxed{1 + 0}\right){\oplus}1 = 2 & \mathsf{L}\forall\mathsf{i} \\
\rightarrow & 1{\oplus}1 = 2 & \mathsf{L}{=}_2 \\
\equiv & 2 = 2 &
\end{array}$$

We end up with the conclusion $2 = 2$, which is provable by a simple click. Notice how the orientation of the two rewrites is determined by which side of the equality is selected. Also, in this case, the rewrites correspond to backward proof steps, because rewriting is performed in the conclusion. Similar rules ($\mathsf{F}{=}_1$ and $\mathsf{F}{=}_2$) are used to perform rewriting in hypotheses.

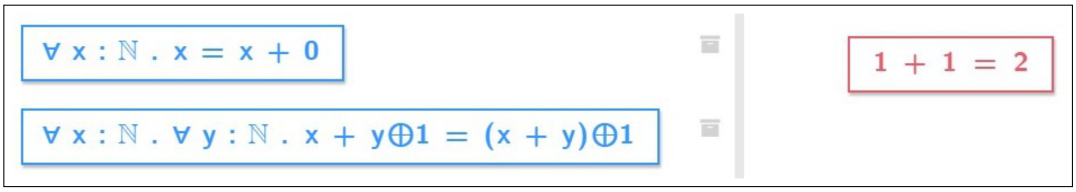


Figure 2.3.: Proving $1 + 1 = 2$ in Peano arithmetic

2.5. Drag-and-dropping through connectives

We mentioned in Section 2.3 that it is possible to destruct logical connectives through click actions. In many cases however, this will not be necessary: because a drag-and-drop involves subterms of the items involved, one can often directly use (resp. act on) the part of the hypothesis (resp. conclusion) which is of interest.

2.5.1. Conjunction and disjunction

The conjunction is an easy to explain case. A hypothesis of the form $A \wedge B$ can be used directly both as evidence for A and as evidence for B. This is modeled by the rules $\mathsf{L}\wedge_1$ and $\mathsf{L}\wedge_2$. A very simple action is thus:

$$\begin{array}{llll}
\boxed{A} \wedge B \ogreaterthan \boxed{A} & \to & \boxed{A} \ogreaterthan \boxed{A} & \mathsf{L}\wedge_1 \\
& \to & \top & \mathsf{id}
\end{array}$$

On the other hand, considering a conjunctive goal $A \wedge B$, one can simplify or solve one of the branches by a DnD action. This involves rules $\mathsf{R}\wedge_1$ and $\mathsf{R}\wedge_2$. For instance:

$$\begin{array}{llll}
\boxed{A} \ogreaterthan \boxed{A} \wedge B & \to & \left(\boxed{A} \ogreaterthan \boxed{A}\right) \wedge B & \mathsf{R}\wedge_1 \\
& \to & \top \wedge B & \mathsf{id} \\
& \to & B & \mathsf{neul}
\end{array}$$

Red disjunctions work similarly to conjunctive goals, except that solving one branch will solve the entire goal. A nice consequence of this, which is hard to simulate with textual tactics, is that one can just simplify one branch of a disjunction without comitting to proving it entirely:

$$\begin{array}{llll}
\boxed{A} \ogreaterthan (B \wedge \boxed{A}) \vee C & \to & \left(\boxed{A} \ogreaterthan B \wedge \boxed{A}\right) \vee C & \mathsf{R}\vee_1 \\
& \to & \left(B \wedge \left(\boxed{A} \ogreaterthan \boxed{A}\right)\right) \vee C & \mathsf{R}\wedge_2 \\
& \to & (B \wedge \top) \vee C & \mathsf{id} \\
& \to & B \vee C & \mathsf{neur}
\end{array}$$

Disjunctive hypotheses also have a backward behavior defined by the rules $\mathsf{L}\vee_1$ and $\mathsf{L}\vee_2$, although in most cases one will prefer the usual subgoal semantics associated with click actions. More interesting is their forward behavior with the rules $\mathsf{F}\vee_1$ and $\mathsf{F}\vee_2$, in particular when they interact with negated hypotheses. For instance:

$$\begin{array}{llll} \boxed{A} \vee B \circledast \neg \boxed{A} & \to & \left(\boxed{A} \circledast \neg \boxed{A}\right) \vee B & \mathsf{F}\vee_1 \\ & \to & \neg\left(\boxed{A} \ominus \boxed{A}\right) \vee B & \mathsf{F}\supset_1 \\ & \to & \neg\top \vee B & \mathsf{id} \\ & \to & \bot \vee B & \mathsf{neul} \\ & \to & B & \mathsf{neul} \end{array}$$

We have noticed that on some examples, such actions could provide a significant speed-up with respect to traditional textual command provers. We give a more concrete example in Section 4.1.

Notice that we used rules associated with implication, since negation can be defined by $\neg A \triangleq A \supset \bot$.

2.5.2. Implication

The implication connective is crucial, because it is not monotone. More precisely, the roles of hypotheses and conclusions are reversed on the left of an implication. We start with some very basic examples for the various elementary cases.

Using the right hand part of a hypothesis $A \supset B$ turns a conclusion B into A.

$$\begin{array}{llll} A \supset \boxed{B} \ominus \boxed{B} & \to & A \wedge \left(\boxed{B} \ominus \boxed{B}\right) & \mathsf{L}\supset_2 \\ & \to & A \wedge \top & \mathsf{id} \\ & \to & A & \mathsf{neul} \end{array}$$

This can also be done under conjunctions and/or disjunctions:

$$A \supset \boxed{B} \ominus C \wedge \left(D \vee \boxed{B}\right) \quad \to^* \quad C \wedge (D \vee A)$$

An interesting point is what happens when using implications with several premisses. The curried and uncurried versions of the implication will behave exactly the same way:

$$A \supset B \supset \boxed{C} \ominus D \vee \boxed{C} \quad \to^* \quad D \vee (A \wedge B)$$

and

$$A \wedge B \supset \boxed{C} \ominus D \vee \boxed{C} \quad \to^* \quad D \vee (A \wedge B)$$

As we have seen in Aristotle's example (Section 2.2), blue implications can also be used in forward steps, where another hypothesis matches one of their premisses.

A first nice feature is the ability to strengthen a hypothesis by providing evidence for any of its premises:

$$B \supset \boxed{A} \supset C \circledast \boxed{A} \quad \to^* \quad B \supset C$$

and again the same can be done for the uncurryfied version:

$$B \wedge \boxed{A} \supset C \circledast \boxed{A} \quad \to^* \quad B \supset C.$$

The two aspects of the implication can be combined:

$$B \supset \boxed{A} \supset C \circledast D \supset \boxed{A} \quad \to^* \quad B \supset D \supset C$$

or:

$$B \wedge \boxed{A} \supset C \circledast D \supset \boxed{A} \quad \to^* \quad B \wedge D \supset C.$$

Note that there is almost no difference in the way one uses different versions of a hypothesis $A \supset B \supset C$, $A \wedge B \supset C$, but also $B \supset A \supset C$, in forward as well as in backward steps[5]. This underlines, we hope, that our proposal makes the proof construction process much less dependent on arbitrary syntactical details, like the order of hypotheses or whether they come in curryfied form or not.

5: When viewed as types through the Curry-Howard isomorphism, $A \supset B \supset C$, $A \wedge B \supset C$, $B \wedge A \supset C$ and $B \supset A \supset C$ are *isomorphic types*; and Roberto di Cosmo [62] has also precisely underlined that type isomorphisms should help to free the programmer from arbitrary syntactical choices.

Also, the rules for implication combined with the rules for equality $\mathsf{L}{=}_\mathsf{i}$ or $\mathsf{F}{=}_\mathsf{i}$ naturally give access to *conditional rewriting*; we detail this in combination with quantifiers in the next section.

As for red implications, they also have a backward semantics with the rules $\mathsf{R}{\supset}_1$ and $\mathsf{R}{\supset}_2$, but most of the time one will want to destruct them immediately by click. An exception could be if one wants to simplify some part of an implicative, inductive goal before starting the induction.

2.5.3. Quantifiers

As the first example of this chapter shows, drag-and-drop actions work through quantifiers and can trigger instantiations of quantified variables. This is made possible by the rules $\mathsf{L}\forall\mathsf{i}$ and $\mathsf{F}\forall\mathsf{i}$, which allow the instantiation of a variable universally quantified in a hypothesis.

Symmetrically, a variable quantified existentially in a conclusion can also be instantiated. For instance:

$$\begin{array}{lll} \boxed{A(t)} \circleddash \exists x. \boxed{A(x)} & \to \quad \boxed{A(t)} \circleddash \boxed{A(t)} & \mathsf{L}\forall\mathsf{i} \\ & \to \quad \top & \mathsf{id} \end{array}$$

An interesting feature is the possibility to modify propositions under quantifiers. Consider the following possible goal:

$$\forall a. \exists b. A(f(a) + g(b))$$

where A, f and g can be complex expressions. Suppose we have a lemma allowing us to prove:

$$\forall a. \exists b. A(g(b) + f(a)).$$

Switching from one formulation to the other, involves one use of the commutativity property $\forall x.\forall y.x + y = y + x$. In our setting, the equality can be used under quantifiers in one single action:

$$\begin{array}{ll} & \forall x.\forall y.\boxed{x+y} = y + x \circleddash \forall a.\exists b.A\left(\boxed{f(a)+g(b)}\right) \\ \to^* & \forall a.\exists b.A(g(b)+f(a)) \end{array}$$

Note also that it is possible to instantiate only some of the universally quantified variables in the items involved. In general, a universally quantified variable can be instantiated when the quantifier is in a negative position; for instance:

$$\forall x.\forall y.\boxed{P(y)} \supset R(x,y) \circledast \boxed{P(a)} \quad \to^* \quad \forall x.R(x,a)$$

This last example illustrates how partial instantiation abstracts away the order in which quantifiers are declared, very much like the partial application presented earlier for implication[6].

6: This fact should not be too surprising to the reader familiar with dependent type theory, where implication is usually defined as a special case of universal quantification.

Again, in some cases, only some existential quantifiers may be instantiated following a DnD:

$$\boxed{P(a)} \circleddash \exists x.\exists y.\boxed{P(y)} \wedge R(x,y) \quad \to^* \quad \exists x.R(x,a)$$

When using an existential assumption, one can either destruct it through a click, or use or transform it through a DnD; for instance:

$$\exists x.\boxed{P(x)} \circledast \forall y.\boxed{P(y)} \supset Q(y) \quad \to^* \quad \exists x.Q(x)$$

2.5.4. Dependency between variables

Some more advanced examples yield simultaneous instantiations of existentially and universally quantified variables. In such cases, the system needs to check some dependency conditions. For instance, the following DnD is valid and solves the goal through one action:

$$\begin{array}{lll} & \exists y.\forall x.\boxed{R(x,y)} \circleddash \forall x'.\exists y'.\boxed{R(x',y')} & \\ \to & \forall y.\left(\forall x.\boxed{R(x,y)} \circleddash \forall x'.\exists y'.\boxed{R(x',y')}\right) & \mathsf{L\exists s} \\ \to & \forall y.\forall x'.\left(\forall x.\boxed{R(x,y)} \circleddash \exists y'.\boxed{R(x',y')}\right) & \mathsf{R\forall s} \\ \to & \forall y.\forall x'.\left(\forall x.\boxed{R(x,y)} \circleddash \boxed{R(x',y)}\right) & \mathsf{R\exists i} \\ \to & \forall y.\forall x'.\left(\boxed{R(x',y)} \circleddash \boxed{R(x',y)}\right) & \mathsf{L\forall i} \\ \to & \forall y.\forall x'.\top & \mathsf{id} \\ \to^* & \top & \end{array}$$

But the converse situation is not provable; the system will refuse the following DnD:

$$\forall x.\exists y.\boxed{R(x,y)} \circleddash \exists y'.\forall x'.\boxed{R(x',y')}$$

Indeed, there is no reduction path starting from this DnD ending with the id rule. This can be detected by the system because the unification of $R(x, y)$ and $R(x', y')$ here results in a cycle in the instantiations of variables[7]. The system thus refuses this action.

7: We will come back to this in Subsection 3.2.2. Also notice that this example requires to use full (first-order) unification, not only matching.

2.5.5. Conditional rewriting

The example given in Section 2.4, although very simple, already combines the rules for equality and for quantifiers. When also using implication, one obtains naturally some form of *conditional rewriting*. To take another simple example, suppose we have a hypothesis of the form:

$$\forall x. x \neq 0 \supset f(x) = g(x)$$

We can use this hypothesis to replace a subterm $f(t)$ by $g(t)$, which will generate a side-condition $t \neq 0$:

$$\begin{array}{lll}
 & \forall x. x \neq 0 \supset \boxed{f(x)} = g(x) \circledgtr A\left(\boxed{f(t)}\right) & \\
\rightarrow & t \neq 0 \supset \boxed{f(t)} = g(t) \circledgtr A\left(\boxed{f(t)}\right) & \mathsf{L}\forall\mathsf{i} \\
\rightarrow & t \neq 0 \wedge \left(\boxed{f(t)} = g(t) \circledgtr A\left(\boxed{f(t)}\right)\right) & \mathsf{L}{\supset}_2 \\
\rightarrow & t \neq 0 \wedge A(g(t)) & \mathsf{L}{=}_1
\end{array}$$

One could similarly do such a rewrite in a hypothesis. Furthermore, the conditional rewrite can also be performed under quantifiers; for instance:

$$\begin{array}{lll}
 & \forall x. x \neq 0 \supset \boxed{f(x)} = g(x) \circledgtr \exists y. A\left(\boxed{f(y)}\right) & \mathsf{R}\exists\mathsf{s} \\
\rightarrow & \exists y. \left(\forall x. x \neq 0 \supset \boxed{f(x)} = g(x) \circledgtr A\left(\boxed{f(y)}\right)\right) & \mathsf{L}\forall\mathsf{i} \\
\rightarrow & \exists y. \left(y \neq 0 \wedge \left(\boxed{f(y)} = g(y) \circledgtr A\left(\boxed{f(y)}\right)\right)\right) & \mathsf{L}{\supset}_2 \\
\rightarrow & \exists y. (y \neq 0 \wedge A(g(t))) & \mathsf{L}{=}_1
\end{array}$$

2.6. Related works

Window inference We have already mentioned Proof-by-Pointing, which was part of the CtCoq and Pcoq efforts [4] to design a graphical user interface for the `Coq` proof assistant. Another contemporary line of work was the one based on *window inference*, also mentioned in Section 1.4. In [216], window inference is described as a general proof-theoretical framework, which aims to accomodate for the pervasive use of *equivalence transformations* throughout mathematics and computer science.

[4]: Amerkad et al. (2001), *Mathematics and Proof Presentation in Pcoq*

[216]: Robinson et al. (1993), 'Formalizing a Hierarchical Structure of Practical Mathematical Reasoning'

Window inference has been used both for general-purpose logics like HOL [105], and in more specialized settings like program refinement [106]. It naturally lends itself to integration in a graphical user interface [146, 155], where the user can *focus* on a subexpression by clicking on it. One is then presented with a new *graphical* window, holding the selected expression as well as an extended set of hypotheses exposing information inferrable from the context of the expression. The user can pick from a list of valid transformations to be applied to the expression, before closing

[105]: Grundy (1991), 'Window Inference In The HOL System'

[106]: Grundy (1992), 'A Window Inference Tool for Refinement'

[146]: Långbacka et al. (1995), 'TkWinHOL'

[155]: Lüth et al. (2000), 'TAS — A Generic Window Inference System'

the window. This propagates the transformations to the parent window by replacing the old subexpression by the new one, without modifying the surrounding context.

This process is quite reminiscent of the rewriting produced by our DnD actions. One key difference is that window inference rules can be applied stepwise, while we choose to hide the sequence of rules that justifies a DnD. The window inference approach gives to the user a precise control of the transformations to be performed and thus could inspire interesting extensions of our work.

Other gestural proving interfaces There are other proving interfaces which include drag-and-drop features. Two of them are the KeY Prover [3] and TAS [155]. TAS is a window inference system tailored for program refinement, and uses DnD actions between an expression and a transformation, in order to apply the latter to the former. As for the KeY Prover, its usage of DnD overlaps only a very small portion of usecases that we hinted at in Section 2.3, namely the instantiation of quantifiers with objects.

[3]: Ahrendt et al. (2016), 'Using the KeY Prover'

[155]: Lüth et al. (2000), 'TAS — A Generic Window Inference System'

We can also mention the recent work of Zhan et al. [262]. They share with us the vision of a proof assistant mainly driven by gestural actions, which requires far less textual inputs from the user. However, they only consider point-and-click actions, and rely on a text-heavy presentation at two levels:

[262]: Zhan et al. (2019), 'Design of Point-and-Click User Interfaces for Proof Assistants'

1. the proof state, which is a structured proof text in the style of `Isar` [185];
2. the proof commands, which can only be performed through choices in textual menus.

[185]: Nipkow (2002), 'Structured Proofs in Isar/HOL'

Explicit proof objects Finally let us mention various recent implementations proposing various ways to construct proofs graphically: Building Blocks [150], the Incredible Proof Machine [27], Logitext[8] and Click & coLLecT [34]. In particular, Logitext and Click & coLLecT exploit the same idea of associating click actions on head connectives to inference rules in sequent calculus. But these systems focus more on explicating the proof object than on making its construction easier.

[150]: Lerner et al. (2015), 'Polymorphic Blocks: Formalism-Inspired UI for Structured Connectors'

[27]: Breitner (2016), 'Visual Theorem Proving with the Incredible Proof Machine'

8: `http://logitext.mit.edu/main`

[34]: Callies et al. (2021), 'Click and coLLecT An Interactive Linear Logic Prover'

Backward

$$
\begin{array}{rcll}
A \ogreaterthan A & \to & \top & \mathsf{id} \\
t = u \ogreaterthan A\{t/x\} & \to & A\{u/x\} & \mathsf{L}{=}_1 \\
t = u \ogreaterthan A\{u/x\} & \to & A\{t/x\} & \mathsf{L}{=}_2 \\[1ex]
(B \wedge C) \ogreaterthan A & \to & B \ogreaterthan A & \mathsf{L}\wedge_1 \\
(C \wedge B) \ogreaterthan A & \to & B \ogreaterthan A & \mathsf{L}\wedge_2 \\
A \ogreaterthan (B \wedge C) & \to & (A \ogreaterthan B) \wedge C & \mathsf{R}\wedge_1 \\
A \ogreaterthan (C \wedge B) & \to & C \wedge (A \ogreaterthan B) & \mathsf{R}\wedge_2 \\[1ex]
(B \vee C) \ogreaterthan A & \to & (B \ogreaterthan A) \wedge (C \supset A) & \mathsf{L}\vee_1{}^* \\
(C \vee B) \ogreaterthan A & \to & (C \supset A) \wedge (B \ogreaterthan A) & \mathsf{L}\vee_2{}^* \\
A \ogreaterthan (B \vee C) & \to & (A \ogreaterthan B) \vee C & \mathsf{R}\vee_1 \\
A \ogreaterthan (C \vee B) & \to & C \vee (A \ogreaterthan B) & \mathsf{R}\vee_2 \\[1ex]
(C \supset B) \ogreaterthan A & \to & C \wedge (B \ogreaterthan A) & \mathsf{L}\supset_2 \\
A \ogreaterthan (B \supset C) & \to & (A \circledast B) \supset C & \mathsf{R}\supset_1{}^* \\
A \ogreaterthan (C \supset B) & \to & C \supset (A \ogreaterthan B) & \mathsf{R}\supset_2{}^* \\[1ex]
(\forall x.B) \ogreaterthan A & \to & B\{x/t\} \ogreaterthan A & \mathsf{L}\forall\mathsf{i} \\
(\forall x.B) \ogreaterthan A & \to & \exists x.(B \ogreaterthan A) & \mathsf{L}\forall\mathsf{s} \\
A \ogreaterthan (\forall x.B) & \to & \forall x.(A \ogreaterthan B) & \mathsf{R}\forall\mathsf{s}^* \\[1ex]
(\exists x.B) \ogreaterthan A & \to & \forall x.(B \ogreaterthan A) & \mathsf{L}\exists\mathsf{s}^* \\
A \ogreaterthan (\exists x.B) & \to & A \ogreaterthan B\{x/t\} & \mathsf{R}\exists\mathsf{i} \\
A \ogreaterthan (\exists x.B) & \to & \exists x.(A \ogreaterthan B) & \mathsf{R}\exists\mathsf{s}
\end{array}
$$

Forward

$$
\begin{array}{rcll}
A\{t/x\} \circledast (t = u) & \to & A\{u/x\} & \mathsf{F}{=}_1 \\
A\{u/x\} \circledast (t = u) & \to & A\{t/x\} & \mathsf{F}{=}_2 \\[1ex]
A \circledast (B \wedge C) & \to & A \circledast B & \mathsf{F}\wedge_1 \\
A \circledast (C \wedge B) & \to & A \circledast B & \mathsf{F}\wedge_2 \\[1ex]
A \circledast (B \vee C) & \to & (A \circledast B) \vee C & \mathsf{F}\vee_1 \\
A \circledast (C \vee B) & \to & C \vee (A \circledast B) & \mathsf{F}\vee_2 \\[1ex]
A \circledast (B \supset C) & \to & (A \ogreaterthan B) \supset C & \mathsf{F}\supset_1 \\
A \circledast (C \supset B) & \to & C \supset (A \circledast B) & \mathsf{F}\supset_2 \\[1ex]
A \circledast (\forall x.B) & \to & A \circledast B\{x/t\} & \mathsf{F}\forall\mathsf{i} \\
A \circledast (\forall x.B) & \to & \forall x.(A \circledast B) & \mathsf{F}\forall\mathsf{s} \\[1ex]
A \circledast (\exists x.B) & \to & \exists x.(A \circledast B) & \mathsf{F}\exists\mathsf{s}^* \\[1ex]
B \circledast A & \to & A \circledast B & \mathsf{Fcomm}
\end{array}
$$

In the rules $\{\mathsf{L}\forall\mathsf{s}, \mathsf{L}\exists\mathsf{s}, \mathsf{R}\forall\mathsf{s}, \mathsf{R}\exists\mathsf{s}, \mathsf{F}\forall\mathsf{s}, \mathsf{F}\exists\mathsf{s}\}$, x is not free in A.

Figure 2.4.: Linking rules

Units

$$
\begin{array}{lrcll}
\langle \circ, \dagger \rangle \in \{\langle \wedge, \top \rangle, \langle \vee, \bot \rangle, \langle \supset, \top \rangle\} & \dagger \circ A & \to & A & \mathsf{neul} \\
\langle \circ, \dagger \rangle \in \{\langle \wedge, \top \rangle, \langle \vee, \bot \rangle\} & A \circ \dagger & \to & A & \mathsf{neur} \\
\langle \circ, \dagger \rangle \in \{\langle \wedge, \bot \rangle, \langle \vee, \top \rangle\} & \dagger \circ A & \to & \dagger & \mathsf{absl} \\
\langle \circ, \dagger \rangle \in \{\langle \wedge, \bot \rangle, \langle \vee, \top \rangle, \langle \supset, \top \rangle\} & A \circ \dagger & \to & \dagger & \mathsf{absr} \\
\langle \diamond, \dagger \rangle \in \{\langle \forall, \top \rangle, \langle \exists, \bot \rangle\} & \diamond x.\dagger & \to & \dagger & \mathsf{absq} \\
 & \bot \supset A & \to & \top & \mathsf{efq}
\end{array}
$$

Figure 2.5.: Unit elimination rules

Subformula Linking 3.

> Logic and mathematics seem to be the only domains where self-evidence manages to rise above triviality; and this it does, in those domains, by a linking of self-evidence on to self-evidence in the chain reaction known as proof.
>
> **Willard van Orman Quine**, *The Web of Belief*, 1978

3.1 Linkages **38**
3.2 Validity **40**
3.2.1 Polarity 41
3.2.2 Identity 42
3.3 Describing DnD actions **43**
3.4 Soundness **44**
3.5 Productivity **47**
3.6 Focusing **51**
3.7 Completeness **52**
3.8 Related works **58**

In this chapter, we engage in a thorough analysis of the logical semantics of DnD actions, which were introduced informally through examples in Chapter 2. We do this mainly from the formal perspective of deep inference proof theory, following the original work of K. Chaudhuri on *subformula linking* [37]. But we always keep in mind the intended application to proof assistants, by motivating various design choices — actual or prospective — as ways to improve the *user experience* (UX) of interactive proof building.

The chapter is organized as follows: Section 3.1 introduces the notions of context and polarity, and explains how DnD actions are specified by the user interactively through schemas called *linkages*. Section 3.2 explains how one can identify a subset of linkages that guarantees a productivity property on DnD actions. Section 3.3 describes the overall structure of how linkages translate into logical steps, and Section 3.6 discusses some subtleties of this translation that are related to the concept of *focusing* in automated proof search. Section 3.4 shows that the logical steps are sound, and Section 3.5 states and proves formally the productivity property. Finally, Section 3.7 shows how DnD actions can be turned into a complete deductive system without any need for click actions.

3.1. Linkages

Like most rewriting systems on terms (that is, tree-shaped data), the rewriting rules of Figure 2.4 and Figure 2.5 apply at any depth inside formulas. However logically, the shape of the *context* in which this rewriting occurs can provide important information, either to ensure soundness of the performed transformation (Section 3.4, Section 3.7), or to understand the status of quantified variables (Subsection 3.2.2).

Definition 3.1.1 (Context) *A* context*, written $A\square$, is a proposition containing exactly one occurrence of a special formula written $\square$, called its* hole*. Given another proposition B, we write $A\boxed{B}$ for the proposition obtained by replacing $\square$ in $A\square$ by B. Note that this replacement is not a* substitution *because it allows variable capture. For instance $\forall x.\boxed{P(x)}$ is the proposition $\forall x.P(x)$.*

Definition 3.1.2 (Path) *A path is a proposition where one subformula has been selected. Formally, a path is a pair $(A\square, B)$ formed by one context and one proposition:*

- *$A\square$ is called the* context *of the path,*
- *B is called the* selection *of the path.*

The path $(A\square, B)$ can be viewed as the proposition $A\boxed{B}$. For readability, we will generally also write $A\boxed{B}$ for the path $(A\square, B)$.

Definition 3.1.3 (Inversions) *Given a context $A\square$, the number of inversions in $A\square$, written $\mathsf{inv}(A\square)$, is the number of subterms of $A\square$ which are of the form $C\square \supset D$; that is the number of times the hole is on the left-hand side of an implication. For instance:*

$$\begin{aligned} \mathsf{inv}(D \land \square) &= 0 \\ \mathsf{inv}((D \land \square) \supset E) &= 1 \\ \mathsf{inv}((\square \supset C) \supset D) &= 2 \end{aligned}$$

Definition 3.1.4 (Polarity) *We will write $A^{+}\square$ to specify that a context is positive, meaning that $\mathsf{inv}(A^{+}\square)$ is even. Symmetrically, $A^{-}\square$ will be used for negative contexts, meaning that $\mathsf{inv}(A^{-}\square)$ is odd.*

In addition to the items involved, every DnD action specifies the *selection* of a subterm in each item, which can be expressed formally as a path. We call *linkage* the combined data of the two items together with the selection, since the intent is to *link* the subterms to make them interact in some way.

Remark 3.1.1 In this thesis we only consider linkages between two subterms. But as noted in Section 2.4, rewriting equalities is an example of action that can benefit from allowing multiple selections[1].

Each kind of DnD action is mapped in the system to a specific form of linkage, which is designed to hold all the information necessary for the correct execution of the action. In this way the system can automatically search for linkages of a certain form, and propose to the user all well-defined actions associated to these linkages.

Remark 3.1.2 In the future, one can imagine several DnD actions associated to a given linkage. In this case, the user could be queried to choose the action to be performed (typically with a pop-up menu). However with the actions considered in this thesis, such ambiguities never arise.

On the "items axis", we already distinguished between backward and forward linkages, written respectively $A \circleddash B$ and $A \circledast B$. If the items types are unspecified, we will write $A \mathbin{@} B$.

1: A restricted kind of multi-occurrence rewrite is already available in the standalone version of `Actema`: one needs to enter *selection mode*, by either toggling the dedicated button, or holding down the `shift` key. Then one can click successively on all occurrences of a term t that are to be rewritten, in order to add them to the selection. To perform the rewriting to some other term u, the last step is to drag an equality hypothesis $t = u$ (or $u = t$) and drop it on any item holding one of the selected occurrences of t.

Using the "selection axis", we can specify a further distinction that was informal up to now: that of *logical* action and *rewrite* action.

- *Logical linkages* link two subformulas: they have the form $B\boxed{A} @ C\boxed{A'}$.
- *Rewrite linkages* link one side of an equality with a first-order term. Using liberally the notations from Definition 3.1.1 and Definition 3.1.2, they thus have the form

$$B\boxed{\boxed{t} = u} @ C\boxed{t'} \quad \left(\text{or symmetrically } B\boxed{u = \boxed{t}} @ C\boxed{t'}\right)$$

By forgetting the information of which subterms are selected, one can see any linkage as a formula whose topmost connective is an interaction operator $@ \in \{\circleddash, \circledast\}$. Then it is natural to view linkages as the redexes of the rewriting rules of Figure 2.4, although from the user's standpoint linkages only happen at the top-level[2].

2: In fact this is more of a limitation of Actema's current interface: one cannot link two subterms that live in the same item, because dragging actions can only be performed on *entire* items. But in the original formulation and implementation of subformula linking [37], linkages can be created between arbitrary subformulas. We come back to this issue in Section 3.7.

3.2. Validity

In the original formulation of subformula linking [37], a semantics is associated to every logical linkage, even when the selected subformulas A and A' are not unifiable. This is made possible by the addition of so-called *release* rules[3], which simply turn interaction operators into their associated logical connective. In our setting this would give the rewriting rules of Figure 3.1. However in this work we opt for a different approach: instead we define a *validity criterion* on linkages, which guarantees that they give rise to the behaviors described in the previous sections. The criterion tackles two issues:

$$\begin{array}{lcll} A \circleddash B & \rightarrow & A \supset B & \mathsf{Brel} \\ A \circledast B & \rightarrow & A \wedge B & \mathsf{Frel} \end{array}$$

Figure 3.1.: Release rules

[37]: Chaudhuri (2013), 'Subformula Linking as an Interaction Method'

3: A terminology coming from the line of works on *focusing* in proof theory [5], see also Section 3.6.

- **Polarity:** the selected subterms must have opposite *polarities*, so that the *negative* subterm justifies the *positive* one;
- **Identity:** the selected subterms must be *unifiable*, so that after instantiating some quantifiers in their context they can interact through the id rule or the equality rules $\mathsf{L}{=_{\mathsf{i}}}$, $\mathsf{F}{=_{\mathsf{i}}}$.

One benefit of using this criterion is that it filters out all linkages whose semantics rely on release rules, capturing intuitively a notion of *productivity*: instead of just moving around subformulas, we know for sure that some "simplification" occurs, either a justification with the id rule on logical linkages, or a rewriting with the equality rules on rewrite linkages. This will be stated more formally in Section 3.5.

Validity is very useful to support the *suggestion* mechanism implemented in `Actema`. The idea is that when the user starts dragging an item, this indicates to the system that she wants to perform a DnD action involving subterms of this item. Then the system can suggest such possible actions by highlighting subterms in the goal which form a valid linkage with the dragged item. Typically in the example of Figure 2.1, dragging the hypothesis $\forall x.\mathsf{Human}(x) \supset \mathsf{Mortal}(x)$ will have the effect of highlighting exactly $\mathsf{Mortal}(\mathsf{Socrates})$ in the conclusion and $\mathsf{Human}(\mathsf{Socrates})$ in the other hypothesis as possible drop targets. In this case this corresponds to all subterms in the goal which are not contained in the dragged item. But if one were to drag the Human(Socrates) hypothesis instead, then only the subterm Human(x) in the other hypothesis would be suggested as a drop target. This could not work with the "release" semantics mentioned earlier, i.e. all subterms would again be highlighted, providing no useful information to the user.

We believe that in more complex situations, this filtering can be quite helpful to guide the user towards the right path to follow in their reasoning. Although non-trivial arguments are often based on "guessing" the right value or lemma to be used, a large part of mathematical reasoning also consists in "connecting the dots" with information already at hand. Our DnD actions capture this metaphor quite directly, and thus shall be especially useful to beginners unfamiliar with proving, who often show difficulties in understanding how to build a proof from scratch. More generally, proof assistants have the potential to provide a well-defined and rigorous methodology in the art of crafting proofs, in the same way that we have been teaching precise algorithms for solving equations in calculus classes for centuries. Having a graphical interface that makes this methodology more intuitive and discoverable is the main goal of this work, and the notion of valid linkage seems to be a good candidate as a core principle for such a methodology.

3.2.1. Polarity

The restrictions on polarities are captured formally by the following condition:

Condition 3.2.1 (Polarity) The following must be true for a logical linkage $B\boxed{A} @ D\boxed{A'}$ to be valid:

1. the parity of $\mathsf{inv}(B\square)$ is:
   a) the same as $\mathsf{inv}(D\square)$ if $@ = ⧁$;
   b) the opposite of $\mathsf{inv}(D\square)$ if $@ = \circledast$;
2. if $@ = ⧁$ and $\mathsf{inv}(D\square) = 0$, then $\mathsf{inv}(B\square) = 0$.

The following must be true for a rewrite linkage $B\boxed{t} @ D\boxed{t'}$ to be valid:

1. if $B\square$ holds the equality, then it must be:
   a) positive if $@ = ⧁$;
   b) positive if $@ = \circledast$;
2. if $D\square$ holds the equality, then it must be:
   a) negative if $@ = ⧁$;
   b) positive if $@ = \circledast$.

One understands that for rewrite linkages, this simply guarantees that the equality is in negative position. For logical linkages, Clause 1 ensures that the selected subformulas have opposite polarities, and Clause 2 ensures that the linkage makes sense in our *intuitionistic* setting. Indeed one could imagine the following behavior in classical logic:

$$\left(\boxed{A} \supset B\right) \supset C \ogreaterthan \boxed{A} \quad \to^* \quad C \supset A$$

which gives a proof of Peirce's law when replacing C with A. We will come back to this example in Chapter 5, but for now we can just remark that there is no way to handle it with the rules of Figure 2.4 because we lack a rule for redexes of the form $B\boxed{A} \supset C \ogreaterthan D\boxed{A'}$.

3.2.2. Identity

A context binds variables in the selected proposition. These variables will be unifiable or not depending upon: (1) the nature of the quantifier ($\forall$ or $\exists$), (2) whether they occur in a hypothesis or a conclusion, and (3) whether they occur on the left-hand of an (odd number of) implication(s). Therefore, we start by splitting the list of variables bound by a context in two parts.

Definition 3.2.1 (Positive and negative variables) *Given a context $A\square$ seen as a tree, one can always start from the root and traverse the branch of $A\square$ that leads to its hole $\square$. We write $l(A\square)$ the list of all variables quantified along the way. This list is ordered, the variables closer to the root coming first.*

$l(A\square)$ can be seen as the interleaving of two sublists $l^+(A\square)$ and $l^-(A\square)$ of positively *and* negatively unifiable *variables, in the following precise sense: $x \in l^+(A\square)$ (resp. $x \in l^-(A\square)$) iff there are contexts $B\square$, $C^+\square$ and $D^-\square$ such that $A\square$ is either $C^+\boxed{\exists x.B\square}$ or $D^-\boxed{\forall x.B\square}$ (resp. $D^-\boxed{\exists x.B\square}$ or $C^+\boxed{\forall x.B\square}$).*

For instance, if $A\square \equiv \forall x.\exists y.(B \wedge ((\exists x'.\forall y'.\square) \supset \forall z.C))$, then we have:

$$l(A\square) = [x, y, x', y'] \qquad l^+(A\square) = [y, y'] \qquad l^-(A\square) = [x, x']$$

Definition 3.2.2 (Unifiable variables) *The set $\mathrm{U}(\mathscr{L})$ of unifiable variables of a linkage $\mathscr{L} \equiv B\boxed{A} @ C\boxed{A'}$ is:*

- *$l^-(B\square) \cup l^+(C\square)$ if $@$ is $\ogreaterthan$, and*
- *$l^-(B\square) \cup l^-(C\square)$ if $@$ is $\circledast$.*

The following notions of substitution and unification are the usual ones and we do not go into details:

Definition 3.2.3 (Substitution) *A substitution is a mapping from vari-*

ables to terms such that $\{x \mid \sigma(x) \not\equiv x\}$ *is finite; we call this set the* domain *of* σ*. When* $\sigma(x) \equiv x$ *we say that* x *is* not instantiated *by* σ*. Given a proposition* A *and a substitution* σ*, we write* $\sigma(A)$ *for the* application *of* σ *to* A *in the usual way.*

Definition 3.2.4 (Unification) *Given two propositions* A *and* A' *and a list of variables* l*, we say that a substitution* σ *unifies* A *and* A' *over* l *when* $\sigma(A) \equiv \sigma(A')$ *and the domain of* σ *is a subset of* l*.*

If such a substitution exists, we say that A *and* A' *are unifiable over* l*.*

Given A, A' and l, the well-known unification algorithm decides whether A and A' are unifiable over l and constructs the substitution when it exists [161].

[161]: Martelli et al. (1982), 'An Efficient Unification Algorithm'

Condition 3.2.2 (Identity) For a linkage $\mathscr{L} := B\boxed{A} @ C\boxed{A'}$ to be valid, the following must be true:

1. There exists a substitution σ which unifies A and A' over $\mathrm{U}(\mathscr{L})$.
2. Furthermore, the unification respects the order over the variables. More precisely, we request that there exists a list l which is an interleaving of $l(B\square)$ and $l(C\square)$ such that, given a unifiable variable x in the domain of σ, all variables occuring in $\sigma(x)$ are placed before x in l:
$$\forall y \in \mathrm{fv}(\sigma(x)) \cap (l(B\square) \cup l(C\square)),\ y <_l x.$$

The last condition ensures acyclicity and will prohibit invalid linkages as described in Section 2.5.4. More precisely, the list l specifies the order in which the quantifiers will be treated in the proof construction.

Finally we can state the full validity criterion for linkages:

Definition 3.2.5 (Valid linkage) *We say that a linkage* $\mathscr{L}$ *is valid if it satisfies Conditions 3.2.1 and 3.2.2.*

One can check that all the examples given up to here were based on valid linkages.

3.3. Describing DnD actions

We are now equipped to specify how logical and rewrite linkages translate deterministically to the backward and forward proof steps shown in all examples.

First some remarks can be made about the rewriting rules of Figure 2.4:

- The set of rewriting rules is obviously non-confluent.
- It is also terminating, because the number of connectives or quantifiers under $\circledast$ or $\circledgt$ decreases[4].

As for the rules of Figure 2.5, they are both terminating *and* confluent. Indeed, they define a function that eliminates redundant occurrences of the units $\top$ and $\bot$.

Here is a high-level overview of the complete procedure followed to generate a proof step:

1. **Selection:** the user selects two subterms in two items of the current goal;
2. **Linkage:** this either gives rise to a logical linkage $B\boxed{A} @ C\boxed{A'}$ (resp. a rewrite linkage $B\boxed{\boxed{t} = u} @ C\boxed{t'}$), or does not correspond to a known form of linkage. In this case the procedure stops here, and the system does not propose any action to the user;
3. **Validity:** the system verifies that the linkage is *valid*, by performing successively the following checks:
   a) **Polarity:** the linkage must satisfy Condition 3.2.1;
   b) **Unification:** the selected subterms A and A' (resp. t and t') must be unifiable, yielding a substitution σ;
   c) **Dependencies:** the substitution σ must satisfy Condition 3.2.2.

   The procedure stops if it fails at any of the above checks;
4. **Linking:** the system then chooses a rewriting starting from the linkage. Thanks to Theorem 3.5.2, this rewriting always ends with a proposition of the form $D\boxed{\sigma(A) \circledgt \sigma(A')}$

$$\left(\text{resp. } D\boxed{\boxed{\sigma(t)} = u @ C_0\boxed{\sigma(t')}}\right)$$

5. **Interaction:** thus one can apply the id rule (resp. an equality rule in $\{\mathsf{L}{=}_1, \mathsf{L}{=}_2, \mathsf{F}{=}_1, \mathsf{F}{=}_2\}$);
6. **Unit elimination:** in the case of a logical action, this creates an occurrence of $\top$, which is eliminated using the rules of Figure 2.5;
7. **Goal modification:** the two previous steps produced a formula E. In the case of a forward linkage, a hypothesis E is added to the goal; in the case of a backward linkage, the goal's conclusion becomes E. In both cases, the logical soundness is guaranteed by Theorem 3.4.1.

3.4. Soundness

All examples up to now followed the scheme for DnD actions sketched in Section 2.2:

4: Except for the Fcomm rule which is just meant to make the $\circledast$ operator commutative; formally, the only infinite reduction paths end with an infinite iteration of Fcomm.

- Given a blue item A and a red item B, backward proof steps produce a new conclusion C by applying a sequence of rewriting rules $A \circleddash B \to^* C$.
- Given two blue items A and B, forward proof steps produce a new hypothesis C by applying a sequence of rewriting rules $A \circledast B \to^* C$.

Thus for such actions to be logically sound, we have to make sure that our rewriting system satisfies the following property:

Theorem 3.4.1 (Soundness)

- *If $A \circleddash B \to^* C$, then $A, C \Rightarrow B$.*
- *If $A \circledast B \to^* C$, then $A, B \Rightarrow C$.*

We will need to reason inductively on contexts, and more precisely on the *depth* of their hole:

Definition 3.4.1 (Depth) *The depth $|A\square|$ of a context $A\square$ is defined recursively by:*

$$\begin{aligned} |\square| &= 0 \\ |A\square \circ B| = |B \circ A\square| &= |A\square| + 1 && \textit{for } \circ \in \{\land, \lor, \supset\} \\ |\diamond x.A\square| &= |A\square| + 1 && \textit{for } \diamond \in \{\forall, \exists\} \end{aligned}$$

The following simple covariance and contravariance property will be used extensively later on:

Lemma 3.4.2 (Variance) *If $\Gamma, A \Rightarrow B$, then $\Gamma, C^+\boxed{A} \Rightarrow C^+\boxed{B}$ and $\Gamma, D^-\boxed{B} \Rightarrow D^-\boxed{A}$.*

Proof. By induction on $|C^+\square|$ and $|D^-\square|$. □

For each rule, interpreting $\circleddash$ as $\supset$ and $\circledast$ as $\land$ is enough to show that the rule satisfies Theorem 3.4.1 locally. Formally, we can define a mapping from formulas containing interaction operators to usual formulas where they have been replaced by their interpretation:

Definition 3.4.2 (Interpretation of interaction operators) *The mapping $\lfloor - \rfloor$ is defined recursively as follows:*

$$\begin{aligned} \lfloor A \circleddash B \rfloor &= \lfloor A \rfloor \supset \lfloor B \rfloor \\ \lfloor A \circledast B \rfloor &= \lfloor A \rfloor \land \lfloor B \rfloor \\ \lfloor A \circ B \rfloor &= \lfloor A \rfloor \circ \lfloor B \rfloor && \textit{for } \circ \in \{\land, \lor, \supset\} \\ \lfloor \diamond x.A \rfloor &= \diamond x.\lfloor A \rfloor && \textit{for } \diamond \in \{\forall, \exists\} \\ \lfloor \dagger \rfloor &= \dagger && \textit{for } \dagger \in \{\top, \bot\} \end{aligned}$$

$$\lfloor a \rfloor = a \qquad\qquad \textit{for } a \textit{ atomic}$$

For rewritings taking place deeper inside a proposition however, we need to consider the polarity of their context.

Lemma 3.4.3 (Local soundness)

- *If* $C^{+}\boxed{A \obslash B} \to D$ *then* $\lfloor D \rfloor \Rightarrow C^{+}\boxed{A \supset B}$.
- *If* $C^{-}\boxed{A \obslash B} \to D$ *then* $C^{-}\boxed{A \supset B} \Rightarrow \lfloor D \rfloor$.
- *If* $C^{+}\boxed{A \circledast B} \to D$ *then* $C^{+}\boxed{A \wedge B} \Rightarrow \lfloor D \rfloor$.
- *If* $C^{-}\boxed{A \circledast B} \to D$ *then* $\lfloor D \rfloor \Rightarrow C^{-}\boxed{A \wedge B}$.

Proof. First notice that D is necessarily of the form $C\boxed{D_0}$ where $A \mathbin{@} B \to D_0$[5]. Then by careful analysis of each rule, it is straightforward to show that $\lfloor D_0 \rfloor \Rightarrow A \supset B$ if $@ = \obslash$ or $A \wedge B \Rightarrow \lfloor D_0 \rfloor$ if $@ = \circledast$. We can conclude in each case by applying Lemma 3.4.2. □

5: Indeed an implicit assumption in this section, which is preserved by all the rules, is that a formula contains at most one interaction operator. Thus if $C\boxed{A \mathbin{@} B} \to D$, the only possible redex is $A \mathbin{@} B$.

Remark 3.4.1 For some rules, like $\mathsf{R}{\supset}_1$, the left-hand and right-hand propositions are equivalent:

$$A \supset B \supset C \quad\Longleftrightarrow\quad A \wedge B \supset C$$

These rules are thus *invertible* and their names are tagged by *. This point will be relevant in Section 3.6.

An easy but important technical point is that rewriting rules preserve the polarity of contexts around redexes, in the following precise sense:

Fact 3.4.1 (Polarity preservation)

- If $C\boxed{A \obslash B} \to C'\boxed{A' \obslash B'}$ (resp. $C\boxed{A \circledast B} \to C'\boxed{A' \circledast B'}$) then $C\square$ and $C'\square$ have the same polarity.
- If $C\boxed{A \obslash B} \to C'\boxed{A' \circledast B'}$ (resp. $C\boxed{A \circledast B} \to C'\boxed{A' \obslash B'}$) then $C\square$ and $C'\square$ have opposite polarities.

Combining Lemma 3.4.3 and Fact 3.4.1, we obtain the central soundness result about the rewriting rules:

Lemma 3.4.4 (Contextual soundness)

- *If* $C^{+}\boxed{A \obslash B} \to^{*} D$ *then* $\lfloor D \rfloor \Rightarrow C^{+}\boxed{A \supset B}$.
- *If* $C^{-}\boxed{A \obslash B} \to^{*} D$ *then* $C^{-}\boxed{A \supset B} \Rightarrow \lfloor D \rfloor$.
- *If* $C^{+}\boxed{A \circledast B} \to^{*} D$ *then* $C^{+}\boxed{A \wedge B} \Rightarrow \lfloor D \rfloor$.

▸ *If* $C^-\boxed{A \circledast B} \to^* D$ *then* $\lfloor D \rfloor \Rightarrow C^-\boxed{A \wedge B}$.

Proof. By induction on the length of the derivation. The base case is trivial by reflexivity of entailment. We give the proof for the first statement in the list, other cases work similarly. We can assume without loss of generality that the derivation has the following shape:

$$C^+\boxed{A \circleddash B} \to C'\boxed{A' @ B'} \to^* D$$

Then we reason by case on the interaction operator @:

- $@ = \circleddash$: by Fact 3.4.1, C' must be positive. Therefore by induction hypothesis $\lfloor D \rfloor \Rightarrow C'\boxed{A' \supset B'}$. By Lemma 3.4.3 we have $C'\boxed{A' \supset B'} \Rightarrow C^+\boxed{A \supset B}$. Thus by transitivity $\lfloor D \rfloor \Rightarrow C^+\boxed{A \supset B}$.
- $@ = \circledast$: by Fact 3.4.1, C' must be negative. Therefore by induction hypothesis $\lfloor D \rfloor \Rightarrow C'\boxed{A' \wedge B'}$. By Lemma 3.4.3 we have $C'\boxed{A' \wedge B'} \Rightarrow C^+\boxed{A \supset B}$. Thus by transitivity $\lfloor D \rfloor \Rightarrow C^+\boxed{A \supset B}$.

□

Finally, soundness (Theorem 3.4.1) is obtained as the special case where the rewriting starts in the (positive) empty context.

3.5. Productivity

An important property of the linking step 4 is that there is always a rewriting sequence that brings together the selected subterms, which ensures that we can proceed to the interaction step 5.

Because the rewriting rules are terminating, the important point is to show that one can always apply a rule until one reaches an interaction rule on the selected subterms. In other words, it is possible to find at least one rule which preserves Conditions 3.2.1 and 3.2.2 on linkages:

Lemma 3.5.1 (Valid Progress) *If a linkage* $\mathscr{L} \equiv C\boxed{A} @ C'\boxed{A'}$ *(resp.* $C\boxed{t} @ C'\boxed{t'}$*) is valid, then either:*

1. $\mathscr{L} \equiv \boxed{A} \circleddash \boxed{A}$ *(resp.* $C\square \in \{\square = u, u = \square\}$ *for some* u *and* $t \equiv t'$*);*
2. *or* $\mathscr{L} \to E\boxed{\mathscr{L}'}$ *for some* $E\square, \mathscr{L}'$ *with* $\mathscr{L}'$ *valid.*

A detailed proof is given hereafter for the case of logical linkages. It is not fundamentally difficult, but understandably verbose. The two main points are:

- The rules involving a connective always preserve validity.
- When one can apply a rule involving a quantifier $\forall x$ (resp. $\exists x$), one checks whether the substitution produced by unification (Condition

3.2.2) instantiates x or not. In the first case one performs the instantiation rule L∀i or F∀i (resp. R∃i); in the second case the corresponding switch rule in {L∀s, R∀s, F∀s} (resp. {L∃s, R∃s, F∃s}).

Proof. Let $\mathscr{L} \equiv B\boxed{A} @ C\boxed{A'}$ be a valid linkage.

1. Suppose $B\square \equiv C\square \equiv \square$. By Condition 3.2.1, we know that a forward linkage cannot verify $(\mathsf{inv}(B\square), \mathsf{inv}(C\square)) = (0, 0)$, thus $\mathscr{L}$ must be a backward linkage. Also $l(B\square)$ and $l(C\square)$ are empty, hence by Condition 3.2.2 A and A' are unified by an empty substitution, which entails that $A \equiv A'$. Therefore we are in the first case where $\mathscr{L} \equiv \boxed{A} \circleddash \boxed{A}$.

2. Otherwise, either $B\square$ or $C\square$ is non-empty. In the following, we show that we can always apply a rewriting rule that produces a new, valid linkage $\mathscr{L}' \equiv B' @' C'$.

   Let σ and l be respectively the substitution and interleaving of the quantified variables of $B\square$ and $C\square$ given by Condition 3.2.2, with l decomposed as $x :: l'$.

   ▸ If x is quantified at the head of either $B\square$ or $C\square$, then we apply the associated quantifier rule:

   Switch rule (L∀s, L∃s, R∀s, R∃s, F∀s, F∃s) Only if x is not in the domain of σ. In forward mode and when $B\square$ binds x, one must first apply the rule Fcomm to put $B\boxed{A}$ on the right of $\circledast$, so that the switch rule is applicable. Now we show that $\mathscr{L}'$ is valid:

   1. $\mathscr{L}'$ satisfies Condition 3.2.1 trivially since none of the switch rules changes the number of inversions.

   2. For each switch rule we can show, using the fact that x is not in the domain of σ, that $\mathsf{U}(\mathscr{L}') = \mathsf{U}(\mathscr{L})$. Since the selected formulas A and A' stay untouched by the rule, we can choose σ as a valid unifier that ranges over $\mathsf{U}(\mathscr{L}')$.

   3. In all switch rules, we have $l(\mathscr{L}') = l'$ because the quantifier of x is moved in the outer context of the linkage. Thus we can just take l' as interleaving, and Condition 3.2.2 will still be verified because l' is a sublist of l.

   Instantiation rule (L∀i, R∃i, F∀i) Only if x is instantiated by σ, using $\sigma(x)$ as witness. Again one might need to apply Fcomm first. Then we check the validity of $\mathscr{L}'$:

   1. $\mathscr{L}'$ satisfies Condition 3.2.1 trivially since none of the instantiation rules changes the number of inversions.

   2. For each instantiation rule we can show, using the fact that x is instantiated by σ, that $\mathsf{U}(\mathscr{L}') = \mathsf{U}(\mathscr{L}) \setminus \{x\}$. Then

we take as unifier σ where the binding for x is removed, written $\sigma \setminus x$.

Now we need to make sure that $\sigma \setminus x$ is indeed a unifier for the selected formulas. We consider only the case where $B\square$ binds x, the proof being exactly symmetric when $C\square$ binds x. Let $B_0\square$ be the direct subcontext of $B\square$, that is $B\square$ without the head quantifier binding x.

First we can assert that $B_0\boxed{A}\{\sigma(x)/x\} \equiv B_0\{\sigma(x)/x\}\boxed{A\{\sigma(x)/x\}}$. Indeed, Clause 2 of Condition 3.2.2 guarantees that for any free variable y of $\sigma(x)$, $y \notin l(B_0\square)$, and thus the above instantiation can propagate safely to A without capture. To convince yourself that $y \notin l(B_0\square)$, suppose the contrary. Then $y \in l(B\square)$, and by Clause 2 y must be placed before x in l. But this is impossible since x is the first element of l!

So we know that the selected formula on the left of $\mathscr{L}'$ is $A\{\sigma(x)/x\}$, while it is still A' on the right. Thus it only remains to show that

$$A\{\sigma(x)/x\}[\sigma \setminus x] \equiv A'[\sigma \setminus x].$$

On the left we have by definition that $A\{\sigma(x)/x\}[\sigma \setminus x] \equiv A[\sigma]$, and on the right we have $A'[\sigma \setminus x] \equiv A'[\sigma]$ because x cannot occur in A' since it is bound in $B_0\square$ (here we rely on the Barendregt convention).

3. In all instantiation rules, we have $l(\mathscr{L}') = l'$ because the quantifier of x is removed by the instantiation. Thus we can again take l' as interleaving.

- If x is not quantified at the head of $B\square$ or $C\square$, then either both heads are propositional connectives, or one is a propositional connective and the other is empty. In both cases we can choose either a rule of the form $\mathsf{L}\circ_i$, $\mathsf{R}\circ_i$ or $\mathsf{F}\circ_i$, where $\circ$ is the connective and i the index of the direct subcontext where A or A' occurs, or the Fcomm rule. Again we check the conditions of Definition 3.2.5:

1. In most rules the number of inversions stays unchanged. The only exceptions are $\mathsf{R}{\supset_1}$ and $\mathsf{F}{\supset_1}$, which decrease the number of inversions of the right context $C\square$ by 1. But since they are also the only rules that change the interaction operator, the truth of Clause 1 is preserved: if the parities were opposite (resp. identical) in $\mathscr{L}$, then $\mathscr{L}$ must be forward (resp. backward). Thus $\mathscr{L}'$ is necessarily backward (resp. forward), and so the parities in $\mathscr{L}'$ must be identical (resp. opposite), which is the case thanks to the inversion decrement.

   For Clause 2, we can distinguish two cases:

   - If $\mathscr{L}$ is backward, then either we apply the $\mathsf{R}{\supset_1}$ rule and

$\mathcal{L}'$ is forward, and thus satisfies Clause 2 trivially; or we apply another backward rule and $\mathcal{L}'$ is backward. Now suppose $\mathsf{inv}(C'\square) = 0$. Then we must have $\mathsf{inv}(C\square) = \mathsf{inv}(C'\square) = 0$ and $\mathsf{inv}(B\square) = \mathsf{inv}(B'\square)$ since all backward rules other than $\mathsf{R}{\supset_1}$ preserve the number of inversions. And because $\mathcal{L}$ satisfies Clause 2 by validity, we can deduce that $\mathsf{inv}(B\square) = 0$, and thus $\mathsf{inv}(B'\square) = 0$.

- If $\mathcal{L}$ is forward, then either we apply a forward rule that is neither $\mathsf{F}{\supset_1}$ nor Fcomm and $\mathcal{L}'$ is forward, and thus satisfies Clause 2 trivially; or we consider applying either $\mathsf{F}{\supset_1}$ of Fcomm. There are three cases:

  * If $\mathsf{inv}(C\square) > 1$, then we can safely apply $\mathsf{F}{\supset_1}$ since we have $\mathsf{inv}(C'\square) = \mathsf{inv}(C\square) - 1 > 0$;

  * If $\mathsf{inv}(C\square) = 0$, then $C\square$ is empty and we are forced to apply Fcomm so that we can apply the forward rule corresponding to the head connective of $B\square$. Then $C\square$ ends up on the left of $\circledast$, thus if we apply $\mathsf{F}{\supset_1}$ for $B\square$ Clause 2 will be satisfied trivially;

  * If $\mathsf{inv}(C\square = 1)$, then either $\mathsf{inv}(B\square) = 0$ and we can safely apply $\mathsf{F}{\supset_1}$ since $\mathsf{inv}(B'\square) = \mathsf{inv}(B\square)$; or $\mathsf{inv}(B\square) > 0$, and we cannot apply $\mathsf{F}{\supset_1}$ because we would end up with $\mathsf{inv}(C'\square) = 0$ and $\mathsf{inv}(B'\square) > 0$, thus violating Clause 2. Hence as in the previous case, we need to apply Fcomm first. Then it cannot be the case that $\mathsf{inv}(B\square) = 1$ because we would have $\mathsf{inv}(B\square) = \mathsf{inv}(C\square)$, which violates Clause 1 from the validity of $\mathcal{L}$. Thus $\mathsf{inv}(B\square) > 1$, which entails that we can safely apply $\mathsf{F}{\supset_1}$ on $B\square$ as in the first case.

  Notice that whenever we apply the Fcomm rule, it is to apply the rule corresponding to the head connective of $B\square$ immediately afterwards: we never enter a loop by applying Fcomm twice in a row. Thus technically there are two reduction steps, but we treat them as one.

2. Since we do not deal with quantifiers, we can just take the same unifier σ.

3. Idem here, we take the same interleaving l.

$\square$

Then we can state the following *productivity theorem*, which is a direct consequence of the previous lemma and the fact that the rewriting rules terminate:

Theorem 3.5.2 (Productivity) *If $\mathcal{L}$ is a valid linkage, then there is a* *sequence of reductions with one of the following forms:*

$$\mathscr{L} \to^* D^+\boxed{\boxed{A} \circleddash \boxed{A}}$$

$$\mathscr{L} \to^* D\boxed{\boxed{t} = u @ A\boxed{t}} \qquad \mathscr{L} \to^* D\boxed{u = \boxed{t} @ A\boxed{t}}$$

This is the formal counterpart to the notion of productivity mentioned in Section 3.2. Intuitively, this theorem ensures non-trivial progress in the reasoning: we managed to connect some dots in the problem and actually solve a "subgoal". That is, either the conclusion is *strictly* weakened after a backward DnD, or the assumptions are *strictly* strengthened after a forward DnD, instead of having just an equivalent goal written in a different way[6]. This again contrasts with the release semantics of subformula linking which do not provide this guarantee of productivity, or with the logical reasoning tactics of proof assistants based on natural deduction rules.

6: This remark only applies to logical linkages however, since rewriting equalities can only produce equivalent statements. Some proof assistants provide facilities to rewrite arbitrary relations in subterms of arbitrary depth, such as `Coq` with its *generalized rewriting* mechanism [227]. This includes non-symmetric relations that can produce non-equivalent statements, and there is no reason in principle it could not be integrated in our paradigm, in the form of generalized substitution rules in place of $\mathsf{L}{=}_i$ and $\mathsf{F}{=}_i$.

3.6. Focusing

A last point to deal with is non-confluence and in particular choosing between first simplifying the head connective on the right or the left of $\circledast$ or $\circleddash$. For instance in $\boxed{A} \lor B \circleddash B \lor \boxed{A}$ one can apply either $\mathsf{L}\lor_1$ or $\mathsf{R}\lor_2$.

Interestingly, an answer is provided by *focusing*. It has been noticed by Andreoli [5] that, in bottom-up proof search, one should apply the invertible inference rules first since they preserve provability. In our framework, this translates into first applying the invertible rewriting rules (the ones marked by a *). In the case of the example above, this means performing $\mathsf{L}\lor_1$ first, which leads to the following behavior:

[5]: Andreoli (1992), 'Logic Programming with Focusing Proofs in Linear Logic'

$$\boxed{A} \lor B \circleddash B \lor \boxed{A} \to^* B \supset B \lor A.$$

This is indeed the "right" choice, since applying $\mathsf{R}\lor_2$ first would lead to a dead-end[7]:

$$\boxed{A} \lor B \circleddash B \lor \boxed{A} \to^* B \lor (B \supset A).$$

7: Interestingly in this case it creates a dead-end only in intuitionistic logic: in classical logic both results are provable.

The general scheme for choosing a rule to apply to a redex $C\boxed{A} @ D\boxed{B}$ is the following[8]:

8: A less deterministic version of this scheme is already present implicitly in the proof of Lemma 3.5.1.

1. If $C\square \equiv D\square \equiv \square$, we just apply the id rule (assuming $A \equiv B$ by Lemma 3.5.1).
2. If only one context is non-empty, say $C\square$, we look at its head connective as well as the side where its hole resides:
   - either $C\square \equiv C_0\square \circ E$ for some binary connective $\circ$, and we choose the rule $\mathsf{L}\circ_1$ (resp. $\mathsf{F}\circ_1$) if $@ = \circleddash$ (resp. $@ = \circledast$);
   - or $C\square \equiv E \circ C_0\square$ and we choose the rule $\mathsf{L}\circ_2$ (resp. $\mathsf{F}\circ_2$) if $@ = \circleddash$ (resp. $@ = \circledast$).

   In the case where it is $D\square$ which is non-empty, we apply the same logic but with the right rules $\mathsf{R}\circ_i$ instead of the left rules $\mathsf{L}\circ_i$.

3. If both contexts are non-empty, then the previous logic determines one rule for $C\square$ and one rule for $D\square$, giving rise to the ambiguity described in the above example.

There are three possibilities when analyzing invertibility of the two rules in the third case:

1. if both are invertible, then the order of application does not matter since we preserve provability in the end;
2. if only one is invertible, we apply it first following the focusing discipline;
3. if neither are invertible, we want to choose the order that maximizes the preservation of provability. It turns out that in almost all cases the two rules commute, that is the formulas obtained in the two orderings are equivalent. The only exceptions are the critical pairs $\mathsf{F}\vee_i/\mathsf{F}\supset_2$ for $i \in \{1, 2\}$, as was noted independently in [39]. In this case, one should rely on information given by the user to choose the right ordering, which can be done by exploiting the *orientation* of the associated DnD action, that is distinguishing between the source path and the destination path[9].

[39]: Chaudhuri (2021), 'Subformula Linking for Intuitionistic Logic with Application to Type Theory'

9: Note that in the current implementation of `Actema`, we instead rely on an arbitrary prioritizing fixed in the system, which can hinder in some cases the ability to prove a goal through DnD actions. In practice, one rarely encounters such cases in real examples.

Remark 3.6.1 Currently we do not have detailed proofs of permutability for all pairs of rules. The reason is mostly pragmatic: given the great number of rules, this would take a lot of time to perform a full case analysis. Actually our claim of permutability comes from [39] which uses a subformula linking system almost identical to ours.

3.7. Completeness

To enable a fully graphical approach to theorem proving that does not rely on a textual proof language, it is important to show that (a subset of) the set of actions exposed to the user is *complete* with respect to provability. That is, any formula A which is *true* in our logic — here intuitionistic FOL — can be proved by executing a sequence of graphical actions that reduces it to the empty goal. We noticed in Remark 2.3.1 that click actions are a sufficient basis for completeness. While we believe that a combination of both click and DnD actions is more comfortable to handle a variety of proof situations, it is still interesting to consider the question of completeness for DnD actions alone. It turns out that the answer is positive: the mechanism of *subformula linking* underlying DnD actions is powerful enough to capture provability in FOL. This has already been shown by Chaudhuri in [37] for linear logic, and [39] for intuitionistic logic. Here we give a completeness proof for a system based on a slight extension of our rewriting rules, following ideas from these works.

Remark 3.7.1 What we prove in this section is a *weak* form of completeness: we show that for any true formula, there always exists a derivation in our subformula linking calculus, but this derivation might not be constructible deterministically by the focusing procedure outlined in Section 3.6. There are two aspects that make the stronger version hard to prove in practice:

- To show that the choices performed by the focusing procedure always allow to find a proof when there exists one, it would be necessary to formulate and prove a *focusing theorem* based on the permutability of rules mentioned in Remark 3.6.1.
- Even then, some additional rules of our subformula linking system are not simulated in any way by the DnD procedure of Section 3.3. We will come back to this point soon.

To the rewriting rules of Figure 2.4 and Figure 2.5, we add *linkage formation* rules (Figure 3.2), which are a deep generalization of linkage creation between two formulas of a sequent. Rule B creates a *backward* linkage between $\supset$-linked formulas in any positive context $C^+\square$, and dually rule F creates a *forward* linkage between $\wedge$-linked formulas in any negative context $C^-\square$[10]. Note that linkage formation rules are not closed under arbitrary contexts: indeed the polarity restrictions are necessary to ensure soundness, as reviewed in Section 3.4. A backward linkage in a sequent $\Gamma, D\boxed{A} \Rightarrow E\boxed{B}$ would be encoded by the instance

$$C^+\boxed{D\boxed{A} \supset E\boxed{B}} \to C^+\boxed{D\boxed{A} \ogreaterthan E\boxed{B}}$$

of B where $C^+\square \equiv \bigwedge\Gamma \supset \square$, while a forward linkage in a sequent $\Gamma, D\boxed{A}, E\boxed{B} \Rightarrow F$ would be encoded by the instance

$$C^-\boxed{D\boxed{A} \wedge E\boxed{B}} \to C^-\boxed{D\boxed{A} \circledast E\boxed{B}}$$

of F where $C^-\square \equiv \bigwedge\Gamma \wedge \square \supset F$.

$$C^+\boxed{A \supset B} \to C^+\boxed{A \ogreaterthan B} \quad \mathsf{B}$$
$$C^-\boxed{A \wedge B} \to C^-\boxed{A \circledast B} \quad \mathsf{F}$$

Figure 3.2.: Linkage formation rules

10: This is reminiscent of the adjunction between products and exponentials in *cartesian closed categories*, which respectively interpret conjunction and implication in the Curry-Howard-Lambek correspondance for intuitionistic logic.

Another necessary ingredient is the addition of a deep version of the *structural rules* of sequent calculus. We already have the Fcomm rule to handle commutativity of the $\circledast$ operator, which acts as a kind of exchange rule. Then we add the equivalent of contraction and weakening with the rules conn and weak (Figure 3.3). These allow to erase and duplicate hypotheses at will, by identifying any subformula occurring in a negative context as a hypothesis. Thus once again we need to be careful about polarity, and cannot close these rules under arbitrary contexts.

$$C^-\boxed{A} \to C^-\boxed{A \wedge A} \quad \mathsf{conn}$$
$$C^-\boxed{A} \to C^-\boxed{\top} \quad \mathsf{weak}$$

Figure 3.3.: Resource rules

An alternative to the full contraction rule conn is to systematically duplicate negative formulas in the linkage formation rules, giving the rules Bconn and Fconn of Figure 3.4. This models more closely what we do in `Actema`, where hypotheses involved in a DnD action are always preserved in the new goal. This is important from a usability standpoint, because this ensures the user needs not fret with manual duplication of hypotheses in order to complete a proof. The downside is that the context always grows bigger, but this can be balanced by exposing the weakening rule in the interface. In `Actema` it is mapped to a "delete" button placed next to

$$C^+\boxed{A \supset B} \to C^+\boxed{A \supset (A \ogreaterthan B)} \quad \mathsf{Bconn}$$
$$C^-\boxed{A \wedge B} \to C^-\boxed{A \wedge (A \circledast B) \wedge B} \quad \mathsf{Fconn}$$

Figure 3.4.: Duplicating linkage formation rules

every hypothesis (the little gray trashbin icons in Figure 2.3). In our completeness proof we will use all the rules in {conn, weak, B, F, Bconn, Fconn}, in order to make derivations more concise.

Because linkages created by rules B and F are not necessarily valid, one needs to add the so-called *release* rules already mentioned in Section 3.1 (Figure 3.1). In fact these rules are crucial in order to simulate rules from sequent calculus, which will be the backbone of our completeness proof as in [37]. It is interesting to consider the question of completeness without release rules, especially since we do not use them in the semantics of DnD actions. We conjecture that it should hold but would require a completely different argument, maybe of a more semantic nature like the original proof of Gödel with Tarski models[11]. Another possibility might be to use a more canonical representation of proofs that is in-between syntax and semantics, like the combinatorial proofs of Hughes [127] which are known to be closely related to deep inference proofs.

11: Or Kripke models in an intuitionistic setting.

[127]: Hughes (2006), 'Proofs without syntax'

Lastly, a trivial but necessary addition is the rule refl of Figure 3.5 stating the reflexivity of =. It was not introduced before because it is already handled by click actions on red items in Actema (Section 2.3), but here we want a self-contained system that models as closely as possible the space of proofs that can be built through DnD actions only. In this context, one could imagine restricting the usage of the refl rule to the unit elimination phase (Section 3.3), where it would play the same role as the rules of Figure 2.5. Thus adding this rule does not correspond morally to modelling a click action, nor to a modification of the semantics of DnD actions as for release rules.

$$t = t \quad \rightarrow \quad \top \quad \mathsf{refl}$$

Figure 3.5.: Reflexivity rule for =

Then we simply rely on the completeness of sequent calculus by performing a proof by simulation. There are many variants of sequent calculi for intuitionistic first-order logic described in the literature. In our case the choice mostly does not matter: all we need is that it is *analytic*, i.e. satisifies the subformula property. Indeed the very idea of subformula linking is based on analyticity: one should be able to prove a statement by the sole act of linking sub-sentences already present in the statement.

We chose the calculus G3i from [183] as our basis, because all structural rules are admissible in it (but they would be straightforward to simulate apart from the cut rule). The first modification we do is that we model hypotheses in sequents as lists instead of multisets, to make the translation from sequents to formulas completely deterministic. Thus we need to add the exchange rule exch, which is simulated straightforwardly with the Fcomm rule as mentioned earlier. The second modification we do is adding introduction rules =R and =L for equality. The left introduction rule =L captures Leibniz's elimination scheme, and is in fact the rule Repl from [183] (modulo the fact that we use single-conclusion instead of multi-conclusion sequents). The right introduction rule =R is a 0-ary reflexivity rule, instead of the Ref rule from [183] (Figure 3.6). The reason is that we cannot simulate the latter directly without adding its equivalent rule ref to our calculus (Figure 3.6), which we do not want to do because it would break the subformula property. We conjecture that using =R instead of Ref does not break the cut admissibility theorem from [183].

$$\frac{\Gamma, t = t \Rightarrow C}{\Gamma \Rightarrow C}\ \mathsf{Ref}$$

$$\top \quad \rightarrow \quad t = t \quad \mathsf{ref}$$

Figure 3.6.: Non-analytic reflexivity rules

Theorem 3.7.1 (Completeness of subformula linking) *If* $\Gamma \Rightarrow A$ *is provable in* G3i + {exch, =R, =L}, *then* $\bigwedge \Gamma \supset A \to^* \top$.

Proof. By induction on the derivation of $\Gamma \Rightarrow A$. The base case simulates the rules ax, $\top$R, $\bot$L and =R. Other rules are simulated as usual by composing the derivations obtained from induction hypotheses, making a crucial use of the release rules Brel and Frel. The full mapping from sequent calculus rules to derivations in our subformula linking calculus is given in the following table. Note that we treat conjunctive formulas modulo associativity to avoid bureaucratic details.

$$
\dfrac{}{\Gamma, A \Rightarrow A}\,ax
\quad\mapsto\quad
\begin{array}{lll}
 & \Gamma \wedge A \supset A & \\
\to & \Gamma \wedge A \ogreaterthan A & \mathsf{B} \\
\to & A \ogreaterthan A & \mathsf{L}\wedge_2 \\
\to & \top & \mathsf{id}
\end{array}
$$

$$
\dfrac{\begin{array}{c}\vdots\,\pi_1\\ \Gamma, B, A, \Gamma' \Rightarrow C\end{array}}{\Gamma, A, B, \Gamma' \Rightarrow C}\,\mathsf{exch}
\quad\mapsto\quad
\begin{array}{lll}
 & \Gamma \wedge A \wedge B \wedge \Gamma' \supset C & \\
\to & \Gamma \wedge (A \circledast B) \wedge \Gamma' \supset C & \mathsf{F} \\
\to & \Gamma \wedge (B \circledast A) \wedge \Gamma' \supset C & \mathsf{Fcomm} \\
\to & \Gamma \wedge B \wedge A \wedge \Gamma' \supset C & \mathsf{Frel} \\
\to^* & \top & IH(\pi_1)
\end{array}
$$

$$
\dfrac{}{\Gamma \Rightarrow \top}\,\top R
\quad\mapsto\quad
\begin{array}{lll}
 & \Gamma \supset \top & \\
\to & \top & \mathsf{absr}
\end{array}
$$

$$
\dfrac{\begin{array}{cc}\vdots\,\pi_1 & \vdots\,\pi_2\\ \Gamma \Rightarrow A & \Gamma \Rightarrow B\end{array}}{\Gamma \Rightarrow A \wedge B}\,\wedge R
\quad\mapsto\quad
\begin{array}{lll}
 & \Gamma \supset A \wedge B & \\
\to & \Gamma \supset (\Gamma \ogreaterthan A \wedge B) & \mathsf{Bconn} \\
\to & \Gamma \supset (\Gamma \ogreaterthan A) \wedge B & \mathsf{R}\wedge_1 \\
\to & \Gamma \supset (\Gamma \supset A) \wedge B & \mathsf{Brel} \\
\to^* & \Gamma \supset \top \wedge B & IH(\pi_1) \\
\to & \Gamma \supset B & \mathsf{neul} \\
\to^* & \top & IH(\pi_2)
\end{array}
$$

$$
\dfrac{\begin{array}{c}\vdots\,\pi_1\\ \Gamma \Rightarrow A_i\end{array}}{\Gamma \Rightarrow A_0 \vee A_1}\,\vee R_i
\quad\mapsto\quad
\begin{array}{lll}
 & \Gamma \supset A_0 \vee A_1 & \\
\to & \Gamma \ogreaterthan A_0 \vee A_1 & \mathsf{B} \\
\to & (\Gamma \ogreaterthan A_i) \vee A_{1-i} & \mathsf{R}\vee_i \\
\to & (\Gamma \supset A_i) \vee A_{1-i} & \mathsf{Brel} \\
\to^* & \top \vee A_{1-i} & IH(\pi_1) \\
\to & \top & \mathsf{absl}
\end{array}
$$

$$
\dfrac{\begin{array}{c}\vdots\,\pi_1\\ \Gamma, A \Rightarrow B\end{array}}{\Gamma \Rightarrow A \supset B}\,\supset R
\quad\mapsto\quad
\begin{array}{lll}
 & \Gamma \supset A \supset B & \\
\to & \Gamma \ogreaterthan A \supset B & \mathsf{B} \\
\to & (\Gamma \circledast A) \supset B & \mathsf{R}\supset_1 \\
\to & \Gamma \wedge A \supset B & \mathsf{Frel} \\
\to^* & \top & IH(\pi_1)
\end{array}
$$

$$\dfrac{\begin{array}{c}\vdots\ \pi_1\\ \Gamma \Rightarrow A\end{array}}{\Gamma \Rightarrow \forall x.A}\,\forall R \quad x \notin \mathsf{fv}(\Gamma) \qquad \mapsto \qquad \begin{array}{lll} & \Gamma \supset \forall x.A & \\ \to & \Gamma \ogreaterthan \forall x.A & \mathsf{B} \\ \to & \forall x.(\Gamma \ogreaterthan A) & \mathsf{R\forall s} \\ \to & \forall x.\Gamma \supset A & \mathsf{Brel} \\ \to^* & \forall x.\top & IH(\pi_1) \\ \to & \top & \mathsf{absq} \end{array}$$

$$\dfrac{\begin{array}{c}\vdots\ \pi_1\\ \Gamma \Rightarrow A\{t/x\}\end{array}}{\Gamma \Rightarrow \exists x.A}\,\exists R \qquad \mapsto \qquad \begin{array}{lll} & \Gamma \supset \exists x.A & \\ \to & \Gamma \ogreaterthan \exists x.A & \mathsf{B} \\ \to & \Gamma \ogreaterthan A\{t/x\} & \mathsf{R\exists i} \\ \to & \Gamma \supset A\{t/x\} & \mathsf{Brel} \\ \to^* & \top & IH(\pi_1) \end{array}$$

$$\dfrac{}{\Gamma \Rightarrow t = t}\,{=}\mathsf{R} \qquad \mapsto \qquad \begin{array}{lll} & \Gamma \supset t = t & \\ \to & \Gamma \supset \top & \mathsf{refl} \\ \to & \top & \mathsf{absr} \end{array}$$

$$\dfrac{\begin{array}{c}\vdots\ \pi_1\\ \Gamma \Rightarrow C\end{array}}{\Gamma, \top \Rightarrow C}\,\top L \qquad \mapsto \qquad \begin{array}{lll} & \Gamma \wedge \top \supset C & \\ \to & \Gamma \supset C & \mathsf{neur} \\ \to^* & \top & IH(\pi_1) \end{array}$$

$$\dfrac{}{\Gamma, \bot \Rightarrow C}\,\bot L \qquad \mapsto \qquad \begin{array}{lll} & \Gamma \wedge \bot \supset C & \\ \to & \Gamma \wedge \bot \ogreaterthan C & \mathsf{B} \\ \to & \bot \ogreaterthan C & \mathsf{L\wedge_2} \\ \to & \bot \supset C & \mathsf{Brel} \\ \to & \top & \mathsf{efq} \end{array}$$

$$\dfrac{\begin{array}{c}\vdots\ \pi_1\\ \Gamma, A, B \Rightarrow C\end{array}}{\Gamma, A \wedge B \Rightarrow C}\,\wedge L \qquad \mapsto \qquad \begin{array}{lll} & \Gamma \wedge A \wedge B \supset C & \\ \to^* & \top & IH(\pi_1) \end{array}$$

$$\dfrac{\begin{array}{c}\vdots\,\pi_1\\ \Gamma, A \Rightarrow C\end{array} \quad \begin{array}{c}\vdots\,\pi_2\\ \Gamma, B \Rightarrow C\end{array}}{\Gamma, A \vee B \Rightarrow C}\,\vee L \quad \mapsto \quad \begin{array}{lll}
 & \Gamma \wedge (A \vee B) \supset C & \\
\to & \Gamma \wedge (A \vee B) \supset (\Gamma \wedge (A \vee B) \ogreaterthan C) & \mathsf{Bconn} \\
\to & \Gamma \wedge \top \supset (\Gamma \wedge (A \vee B) \ogreaterthan C) & \mathsf{weak} \\
\to & \Gamma \supset (\Gamma \wedge (A \vee B) \ogreaterthan C) & \mathsf{neur} \\
\to & \Gamma \supset (A \vee B \ogreaterthan C) & \mathsf{L}\wedge_2 \\
\to & \Gamma \supset (A \ogreaterthan C) \wedge (B \supset C) & \mathsf{L}\vee_1 \\
\to & \Gamma \supset (A \supset C) \wedge (B \supset C) & \mathsf{Brel} \\
\to & \Gamma \supset (\Gamma \ogreaterthan (A \supset C) \wedge (B \supset C)) & \mathsf{Bconn} \\
\to & \Gamma \supset (\Gamma \ogreaterthan A \supset C) \wedge (B \supset C) & \mathsf{R}\wedge_1 \\
\to & \Gamma \supset ((\Gamma \circledast A) \supset C) \wedge (B \supset C) & \mathsf{R}\supset_1 \\
\to & \Gamma \supset (\Gamma \wedge A \supset C) \wedge (B \supset C) & \mathsf{Frel} \\
\to^* & \Gamma \supset \top \wedge (B \supset C) & IH(\pi_1) \\
\to & \Gamma \supset B \supset C & \mathsf{neul} \\
\to & \Gamma \ogreaterthan B \supset C & \mathsf{B} \\
\to & (\Gamma \circledast B) \supset C & \mathsf{R}\supset_1 \\
\to & \Gamma \wedge B \supset C & \mathsf{Frel} \\
\to^* & \top & IH(\pi_2)
\end{array}$$

$$\dfrac{\begin{array}{c}\vdots\,\pi_1\\ \Gamma, A \supset B \Rightarrow A\end{array} \quad \begin{array}{c}\vdots\,\pi_2\\ \Gamma, B \Rightarrow C\end{array}}{\Gamma, A \supset B \Rightarrow C}\,\supset L \quad \mapsto \quad \begin{array}{lll}
 & \Gamma \wedge (A \supset B) \supset C & \\
\to & \Gamma \wedge \Gamma \wedge (A \supset B) \supset C & \mathsf{conn} \\
\to & \Gamma \wedge (\Gamma \circledast A \supset B) \supset C & \mathsf{F} \\
\to & \Gamma \wedge ((\Gamma \ogreaterthan A) \supset B) \supset C & \mathsf{F}\supset_1 \\
\to & \Gamma \wedge ((\Gamma \supset A) \supset B) \supset C & \mathsf{Brel} \\
\to^* & \Gamma \wedge (\top \supset B) \supset C & IH(\pi_1) \\
\to & \Gamma \wedge B \supset C & \mathsf{neul} \\
\to^* & \top & IH(\pi_2)
\end{array}$$

$$\dfrac{\begin{array}{c}\vdots\,\pi_1\\ \Gamma, A\{t/x\} \Rightarrow C\end{array}}{\Gamma, \forall x.A \Rightarrow C}\,\forall L \quad \mapsto \quad \begin{array}{lll}
 & \Gamma \wedge (\forall x.A) \supset C & \\
\to & \Gamma \wedge (\forall x.A) \supset (\Gamma \wedge (\forall x.A) \ogreaterthan C) & \mathsf{Bconn} \\
\to & \Gamma \wedge (\forall x.A) \supset (\forall x.A \ogreaterthan C) & \mathsf{L}\wedge_2 \\
\to & \Gamma \wedge \top \supset (\forall x.A \ogreaterthan C) & \mathsf{weak} \\
\to & \Gamma \supset (\forall x.A \ogreaterthan C) & \mathsf{neur} \\
\to & \Gamma \supset (A\{t/x\} \ogreaterthan C) & \mathsf{L}\forall\mathsf{i} \\
\to & \Gamma \supset (A\{t/x\} \supset C) & \mathsf{Brel} \\
\to & \Gamma \ogreaterthan (A\{t/x\} \supset C) & \mathsf{B} \\
\to & (\Gamma \circledast A\{t/x\}) \supset C & \mathsf{R}\supset_1 \\
\to & \Gamma \wedge A\{t/x\} \supset C & \mathsf{Frel} \\
\to^* & \top & IH(\pi_1)
\end{array}$$

$$\dfrac{x \notin \mathsf{fv}(\Gamma) \cup \mathsf{fv}(C) \quad \begin{array}{c}\vdots\,\pi_1\\ \Gamma, A \Rightarrow C\end{array}}{\Gamma, \exists x.A \Rightarrow C}\,\exists L \quad \mapsto \quad \begin{array}{lll}
 & \Gamma \wedge (\exists x.A) \supset C & \\
\to & \Gamma \wedge (\exists x.A) \supset (\Gamma \wedge (\exists x.A) \ogreaterthan C) & \mathsf{Bconn} \\
\to & \Gamma \wedge (\exists x.A) \supset (\exists x.A \ogreaterthan C) & \mathsf{L}\wedge_2 \\
\to & \Gamma \wedge \top \supset (\exists x.A \ogreaterthan C) & \mathsf{weak} \\
\to & \Gamma \supset (\exists x.A \ogreaterthan C) & \mathsf{neur} \\
\to & \Gamma \supset \forall x.(A \ogreaterthan C) & \mathsf{L}\exists\mathsf{s} \\
\to & \Gamma \supset \forall x.A \supset C & \mathsf{Brel} \\
\to & \Gamma \ogreaterthan \forall x.A \supset C & \mathsf{B} \\
\to & \forall x.(\Gamma \ogreaterthan A \supset C) & \mathsf{R}\forall\mathsf{s} \\
\to & \forall x.(\Gamma \circledast A) \supset C & \mathsf{R}\supset_1 \\
\to & \forall x.\Gamma \wedge A \supset C & \mathsf{Frel} \\
\to^* & \forall x.\top & IH(\pi_1) \\
\to & \top & \mathsf{absq}
\end{array}$$

$$\dfrac{\dfrac{\vdots\ \pi_1}{\Gamma, t = u, A\{u/x\}, A\{t/x\} \Rightarrow C}}{\Gamma, t = u, A\{t/x\} \Rightarrow C}\ {=}\mathsf{L} \quad \mapsto \quad \begin{array}{lll} & \Gamma \wedge t = u \wedge A\{t/x\} \supset C & \\ \rightarrow & \Gamma \wedge t = u \wedge (t = u \circledast A\{t/x\}) \wedge A\{t/x\} \supset C & \mathsf{Fconn} \\ \rightarrow & \Gamma \wedge t = u \wedge A\{u/x\} \wedge A\{t/x\} \supset C & \mathsf{F}{=}_1 \\ \rightarrow^* & \top & IH(\pi_1) \end{array}$$

□

3.8. Related works

Subformula linking under quantifiers Very recently, Mulder and Krebbers [179] proposed an improvement over both our method of subformula linking implemented in `Actema`, and the method of Chaudhuri implemented in the `Profint` prototype [39]. Like us, they perform *a priori* unification on the linked subformulas, both to determine appropriate substitutions for instantiating quantifiers, and rule out invalid linkages. But their method improves upon ours by being able to link subformulas with non-trivial quantifier instantiations, such as the following linkage that currently fails in `Actema`[12]:

$$(\forall x. P(x) \supset \exists y. \boxed{Q(x, y)}) \circleddash \exists y. \exists z. \boxed{Q(f(z), y)}$$

Because of the intended application of their method to *automated* theorem proving in the `Iris` framework for program verification in `Coq` [136], it is for now limited to *backward* linkages. They provide a detailed formalization in `Coq` that relates their method with those of `Actema` and `Profint`, based on *linking judgments* of the form $H \wedge [O] \Vdash G$ that have a derivation precisely when $H \circleddash G \rightarrow^* O$[13].

Canonical proofs The idea of reducing a proof to a collection of links between its dual formulas is not new, and can be traced back to the *matings* of Andrews [6] in the context of automated deduction. Matings are *sets* of links covering all *atomic* occurrences, and proofs are matings satisfying certain conditions. Our work differs in that we are interested in *interactive* deduction, and thus consider links as a mechanism of inference rather than a syntactic criterion to discriminate proofs. Then a proof is better understood as a *list* of links, and the atomicity constraint is relaxed to gain expressivity, since the creation of links is offloaded to the user instead of the search procedure.

Another line of work, starting with the *proof nets* of Girard [92], is concerned with the more fundamental problem of *proof identity*, which requires a canonical notion of proof object [233]. In the case of unit-free multiplicative linear logic, the absence of any form of duplication/sharing/removal mechanism allows to completely characterize a proof net by the set of its *axiom* links[14], because of the absence of duplication. This is because adding additives or exponentials, which can encode intuitionistic and classical logic, requires additional structure to represent uses of weakening and contraction. The *combinatorial proofs* of Hughes [117, 127] are examples of polynomially-checkable proof objects exhibiting such

[179]: Mulder et al. (2024), 'Unification for Subformula Linking under Quantifiers'

[39]: Chaudhuri (2021), 'Subformula Linking for Intuitionistic Logic with Application to Type Theory'

12: It would be interesting to understand precisely why it fails, and if this invalidates the productivity theorem.

[136]: Jung et al. (2018), 'Iris from the ground up'

13: Contrary to our usage in this chapter, they use the terminology "linkage" to denote derivations of these linking judgments, rather than paths to selected subformulas.

[6]: Andrews (1976), 'Refutations by Matings'

[92]: Girard (1987), 'Linear logic'

[233]: Straßburger (2019), 'The problem of proof identity, and why computer scientists should care about Hilbert's 24th problem'

14: The difference with matings is that correctness of a proof structure can be checked in *polynomial* instead of exponential time.

[117]: Heijltjes et al. (2019), 'Intuitionistic proofs without syntax'

[127]: Hughes (2006), 'Proofs without syntax'

structure, and have recently been extended to handle first-order classical quantifiers [128] (intuitionistic quantifiers are still an open problem). This compartmentalization of axiom links and structural rules resembles the distinction between interaction phases and manual applications of conn and weak (Figure 3.3), which is itself inspired by the *decomposition theorem* of the calculus of structures [242, Theorem 4.1.3].

[128]: Hughes (2019), *First-order proofs without syntax*

[242]: Tubella et al. (2019), 'Introduction to Deep Inference'

There is also an analogy between the correctness criterions of proof nets, and the validity criterion of linkages:

- they both identify a subset of valid objects among a larger set of structures characterized by links on subformulas;
- they both allow many different *sequential* readings, that is sequent calculus proofs for proof nets, and CoS proofs for linkages[15].

15: Note that invalid linkages still give rise to CoS-style *derivations*, but not *proofs* since they do not end with $\top$. The incorrect proof structures of Girard are in a sense more parallel as they cannot always be mapped to correct sequent calculus derivations.

Hence, our approach to subformula linking seems to exhibit some properties of canonical proofs, but at the level of *partial proofs*: valid linkages make for *compact-parallel-spatial* representations of inferences, whose operational meaning is given by their *detailed-sequential-temporal* CoS derivations.

Tangible functional programming We noticed an interesting connection with the work of Conal Elliott on *tangible functional programming* [75]. His concept of *deep application* of λ-terms seems related to the notion of subformula linking, when viewing function and product types as implications and conjunctions through the formulae-as-types interpretation. He also devised a system of basic combinators which are composed sequentially to compute the result of a DnD, though it follows a more complex dynamic than our rewriting rules. Even if the mapping between proofs and programs is not exact in this case, it suggests a possible interesting field of application for the Curry-Howard correspondance, in the realm of graphical proving/programming environments.

[75]: Elliott (2007), 'Tangible Functional Programming'

Proof-by-Action in Practice | 4.

4.1 Forward reasoning . . . 61
4.1.1 A gestural proof 61
4.1.2 Comparison with a textual proof 62
4.2 Sets and functions . . . 65
4.2.1 A simple exercise 65
4.2.2 Nominal definitions . . . 65
4.2.3 Axiomatic definitions . . 66
4.2.4 Computational definitions 67
4.2.5 Finishing the proof 68
4.3 Peano arithmetic 71

> Practical life is not necessarily directed toward other people, as some think; and it is not the case that practical thoughts are only those which result from action for the sake of what ensues. On the contrary, much more practical are those mental activities and reflections which have their goal in themselves and take place for their own sake.
>
> **Aristotle**, *Politics, VII, 3, 8, 1325b16-20*

In the previous chapters, we explained the core principles of our so-called Proof-by-Action paradigm and especially of its drag-and-drop actions, first through basic and abstract examples in Chapter 2, and then from a proof-theoretical perspective in Chapter 3. The goal of this chapter is to provide a better sense of what proofs by action/DnD look like in practice, and how they compare to more traditional approaches to interactive theorem proving. To that effect, we perform a case study of a few select examples, unrolling and commenting in details one or many of their proofs. Although still basic, they are fully fledged, concrete logical riddles or mathematical problems that one might give as exercise to an undergraduate student learning formal proofs. Note that our analysis will stay quite informal and opinionated: a more systematic approach such as a user study would allow for a better evaluation of our paradigm, but at the time of writing of this thesis the `Actema` prototype is not mature enough to conduct a project of this scale.

The chapter is organized as follows: Section 4.1 studies a proof of a small logical riddle in `Actema`, highlighting some benefits of our approach compared to textual systems. Section 4.2 explores how basic properties about functions between sets can be proved graphically, introducing the use of *definitions* in addition to logical reasoning steps. In Section 4.3 we prove equations in Peano arithmetic, showing how one can incorporate additional actions into the paradigm to deal with more specialized forms of reasoning: *induction* and *automatic computation*.

Note

For each example, we provide a `Coq` proof script that tries to follow the structure of the graphical proof, for the sake of comparison with a textual interface. But this would obviously compare differently with other textual interfaces, like the `Isar` proof language which is more declarative, and thus farther from the imperative aspects of PbA.

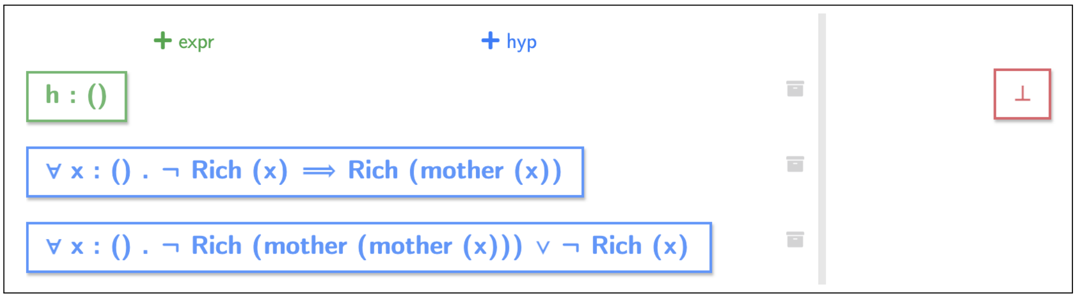


Figure 4.1.: The beginning of an example due to `Edukera`

4.1. Forward reasoning

4.1.1. A gestural proof

Our first example is a small logical riddle, which we borrow from a textual educational system, `Edukera` [217]. We invite readers to try to perform the proof themselves in the online version of `Actema`[1]. One considers a population of people, with at least one individual h, together with a single function mother and one predicate Rich. The aim is to show that the two following assumptions are incompatible:

[217]: Rognier et al. (2016), ‘Présentation de la plateforme edukera’

1: `https://www.actema.xyz/courses/edukera`

$$\forall x. \neg \mathrm{Rich}(x) \vee \neg \mathrm{Rich}(\mathrm{mother}(\mathrm{mother}(x))) \tag{4.1}$$

$$\forall x. \neg \mathrm{Rich}(x) \supset \mathrm{Rich}(\mathrm{mother}(x)) \tag{4.2}$$

The original goal thus corresponds to the illustration of Figure 4.1.

It is quite natural to approach this problem in a forward manner, by starting from the hypotheses to establish new facts. And a first point illustrated by this example is that DnD actions allow to do this in a smooth and precise manner. A possible first step is to bring h to hypothesis (4.1), to obtain a new fact:

$$\neg \mathrm{Rich}(h) \vee \neg \mathrm{Rich}(\mathrm{mother}(\mathrm{mother}(h))) \tag{4.3}$$

Double-clicking on this new fact yields two cases:

$$\neg \mathrm{Rich}(h) \tag{4.4}$$

$$\neg \mathrm{Rich}(\mathrm{mother}(\mathrm{mother}(h))) \tag{4.5}$$

Let us detail how one solves the second one. By bringing hypothesis (4.5) on the premise of hypothesis (4.2) one obtains

$$\mathrm{Rich}(\mathrm{mother}(\mathrm{mother}(\mathrm{mother}(h)))) \tag{4.6}$$

The next step is a good example where DnD actions are useful. By bringing this new fact to the right-hand part of hypothesis (4.3) one immediately obtains a new fact

$$\neg \mathrm{Rich}(\mathrm{mother}(h)) \tag{4.7}$$

In textual proof languages, this last step requires a somewhat intricate tactic line and/or writing down at least the statement of the new fact.

One can then finish the case by combining hypotheses (4.7) and (4.2), which yields

$$\mathsf{Rich}(\mathsf{mother}(\mathsf{mother}(h)))$$

contradicting hypothesis (4.5). These two last steps each correspond to a simple DnD. The other case, $\neg\mathsf{Rich}(h)$, is quite similar.

Note that once a user has understood the proof, the riddle is routinely solved in less than a minute in `Actema`, which seems out of reach for about any user in a tactic-based prover. At least as important is the fact that the proof can be performed without typing any text, especially no intermediate statement.

4.1.2. Comparison with a textual proof

To conclude this example, we propose in Figure 4.2 a complete proof of the riddle formalized in the `Coq` proof assistant, which follows very closely the structure of the graphical proof just outlined. To make the correspondence more visible and ease the comparison, we interspersed the proof script with comments of the form `(** [actions] *)`, where `[actions]` is a sentence describing a sequence of actions in `Actema` that produces the same goal transformation as the tactics preceding the comment. There are a few interesting things to note:

Hypotheses management We chose to name manually all the hypotheses introduced in the course of the proof. This is generally considered good practice in the `Coq` community, because it makes proof scripts easier to maintain. In our case it also has the advantage of expliciting which hypotheses are used exactly in the reasoning, something that an `Actema` user does with her pointing device when designating the blue items involved in an action.

It appears clearly in Figure 4.2 that in a moderately long proof like this based mostly on forward reasoning, one needs to keep track of *a lot* of names, which can be overwhelming for many users. This is especially true for beginners discovering `Coq`, because the syntax for assigning names, based on patterns like `[H | H]` that reproduce the subgoal structure, can induce a steep learning curve. Of course this problem is mitigated trivially in `Actema`, since names are not needed.

Tactics vs. actions There is no exact correspondence between the tactics of `Coq` and the actions of `Actema`: some tactics are simulated by multiple actions, and often a complex sequence of tactics can be simulated by a single action[2].

2: This was already noticed in [18], where clicking on a subformula can simulate a sequence of introduction rules of arbitrary length.

For instance, line 23 does at the same time a specialization of the hypothesis $H_2 : \forall x.\neg\mathsf{Rich}(\mathsf{mother}(\mathsf{mother}(x))) \vee \neg\mathsf{Rich}(x)$ to the individual h with the application `(H2 h)`, and a case analysis with the `destruct` tactic. In `Actema` this is performed in two steps, first by drag-and-dropping h on H_2, and then by clicking on the resulting hypothesis[3].

3: This could also be achieved in two steps in `Coq`, by using the `specialize` tactic instead of the inlined application.

In the other direction, a pattern of reasoning that occurs multiple times in the proof is the combination of H_2 with another hypothesis which contradicts one of the two cases, in order to deduce the truth of the other case. While it is captured straightforwardly in `Actema` with a single DnD between the contradictory statements, it requires in `Coq` a decomposition into many administrative steps:

1. first a case analysis with `destruct`, where the expression instantiating H_2 (e.g. $\mathsf{mother}(\mathsf{mother}(h))$) needs to be written down explicitly, instead of being inferred automatically from unification;
2. optionally focusing on the subgoal corresponding to the contradictory case if it is the right disjunct (line 56), which requires to know a somewhat idiosyncratic and infrequently used syntax of the tactic language;
3. and finally expliciting the contradiction with `apply` and `exact`.

Context-sensitivity More generally, the actions of `Actema` are more *versatile* and *context-aware* than the tactics of `Coq`. For instance, click actions have a different effect depending on the main connective of the formula being clicked, but provide a unique interface for applying rules of natural deduction/sequent calculus. On the contrary, there is almost one tactic for dealing with each logical connective in `Coq`, e.g. `intros` for $\supset$ and $\forall$, `split` for $\wedge$, `left` and `right` for $\vee$, `exists` for $\exists$, etc. The same remark applies to DnD actions, whose functionalities are provided in `Coq` by many different tactics: `apply _`, `apply _ in _`, `pose proof`, `specialize`, etc.

From this detailed comparison, it appears that the interface offered by the PbA paradigm might be more suited to forward reasoning than the tactic language of `Coq`, at least in some respects. It makes the flow of argumentation more straightforward to express with DnD actions, and avoids the overheads of name management and syntax memorization. This altogether shall prove to be particularly helpful to beginners and learners of the proof assistant.

```

(* Declaration of constants used in the statement of the riddle *)

Context (i : Type).
Context (Rich : i -> Prop).
Context (mother : i -> i).
Context (h : i).

(* Statement of the riddle *)

Theorem rich_mothers :
  (forall x, ~Rich(x) -> Rich(mother(x))) ->
  (forall x,  ~Rich(mother(mother(x))) \/ ~Rich(x)) ->
  False.

(* Proof of the riddle *)

Proof.
  intros H1 H2.
  (** 2 clicks on the conclusion *)

  destruct (H2 h) as [H | H].
  (** DnD of [h] onto [H2], then click on the resulting hypothesis *)

  * pose proof (H1 _ H) as H'.
    (* If one naively uses [apply _ in], then one loses [H] although
       it is needed later! Hence the use of [pose proof]. *)
    (** DnD of [H1] onto [H] *)

    destruct (H2 (mother h)) as [H2' | H2'].
    apply H2'. exact H'.
    (** DnD of [H'] onto [H2]. Could also be performed stepwise:
        - Selection of [mother(h)] in [H']
        - DnD of [H'] onto [H2]
        - Click on the resulting hypothesis
        - DnD of [H2'] onto [H'] *)

    apply H1 in H2'.
    (** DnD of [H1] onto [H2'] *)

    apply H. exact H2'.
    (** DnD of [H] onto [H2'] *)

  * pose proof (H1 _ H) as H'.
    (** DnD of [H1] onto [H] *)

    destruct (H2 (mother h)) as [H2' | H2'].
    2: { apply H2'. exact H'. }
    (** DnD of [H'] onto [H2] *)

    pose proof (H1 _ H2') as H2''.
    (** DnD of [H1] onto [H2'] *)

    destruct (H2 (mother (mother h))) as [H2''' | H2'''].
    apply H2'''. exact H2''.
    (** DnD of [H2''] onto [H2] *)

    apply H1 in H2'''.
    (** DnD of [H1] onto [H2'''] *)

    apply H2'. exact H2'''.
    (** DnD of [H2'] onto [H2'''] *)
Qed.

```

Figure 4.2.: Coq proof script formalizing Edukera's riddle

4.2. Sets and functions

4.2.1. A simple exercise

Our second example comes from a preprint of Bartzia et al. [15], where it is chosen specifically for a case study aiming to compare the features of different proof assistants' interfaces in an educational setting. It is "a typical exercise about sets, relations and functions, as commonly found in introductory courses about reasoning and proof." (p. 6):

[15]: Bartzia et al. (2023), 'Proof assistants for undergraduate mathematics education: elements of an a priori analysis'

Exercise 4.2.1 Given three sets A, B and C such that $C \subseteq A$ and a function $f : A \to B$, show that:

1. $C \subseteq f^{-1}(f(C))$.
2. If f is injective then $f^{-1}(f(C)) \subseteq C$.

where $f(D)$ and $f^{-1}(E)$ denote respectively the direct and inverse image (or preimage) of sets $D \subseteq A$ and $E \subseteq B$ by f.

Compared to our previous example, this exercise has the particularity of involving multiple *definitions*, here about sets and functions between them. There are many possible ways to handle definitions in a formal proof system. A common one, which might be termed *nominal*, is to decompose the definition into a new function or predicate symbol, the definition's *head*, and a universally parametrized equality or equivalence between the head and the *body* of the definition. For instance, the concept of injectivity can be encoded as a unary predicate $\mathsf{injective}(-)$ on functions, satisfying the following equivalence:

$$\forall A.\, \forall B.\, \forall f\colon A \to B.\ \ \mathsf{injective}(f) \ \Leftrightarrow\ \forall x \in A.\, \forall y \in A.\ f(x) = f(y) \supset x = y$$

Notice that $\mathsf{injective}(-)$ is a *higher-order* predicate, since it takes any function as argument. Depending on the underlying logical framework, this might have an impact on the exact way the definition is encoded. Here we do not want to bother with such implementation details, and simply assume that higher-order definitions are possible, even though `Actema` is currently limited to first-order logic. In practice this does not affect the semantics of graphical actions, and we can imagine having a first-order set theory such as `ZF` to make everything work behind the scenes[4].

4: See also Section 6.6 for a discussion on a higher-order extension of `Actema`.

4.2.2. Nominal definitions

Let us now describe how to prove the second question of the exercise in the PbA paradigm. The first thing we want to do is to unfold the definition of set inclusion in the conclusion $f^{-1}(f(C)) \subseteq C$, so that we can see how to prove logically such an inclusion. One might imagine multiple kinds of graphical actions for doing this. In `Actema` we implemented a general *subterm selection* mechanism, where the user can successively point at different subterms appearing in the goal and then choose from a list of so-called *contextual actions* that take the selection as argument. In our case we can select the whole conclusion, and then choose to apply the Unfold action:

$$\boxed{f^{-1}(f(C)) \subseteq C} \qquad \text{(Unfold)}$$

The system will be able to tell that we selected an instance of the two-place inclusion predicate $- \subseteq -$, and thus will replace the head of this definition by its body, instantiating parameters accordingly. This gives us a new conclusion

$$\forall x. x \in f^{-1}(f(C)) \supset x \in C$$

on which we can click twice to introduce a new variable x in the context, together with the new hypothesis

$$x \in f^{-1}(f(C)) \tag{4.8}$$

Now there is no available action on the conclusion $x \in C$, because set membership is a *primitive* predicate in set theory. But we can still unfold some definitions in the context, which might suggest further possible interactions. First we can unfold the definition of preimage used in hypothesis (4.8) by selecting the precise corresponding subterm:

$$x \in \boxed{f^{-1}(f(C))} \qquad \text{(Unfold)}$$

which, assuming a definition based on set comprehension, gives:

$$x \in \{a \in A \mid f(a) \in f(C)\} \tag{4.9}$$

At this stage, we would like to make the set comprehension in hypothesis (4.9) disappear, by simply deducing $f(x) \in f(C)$ from it. But depending on the underlying logical framework, the way to perform this deduction step in PbA will vary.

4.2.3. Axiomatic definitions

In a set theory such as ZF, the meaning of $\in$ comes from the various axioms involving it. One might call this a *behavioral* (or *axiomatic*) definition, since the meaning of the symbol emerges from the way it can be used in proofs through axioms. This contrasts with the previous nominal definitions, that have a much simpler semantics captured by Unfold actions[5].

In particular, we can simplify the set comprehension in hypothesis (4.9) by invoking explicitly the *axiom of comprehension*, which states the following:

$$\forall \phi. \forall D. \forall y. \; y \in \{d \in D \mid \phi\} \;\; \Leftrightarrow \;\; (y \in D \wedge \phi\{y/d\})$$

Note that this is again a higher-order statement, but this time because it quantifies over every formula ϕ[6]. Thus we assume that the underlying proof engine can handle such higher-order statements, and in particular that the axiom of comprehension is available in its database of lemmas.

5: Note that the syntax of first-order logic is unaware of this distinction, since in both cases the defined concepts are encoded as *atomic* predicates. This is usually not the case of logical frameworks found in proof assistants: for instance, the duality between judgmental (nominal) and *propositional* (behavioral) equality is at the heart of Martin-Löf type theory, and it is used extensively in `Coq` to perform automation, both in the computation of expressions and the matching of statements modulo definitions.

6: Traditionnally, logicians preferred to speak of *axiom schemas*, that is countable sets of axioms, rather than higher-order axioms, in order to stay purely in a first-order setting. But this does not make much sense from an implementation point of view, as a proof engine will only be able to manipulate a finite amount of axioms.

In Actema, one can freely search in this database by typing text in a search bar, typically in this case the keyword “comprehension”. Then the system will show a list of lemmas whose names match the keywords, and the user can click on the lemma she is interested in, in order to add it as a blue item in the current context.

An alternative and more precise way of retrieving a lemma is to search by *content* of the statement instead of searching by name. State-of-the-art proof assistants usually provide facilities for this: for instance Coq has a `Search` command which can take *patterns*, i.e. terms with so-called *holes* or *metavariables*, in order to filter out results that do not match the given pattern.

In Actema, we implemented a novel feature which replaces patterns by a selection of subterms in the current goal, similarly to what is given as argument to contextual actions like Unfold. Then the system will only look for lemmas which can be used in a DnD action involving precisely the current selection of subterms.

Coming back to our proof, selecting the full statement of hypothesis (4.9) and searching:

$$\boxed{x \in \{a \in A \mid f(a) \in f(C)\}} \qquad \text{(Search)}$$

should return the comprehension axiom among other lemmas, because this axiom and hypothesis can interact through the following forward DnD:

$$\boxed{x \in \{a \in A \mid f(a) \in f(C)\}} \circledast \forall \phi. \forall D. \forall y. \boxed{y \in \{d \in D \mid \phi\}} \Leftrightarrow (y \in D \wedge \phi\{y/d\})$$

with the unifying substitution $\{D := A, d := a, y := x, \phi := f(a) \in f(C)\}$. Performing this DnD will finally result in a new hypothesis corresponding to the intended “unfolding” of set comprehension:

$$x \in A \wedge f(x) \in f(C)$$

4.2.4. Computational definitions

In type theory, every mathematical object or statement is ultimately encoded as a function, in the sense of the λ-calculus. It is Alonzo Church who first got the idea of representing a set by its *characteristic function* in his *higher-order logic* (HOL), in the form of a λ-term of type $\iota \to o$ where ι is the type of individuals and o the type of propositions [48]. With this encoding, the only way to construct a set is by comprehension, and set membership corresponds to function application; that is, $\{x \in A \mid \phi\}$ is identified with the characteristic function $\lambda x \colon A.\ \phi$, and $x \in t$ is the application $t\,x$[7]. Then if we consider λ-terms modulo β-equivalence, “unfolding” the definition of set comprehension just amounts to performing one step of β-reduction: hypothesis (4.9) becomes

$$(\lambda a \colon A.\ f(C)\ f(a))\ x \quad \text{which } \beta\text{-reduces to} \quad f(C)\ f(x)$$

[48]: Church (1940), ‘A Formulation of the Simple Theory of Types’

7: Note that this induces a strict hierarchical notion of set as in Russell’s type theory, where sets containing other sets have a higher-order type, i.e. they correspond to functions taking other functions as arguments.

which we can translate back as $f(x) \in f(C)$. One can consider this as a third kind of definition qualified of *computational*[8].

This encoding of sets has now found its way in the libraries of many proof assistants based on type theory, and is the one used in the `Coq` solution to the exercise provided in annex of [15]. To perform β-reduction in `Coq`, there is a tactic called `simpl` as in "simplify"[9]. Coming back to PbA, one can easily imagine a corresponding Simplify or Compute contextual action, which performs β-reduction inside of the selected subterm. Then the previous transormation is achieved by applying this action on hypothesis (4.9):

$$\boxed{x \in \{a \in A \mid f(a) \in f(C)\}} \qquad (\text{Simplify})$$

8: In `MLTT`, computational definitions are merged with nominal definitions in the concept of judgmental equality. In `Coq` they are distinguished: computational and nominal definitions correspond respectively to β-reduction and δ-reduction.

9: There are also `simp` tactics available in `Isabelle` and `Lean`, although they are not restricted to β-reduction and can perform rewriting of arbitrary equalities and equivalences present in the lemma database, as long as those are flagged as simplification rules.

4.2.5. Finishing the proof

From a user perspective, the two styles of actions induced by the two types of definitions (axiomatic and computational) differ mainly in one respect: while the definition of set comprehension is *implicit* in the type-theoretical encoding, it must be manipulated *explicitly* when using the axiom of comprehension[10]. Depending on the user's background and context of usage, one might prefer one style over the other. Typically in an educational setting, having to manipulate explicitly the axiomatic definition might be more instructive, but also more confusing when carrying formal proofs for the first time.

10: In fact one could also use the axiom of comprehension implicitly by relying on stronger automation. For example in `Isabelle/Isar`, one would write explicitly the desired goal $f(x) \in f(C)$, refer to the axiom by its name in the library, and then let the engine figure out the details of how to apply it. But writing statements manually goes against the philosophy of PbA, which explores to what extent proofs can be carried by pure manipulation of the proof state. Of course there is still an escape hatch for doing this when strictly necessary or more convenient.

Going back to the exercise, we now have the following context of hypotheses:

$$\mathsf{injective}(f) \qquad (4.10)$$

$$f(x) \in f(C) \qquad (4.11)$$

We can unfold the definition of direct image in hypothesis (4.11) the same way we did for the inverse image in hypothesis (4.8): first perform the contextual action

$$f(x) \in \boxed{f(C)} \qquad (\text{Unfold})$$

which gives us

$$f(x) \in \{b \in B \mid \exists a \in A.\, a \in C \wedge b = f(a)\}$$

Then we can simplify the set comprehension with

$$\boxed{f(x) \in \{b \in B \mid \exists a \in A.\, a \in C \wedge b = f(a)\}} \qquad (\text{Simplify})$$

which gives us

$$\exists a \in A.\, a \in C \wedge f(x) = f(a) \qquad (4.12)$$

Now since f is injective, we should be able to deduce that $x = a$. First we unfold the definition of injectivity in hypothesis (4.10):

$$\boxed{\mathsf{injective}(f)} \qquad (\text{Unfold})$$

```
Definition Ens {A : Type} := A -> Prop.

Definition subset {A : Type} (C D : Ens) :=
  forall (x : A), C x -> D x.

Infix "⊆" := subset (at level 30).

Definition image {A B : Type} (f : A -> B) (C : Ens) : Ens :=
  fun y => exists x, C x /\ y = f x.

Definition preimage {A B : Type} (f : A -> B) (C : Ens) : Ens :=
  fun x => C (f x).

Definition injective {A B : Type} (f : A -> B) :=
  forall x x', f x = f x' -> x = x'.

Context (A B : Type).
```

Figure 4.3.: Preliminary definitions in `Coq` of an exercise on abstract functions

which gives us

$$\forall y \in A.\ \forall z \in A.\ f(y) = f(z) \supset y = z \tag{4.13}$$

Then we can use injectivity with the following forward DnD between hypotheses (4.13) and (4.12):

$$\forall y \in A.\ \forall z \in A.\ \boxed{f(y) = f(z)} \supset y = z \quad \circledast \quad \exists a \in A.\ a \in C \wedge \boxed{f(x) = f(a)}$$

which gives us immediately that $x = a$ in

$$\exists a \in A.\ a \in C \wedge x = a \tag{4.14}$$

The last steps consist in clicking on hypothesis (4.14) to introduce a in the context together with

$$a \in C \wedge x = a \tag{4.15}$$

doing a backward DnD with hypothesis (4.15) to rewrite x in the conclusion into a:

$$a \in C \wedge \boxed{x} = a \quad \circleddash \quad \boxed{x} \in C$$

and finally another backward DnD with hypothesis (4.15) to conclude the proof:

$$\boxed{a \in C} \wedge x = a \quad \circleddash \quad \boxed{a \in C}$$

For the sake of completeness, we included in Figure 4.3, Figure 4.4 and Figure 4.5 a `Coq` formalization of the definitions, solution to the first question, and solution to the second question of the exercise, respectively. We simply took the data provided in [15], and added corresponding `Actema` actions below tactic invokations, as in the previous section. It is quite close to the graphical proof just outlined for the second question, hence we do not add further commentary.

```
Theorem subset_preimage_image (f : A -> B) :
  forall C, C ⊆ preimage f (image f C).
Proof.
  intros C.
  (** Click on conclusion *)

  unfold subset.
  (** Select conclusion, then apply Unfold from contextual action menu *)
  intros a H.
  (** Two clicks on conclusion *)

  unfold preimage.
  (** Select conclusion, then apply Unfold from contextual action menu *)
  unfold image.
  (** Select conclusion, then apply Unfold from contextual action menu *)
  exists a.
  split.
  apply H.
  (** DnD of [H] on [C(x)] in conclusion *)
  reflexivity.
  (** Click on conclusion *)
Qed.
```

Figure 4.4.: Solution in `Coq` to the first question of an exercise on abstract functions

```
Theorem preimage_image_subset (f : A -> B) :
  injective f -> forall C, preimage f (image f C) ⊆ C.
Proof.
  intros Hinj C.
  (** 2 clicks on conclusion *)

  unfold subset.
  (** Select conclusion, then apply Unfold from contextual action menu *)
  intros x H.
  (** 2 clicks on conclusion *)

  unfold preimage in H.
  (** Select [H], then apply Unfold from contextual action menu *)
  unfold image in H.
  (** Select [H], then apply Unfold from contextual action menu *)
  destruct H as [x' [H1 H2]].
  apply Hinj in H2.
  (** Here we need to unfold [injective] so that Actema detects a possible DnD action:
      1. Select [Hinj], then apply Unfold from contextual action menu
      2. Click on [H]
      3. DnD of [H] on [f x = f x'] in [Hinj] *)
  rewrite <- H2 in H1.
  (** DnD of last hypothesis on the leftmost [x] in [H] *)
  apply H1.
  (** DnD of last hypothesis on conclusion *)
Qed.
```

Figure 4.5.: Solution in `Coq` to the second question of an exercise on abstract functions

4.3. Peano arithmetic

In our last example, we will analyze a common proof often taught in logic courses: the commutativity of addition in Peano arithmetic. Once again, we invite readers to try to perform the proof themselves in `Actema`[11].

11: `https://www.actema.xyz/courses/peano`

The novelty compared to previous examples is that it involves reasoning by *induction*, which usually has special support in mainstream proof assistants. In the PbA paradigm, it seems natural to map it to a contextual action Induction, whose availability and effect will depend on the selected subterm. One could also imagine manipulating explicitly induction schemes as blue items, similarly to how we manipulated the axiom of comprehension in the previous section. In this section we focus on describing features of the more convenient contextual action.

First and foremost, it only makes sense to apply induction to a variable which is quantified *universally*, either because it appears in the context, or because it is bound by a $\forall$ in the conclusion. In the first case, one can access the contextual action in `Actema` by selecting the green item corresponding to the variable; in the latter case, by selecting the subterm of the conclusion whose head connective is the binding $\forall$[12]. This could also work by selecting any occurrence of the variable, since the system can always infer its binding point. The only condition is that if the variable is bound by a $\forall$, it must occur in a *strictly positive* location, i.e. in the conclusion and not on the left of an implication $\supset$. Obviously the variable should also be of an *inductive* type, i.e. one which is mapped to an induction scheme in the system[13]. Then the Induction action will simply apply the associated induction scheme. In our commutativity example, we can thus start the proof like so:

12: In the standalone version of `Actema`, induction is performed by clicking directly on green items, rather than through a contextual action.

13: The standalone version of `Actema` only supports induction on natural numbers, but in a proof assistant like `Coq` or `Lean` one can easily check if a given type is inductive.

$$\boxed{\forall n \in \mathbb{N}.\ \forall m \in \mathbb{N}.\ n + m = m + n} \qquad (\text{Induction})$$

which performs an induction on n, generating the two following subgoals:

$$\forall m \in \mathbb{N}.\ 0 + m = m + 0 \qquad \text{and} \qquad \forall m \in \mathbb{N}.\ n{\oplus}1 + m = m + n{\oplus}1$$

where the second subgoal has a new variable $n \in \mathbb{N}$ in its context, together with the induction hypothesis

$$\forall x \in \mathbb{N}.\ n + x = x + n \tag{4.16}$$

The proofs of the two subgoals can be done by induction on m. We focus on the second one, which is a bit more involved. As mentioned above, an alternative way of performing induction is to first introduce m in the context by clicking on the conclusion, and then selecting it to access the contextual action:

$$\boxed{m \in \mathbb{N}} \qquad (\text{Induction})$$

which generates again two new subgoals:

$$n{\oplus}1 + 0 = 0 + n{\oplus}1 \qquad \text{and} \qquad n{\oplus}1 + m{\oplus}1 = m{\oplus}1 + n{\oplus}1$$

where the second subgoal has a new variable $m \in \mathbb{N}$ in its context, together with the induction hypothesis

$$n \oplus 1 + m = m + n \oplus 1 \tag{4.17}$$

Let us now focus on the first subgoal. Once again, as with set comprehension in the previous section, the addition operation may have a more axiomatic or more computational definition, depending on the underlying logical framework and library used. For instance in `Coq`'s standard library, it is implemented as a *program* built by recursion on the left-hand argument. Thus in this setting, if one performs computation in the whole conclusion like so:

$$\boxed{n \oplus 1 + 0 = 0 + n \oplus 1} \qquad \text{(Simplify)}$$

this will give a new conclusion

$$(n + 0) \oplus 1 = n \oplus 1$$

where the addition program was able to automatically evaluate $n \oplus 1 + 0$ to $(n + 0) \oplus 1$ on the left-hand side of the equality, and $0 + n \oplus 1$ to $n \oplus 1$ on the right-hand side. If the proof engine does not offer such a level of automation, one can always fallback to using Peano axioms manually, provided that they are available in the lemma database. As we have already seen, the most convenient way to do this in the PbA paradigm is to perform a Search action. For example to evaluate $n \oplus 1 + 0$, one might first select it in the conclusion and then make a search query:

$$\boxed{n \oplus 1 + 0} = 0 + n \oplus 1 \qquad \text{(Search)}$$

Among the results which are Peano axioms, one will only find the following ones:

(a) $\forall x.\ 0 \neq x \oplus 1$

(b) $\forall x.\ \forall y.\ x \oplus 1 = y \oplus 1 \supset x = y$

(c) $\forall x.\ \forall y.\ x \oplus 1 + y = (x + y) \oplus 1$

because the other axioms do not contain any subterm that could form a valid backward linkage with the selection[14]. Of course we are interested in axiom (*c*), which we can use through the following backward DnD:

14: See Section 3.2 for the notion of validity of a linkage.

$$\forall x.\ \forall y.\ \boxed{x \oplus 1 + y} = (x + y) \oplus 1 \quad ⧀ \quad \boxed{n \oplus 1 + 0} = 0 + n \oplus 1$$

in order to rewrite $n \oplus 1 + 0$ into $(n + 0) \oplus 1$ in the conclusion.

Now notice that the addition program earlier was not able to evaluate $n + 0$ to n. This is because 0 occurs on the right-hand side of $+$, and the program is not aware of the commutativity of addition, which is precisely what we are trying to prove. Fortunately, we can apply our induction hypothesis (4.16) on n, with the following backward DnD:

$$\forall x \in \mathbb{N}.\ \boxed{n + x} = x + n \quad ⧀ \quad \left(\boxed{n + 0}\right) \oplus 1 = n \oplus 1$$

which rewrites $n + 0$ into $0 + n$ in the conclusion:

$$(0 + n) \oplus 1 = n \oplus 1$$

Now we can continue the computation:

$$\left(\boxed{0 + n}\right) \oplus 1 = n \oplus 1 \qquad \text{(Simplify)}$$

and conclude the base case of the induction on m by reflexivity, by clicking on the conclusion $n \oplus 1 = n \oplus 1$.

The inductive case of the induction on m works similarly, but obviously we will also need to use the induction hypothesis (4.17) on m. First we compute everywhere we can in the conclusion:

$$\boxed{n \oplus 1 + m \oplus 1 = m \oplus 1 + n \oplus 1} \qquad \text{Simplify}$$

Then we can apply the induction hypothesis (4.16) on n:

$$\forall x.\, \boxed{n + x} = x + n \quad \circledgtr \quad \left(\boxed{n + m \oplus 1}\right) \oplus 1 = (m + n \oplus 1) \oplus 1$$

and also the induction hypothesis (4.17) on m:

$$n \oplus 1 + m = \boxed{m + n \oplus 1} \quad \circledgtr \quad (m \oplus 1 + n) \oplus 1 = \left(\boxed{m + n \oplus 1}\right) \oplus 1$$

Again we compute everywhere:

$$\boxed{(m \oplus 1 + n) \oplus 1 = (n \oplus 1 + m) \oplus 1} \qquad \text{(Simplify)}$$

apply the induction hypothesis (4.16) on n once again:

$$\forall x.\, n + x = \boxed{x + n} \quad \circledgtr \quad \left(\left(\boxed{m + n}\right) \oplus 1\right) \oplus 1 = (n + m \oplus 1) \oplus 1$$

and conclude by reflexivity on $((n + m) \oplus 1) \oplus 1 = ((n + m) \oplus 1) \oplus 1$.

As in previous sections, the interested reader can find a complete `Coq` formalization in Figure 4.6, which follows the same structure as the graphical proof just outlined. In this case the correspondence between `Coq` tactics and graphical actions is almost exact. This suggests that designing a compiler from graphical proofs to `Coq` proof scripts might be a reasonable endeavor. It will indeed be one of the subjects of Chapter 6.

```
Theorem add_comm :
  forall (n m : nat), n + m = m + n.
Proof.
  induction n as [|n IHn].
  (** Select conclusion, then apply Induction from contextual action menu *)

  * induction m as [|m IHm].
    (** Select conclusion, then apply Induction from contextual action menu *)
    - reflexivity.
      (** Click on conclusion *)
    - simpl.
      (** Select conclusion, then apply Simplify from contextual action menu *)
      rewrite <- IHm.
      (** DnD of [IHm] on conclusion *)
      reflexivity.
      (** Click on conclusion *)

  * induction m as [|m IHm].
    (** Select conclusion, then apply Induction from contextual action menu *)
    - simpl.
      (** Select conclusion, then apply Simplify from contextual action menu *)
      rewrite IHn.
      (** DnD of [IHn] on conclusion *)
      reflexivity.
      (** Click on conclusion *)
    - simpl.
      (** Select conclusion, then apply Simplify from contextual action menu *)
      rewrite IHn.
      (** DnD of [IHn] on conclusion *)
      rewrite <- IHm.
      (** DnD of [IHm] on conclusion *)
      simpl.
      (** Select conclusion, then apply Simplify from contextual action menu *)
      rewrite IHn.
      (** DnD of [IHn] on conclusion *)
      reflexivity.
      (** Click on conclusion *)
Qed.
```

Figure 4.6.: Proof of commutativity of addition on natural numbers in `Coq`

Parallel Conclusions and Classical Logic 5.

5.1 Backtracking 76
5.2 Implementation in theorem provers 76
5.3 Click actions 77
5.4 DnD actions 78
5.5 Metatheory of parallel reasoning 79

Run parallel to reality, they symbolize it, they squint at it,
They never touch it: consider what an explosion
Would rock the bones of men into little white fragments and unsky the world
If any mind for a moment touch truth.

Robinson Jeffers, *The Silent Shepherds*, 1958

In virtually every proof assistant, the goals the user is faced with are sequents of the form $\Gamma \Rightarrow C$, with a *single* conclusion C to be proved under many hypotheses in Γ. Historically, this form of sequent was introduced by Gentzen to formalize the rules of intuitionistic logic in his sequent calculus LJ. But his main interest was in classical logic, as intuitionistic logic was still in its infancy and almost all of mathematics had been developed in a classical setting. Interestingly, he found that the right syntax to develop a rich metatheory of his classical sequent calculus LK consisted in *multi-conclusion* sequents of the form $\Gamma \Rightarrow \Delta$, where Δ is a list of conclusions that should be read *disjunctively*. That is, a sequent

$$A_1, \ldots, A_n \Rightarrow C_1, \ldots, C_m$$

has the same meaning as the formula

$$\bigwedge_{i=1}^{n} A_i \supset \bigvee_{j=1}^{m} C_j$$

$$\frac{\Gamma \Rightarrow A, B, \Delta}{\Gamma \Rightarrow A \vee B, \Delta}\ \vee\mathsf{R}$$

$$\frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \bot, \Delta}\ \bot\mathsf{R}$$

Figure 5.1.: Multiplicative right introduction rules for disjunction and absurdity

One way to get a multi-conclusion sequent in LK is to apply the "multiplicative"[1] introduction rule $\vee$R (Figure 5.1). For instance, Figure 5.2 shows a proof of the law of excluded middle, where for each rule we squared the principal formula. One could imagine performing the same proof in `Actema` by successively clicking on these principal formulas, following a bottom-up reading of the sequent calculus derivation seen as the trace of a proof search process. First we decide to prove $A \vee \neg A$: this amounts to proving alternatively A or $\neg A$, which now appear as two separate red items in the same tab. Then we try to prove $\neg A$, with the usual method of supposing A to find a contradiction. However instead of a contradiction, we decide to *backtrack* and prove the alternative conclusion A, which is now trivial by assumption. We come back to these multi-conclusion click actions in Section 5.3.

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\boxed{A} \Rightarrow A}\ \mathsf{ax}}{A \Rightarrow A, \boxed{\bot}}\ \bot\mathsf{R}}{\Rightarrow A, \boxed{\neg A}}\ \supset\mathsf{R}}{\Rightarrow \boxed{A \vee \neg A}}\ \vee\mathsf{R}$$

Figure 5.2.: Proof of the excluded middle in LK

It is clear in the proof that the negative occurrence of A in $\neg A$ is the one that becomes the assumption A that justifies the conclusion A in the last step. It would be natural to specify this causal relationship by linking directly the two occurrences of A, as we do with DnD actions in `Actema`. However for this to be possible, we need to introduce a new interaction operator — let us note it $⦶$ — that works between two conclusions, where $A ⦶ B$ is obviously interpreted as $A \vee B$. Then after clicking on $A \vee \neg A$ we

1: Terminology borrowed from linear logic, where $\vee$R is exactly the right introduction rule for multiplicative disjunction $⅋$.

can just finish the proof by connecting A and $\neg A$:

$$\boxed{A} \varobar \neg \boxed{A} \quad \to^* \quad \top$$

This would avoid the additional step of clicking on $\neg A$ to turn it into an hypothesis, and suggests the possibility of a more general behavior associated to this $\varobar$ operator for arbitrary logical connectives. This is what we explore in Section 5.4.

5.1. Backtracking

Interestingly, the classical aspect of the proof of $A \vee \neg A$ in Figure 5.2 comes exclusively from the *backtracking* operation during the last step, a phenomemon which is well known in the folklore around constructive/computational interpretations of classical logic, and is related to the notion of *continuation* in programming[2]. Then one can see the introduction rules $\vee R_1$ and $\vee R_2$, and the restriction of intuitionistic sequents to one conclusion, as ways to prevent such backtracking by forcing the choice of disjunct to prove at an early stage.

2: See for instance [169, Section 5.6].

In fact backtracking can still be performed in intuitionistic logic, but at the meta-level of the proof search process instead of the object-level of inference rules. In an interactive theorem prover, this corresponds to the ability for the user to *undo* an inference and go back to a previous proof state. However keeping track of the times we undo/redo inferences is very hard to do as humans, and the user interfaces of current proof assistants do not provide any mechanism that helps in this respect. This has already been noted by Ayers[3], and is a good motivation for trying to design proof systems where the need for meta-level backtracking is reduced, or even removed. One way to do this is to maximize the proportion of inference rules that are *invertible*, meaning that their premisses always follow from their conclusion. Indeed when looking at rules as tactics (see Remark 1.2.1), it means that they will always reduce a provable goal to provable subgoals, and thus can never induce a backtracking point[4]. The $\vee R$ rule is an example of invertible rule, and in LK it can replace the other, non-invertible rules $\vee R_1$ and $\vee R_2$.

3: Section 3.1.3 of his thesis [11].

4: Except of course if the user deems the subgoal too complex to prove in its current form, and explicitly wants to backtrack to shape it differently.

But $\vee R$ requires multi-conclusion sequents, and Gentzen restricted their use to classical logic. It turns out that logicians after Gentzen have proposed various multi-conclusion sequent calculi that work for intuitionistic logic, the most famous being GHCP from Dragalin [72], which uses $\vee R$. In the rest of this chapter, we will consider to what extent one can benefit from having multiple conclusions in an intuitionistic setting.

[72]: Dragalin et al. (1990), 'Mathematical Intuitionism. Introduction to Proof Theory'

5.2. Implementation in theorem provers

Despite the aforementioned benefits of multi-conclusion sequents, we do not know of any proof assistant, whether classical or intuitionistic, that exposes them in its user interface. One reason is that most proof/tactic languages are based on the rules of natural deduction, which use single-conclusion sequents. Another reason is that having one conclusion removes the need to designate it with an explicit name or number, as is the case with hypotheses[5]. And the explicit handling of names in tactic invokations is known to be tedious and time-consuming, to the point that some tactic languages like `SSReflect` have been designed around this problem [100]. Thus having multiple conclusions would only double the effort for no compelling reason.

5: The current trend is to have user-chosen or automatically generated strings for names as in `Coq` and `Lean`, but some provers like `HOL Light` ask for the position in the list as an integer to designate a particular hypothesis.

[100]: Gonthier et al. (2016), *A Small Scale Reflection Extension for the Coq system*

However in our graphical paradigm based on direct manipulation, hypotheses and conclusions are designated by the act of *pointing* at them with a mouse, finger or any other pointing device[6]. This opens up the possibility of exposing multiple conclusions in the interface with associated graphical proof actions, as outlined in the introductory example of this chapter. While we did not implement such an extension in `Actema`, we explore in the following sections its design, and the theoretical foundations that lay behind it.

6: With the recent advances in natural language processing and voice recognition, one could also imagine a system based on the selection of subterms by *spelling* their content. Then click and DnD actions could be triggered by voice commands once the subterms they apply to have been selected. This could be an important alternative for users with impaired vision and/or motricity.

5.3. Click actions

In Table 2.1, we showed how click actions in `Actema` are in direct correspondance with the rules of the single-conclusion sequent calculus LJ for intuitionistic logic. Following the literature mentioned earlier, we just need to replace two actions/introduction rules to get a multi-conclusion system capturing either intuitionistic or classical first-order logic:

- clicking on a red disjunction $A \vee B$ breaks it into two conclusions A and B. This is the dual behavior to click actions on blue conjunctions, and corresponds to the $\vee$R rule of Figure 5.1, which is common to both the intuitionistic and classical variants;
- as before, clicking on a red implication $A \supset B$ breaks it into an hypothesis A and a conclusion B. Without further changes, this corresponds to the right introduction rule from the classical sequent calculus LK of Gentzen (named $\supset$R$*$c in Figure 5.3), and our set of actions becomes a proof system for classical logic. To go back to intuitionistic logic, one needs the additional behavior that all the other conclusions of the goal are removed. This corresponds to the right introduction rule from the GHCP calculus of Dragalin (named $\supset$R$*$i in Figure 5.3).

$$\frac{\Gamma, A \Rightarrow B, \Delta}{\Gamma \Rightarrow A \supset B, \Delta}\ \supset\mathrm{R}{*}\mathrm{c}$$

$$\frac{\Gamma, A \Rightarrow B}{\Gamma \Rightarrow A \supset B, \Delta}\ \supset\mathrm{R}{*}\mathrm{i}$$

Figure 5.3.: Multi-conclusion right introduction rules for implication

Remark 5.3.1 In the special case of intuitionistic sequents with one conclusion, the two variants $\supset$R$*$c and $\supset$R$*$i collapse into the usual $\supset$R rule.

Note that we only modified the behavior of the disjunction $\vee$ and implication $\supset$ connectives; and for the latter, only in the case when there are at least two parallel conclusions, and thus implicitly a disjunction. Then it is interesting to notice that the classical behavior of the other connectives ($\bot, \wedge, \forall, \exists$) essentially arises from their interaction with (positive) disjunctive statements.

If we stick to intuitionistic logic, the benefits of having multiple conclusions are unclear. Indeed while the $\vee$R rule is invertible, the $\supset$R$*$i rule is not, and thus at some point the choice of which conclusion to prove must be made by the user irreversibly, even if the choice is delayed[7]. On the other hand the $\supset$R$*$c rule *is* invertible: this is known to allow the formulation of sequent calculi for classical logic where *all* rules are invertible, like the G3c calculus of [183]. In the propositional case, this gives a constructive decision procedure for the question of provability: given a sequent $\Gamma \Rightarrow \Delta$, one just has to choose any formula in Γ or Δ and apply the introduction rule associated to its main connective, or the axiom rule whenever possible. In `Actema`, this would correspond to having the user click randomly on blue and red items until all goals are solved. The procedure ends because all introduction rules destroy the main connective, and none of them duplicate formulas: thus the total number of connectives in the sequent decreases strictly after each rule application.

7: In Section 8.3, we will see how to overcome this limitation by using a *nested sequent* system.

[183]: Negri et al. (2001), *Structural Proof Theory*

When dealing with quantifiers, the situation is not so simple: if one wants invertible introduction rules, it is necessary to duplicate the quantified formula being instantiated, which can be seen as the root cause of undecidability in predicate logic as noted by Girard [94, Section 3.3.2]. This is already what happens in `Actema` for the universal quantifier: dropping a term t on a blue item $\forall x.A$ will produce a new hypothesis $A\{t/x\}$, while keeping the original $\forall x.A$ item. This corresponds to the invertible left introduction rule of G3c ($\forall$L$*$ in Figure 5.4). But in the single-conclusion framework, dropping a term t on a red item $\exists x.A$ necessarily replaces it by the instantiated conclusion $A\{t/x\}$. Allowing multiple conclusions circumvents this problem and restores the symmetry between $\forall$ and $\exists$, since we can create a new conclusion for $A\{t/x\}$ while preserving the old one. This corresponds to the invertible right introduction rule of G3c ($\exists$R$*$ in Figure 5.4).

$$\frac{\Gamma, \forall x.A, A\{t/x\} \Rightarrow \Delta}{\Gamma, \forall x.A \Rightarrow \Delta}\ \forall\mathsf{L}*$$

$$\frac{\Gamma \Rightarrow A\{t/x\}, \exists x.A, \Delta}{\Gamma \Rightarrow \exists x.A, \Delta}\ \exists\mathsf{R}*$$

Figure 5.4.: Multi-conclusion instantiation rules for quantifiers

5.4. DnD actions

Once we allow for more than one conclusion, it is natural to wonder whether it makes sense to also allow for DnD actions between two conclusions. But we already capture *backward* reasoning with the $A \circleddash B$ operator between a hypothesis A and a conclusion B, and *forward* reasoning with the $A \circledast B$ operator between two hypotheses. There does not seem to be room for a third mode of reasoning, at least in the traditional way of building proofs we are used to, either on paper or with proof assistants. However from a purely formal point of view, there is nothing preventing us from trying to design rewriting rules for a new interaction operator, by following the same recipe we used for $\circleddash$ and $\circledast$. Furthermore, we already saw earlier in the excluded middle example that such an operator did seem useful.

When looking for a proof of a sequent with multiple conclusions, unless we commit to proving one conclusion and give up on the others by applying the $\supset$R$*$i rule, we can switch freely our focus between the different conclusions. Thus in a way, we are looking for proofs of all the conclusions *in parallel*, and we stop as soon as we find one[8]. Hence

8: This is also related to the $⅋$ connective of linear logic (LL) which uses the $\vee$R rule of Figure 5.1, and whose spelling “par” is historically motivated by an understanding of the multiplicative fragment of LL as a logic of parallel computation [92].

we chose to call this third kind of interaction between two conclusions *parallel* reasoning. This justifies the choice of notation for the parallel interaction operator $\obar$, which is suggestive of the parallel composition $|$ from process calculi.

The rules governing $\obar$ are presented in Figure 5.5. A parallel linkage can be created either by drag-and-dropping two conclusions together, or through an instance of the new backward rule $\mathsf{L}{\supset_1}$. It is important to note that this rule is only sound *classically*; indeed we can now come back to the example of Subsection 3.2.1 and give the following derivation of Peirce's law with it:

$$
\begin{array}{rcll}
\left(\boxed{A} \supset B\right) \supset A \obslash \boxed{A} & \to & \left(\boxed{A} \supset B \obar \boxed{A}\right) \wedge (A \supset A) & \mathsf{L}{\supset_1} \\
 & \to & \left(\left(\boxed{A} \obslash \boxed{A}\right) \vee B\right) \wedge (A \supset A) & \mathsf{P}{\supset_1} \\
 & \to & (\top \vee B) \wedge (A \supset A) & \mathsf{id} \\
 & \to & \top \wedge (A \supset A) & \mathsf{absl} \\
 & \to & A \supset A & \mathsf{neul}
\end{array}
$$

Backward

$$
B \supset C \obslash A \;\to\; (B \obar A) \wedge (C \supset A) \qquad \mathsf{L}{\supset_1}
$$

Parallel

$$
\begin{array}{rcll}
A \obar (B \wedge C) & \to & (A \obar B) \wedge C & \mathsf{P}{\wedge_1} \\
A \obar (C \wedge B) & \to & C \wedge (A \obar B) & \mathsf{P}{\wedge_2} \\[1ex]
A \obar (B \vee C) & \to & A \obar B & \mathsf{P}{\vee_1} \\
A \obar (C \vee B) & \to & A \obar B & \mathsf{P}{\vee_2} \\[1ex]
A \obar (B \supset C) & \to & (B \obslash A) \vee C & \mathsf{P}{\supset_1}{}^* \\
A \obar (C \supset B) & \to & C \supset (A \obar B) & \mathsf{P}{\supset_2}{}^* \\[1ex]
A \obar (\forall x.B) & \to & \forall x.(A \obar B) & \mathsf{P}\forall\mathsf{s}^* \\[1ex]
A \obar (\exists x.B) & \to & A \obar B\{x/t\} & \mathsf{P}\exists\mathsf{i} \\
A \obar (\exists x.B) & \to & \exists x.(A \obar B) & \mathsf{P}\exists\mathsf{s}^* \\[1ex]
B \obar A & \to & A \obar B & \mathsf{Pcomm}
\end{array}
$$

In the rules $\{\mathsf{P}\forall\mathsf{s}, \mathsf{P}\exists\mathsf{s}\}$, x is not free in A.

Figure 5.5.: Parallel linking rules

The other rules of Figure 5.5 handle the rewriting of parallel linkages, and were conceived as the dual counterpart of forward rules. Indeed, while a forward linkage combines two negative subformulas in the same context to produce a new *hypothesis*, a parallel linkage combines two positive subformulas in the same context to produce a new *conclusion*. Backward linkages can then be seen as mediating between these two opposite modes of reasoning, by handling the interaction of a positive and a negative subformula in the same context. A schematic view of the back and forth between the different modes through the 4 rewriting rules that change interaction operators is provided in Figure 5.6. Notice that the latter correspond exactly to the rules that handle interaction with the antecedant of an implication: this is because it is the only way to switch polarity when descending into a direct subformula, which is what triggers the change of mode.

$$
\begin{tikzcd}
\circledast \arrow[r, bend left, "\mathsf{F}{\supset_1}"] & \obslash \arrow[l, bend left, "\mathsf{R}{\supset_1}"] \arrow[r, bend left, "\mathsf{L}{\supset_1}"] & \obar \arrow[l, bend left, "\mathsf{P}{\supset_1}"]
\end{tikzcd}
$$

Figure 5.6.: Alternating structure between reasoning modes

5.5. Metatheory of parallel reasoning

Like backward rules, a parallel rule $A \to B$ will be logically sound if B entails A. Then if one wants to stick to an intuitionistic setting, one has to remove the rules $\mathsf{P}{\supset_1}$, $\mathsf{P}{\supset_2}$ and $\mathsf{P}\forall\mathsf{s}$ from the system, in addition to the $\mathsf{L}{\supset_1}$ rule. Indeed those are all sound classically but not intuitionistically[9].

Now if we look back at the schema from Figure 5.6, removing $\mathsf{L}{\supset_1}$ and $\mathsf{P}{\supset_1}$ in particular has the consequence of isolating completely parallel reasoning from the other modes. But remember from Section 3.3 that we are only interested in *valid* linkages, that is those linkages who satisfy productivity (Theorem 3.5.2) and thus will always terminate on an instance of either the id rule (backward mode) or the equality rules (backward/forward modes). Thus if there is no way to reach either forward or backward mode from a parallel linkage, it has no meaning in our paradigm. Then if we trust that rules $\mathsf{L}{\supset_1}$ and $\mathsf{P}{\supset_1}$ are necessary to get the intended semantics,

9: We do not prove this formally here, but this can be done by exhibiting intuitionistic counter-models in which the entailments are false, i.e. Kripke structures or Heyting algebras.

it seems that parallel reasoning only makes sense in a classical setting. In the following, we only show that the rules of Figure 5.5 are sufficient for our purpose, by extending the results of Chapter 3 to the classical, multi-conclusion setting.

We start by updating the validity criterion on linkages, more specifically we drop Clause 2 of Condition 3.2.1 about the polarities of linked subformulas. Indeed it was introduced precisely to forbid behaviors which only make sense in classical logic, but are now given a semantics with the $\mathsf{L}\supset_1$ rule. We also need to add a case for the $⦶$ operator in the other clauses of Condition 3.2.1, which gives the following updated condition:

Condition 5.5.1 (Classical Polarity) The following must be true for a logical linkage $B\boxed{A} @ D\boxed{A'}$ to be *classically valid*:

1. the parity of $\mathsf{inv}(B\square)$ is:
   a) the same as $\mathsf{inv}(D\square)$ if $@ = ⧀$
   b) the opposite of $\mathsf{inv}(D\square)$ if $@ = ⊛$
   c) the opposite of $\mathsf{inv}(D\square)$ if $@ = ⦶$

The following must be true for a rewrite linkage $B\boxed{t} @ D\boxed{t'}$ to be *classically valid*:

1. if $B\square$ holds the equality, then it must be:
   a) positive if $@ = ⧀$;
   b) positive if $@ = ⊛$;
   c) negative if $@ = ⦶$;
2. if $D\square$ holds the equality, then it must be:
   a) negative if $@ = ⧀$;
   b) positive if $@ = ⊛$.
   c) negative if $@ = ⦶$.

Then we add the following case to the statement of Theorem 3.4.1:

$$\text{If } A ⦶ B \to^* C\text{, then } C \Rightarrow A, B.$$

and we interpret $⦶$ as disjunction:

$$\lfloor A ⦶ B \rfloor = \lfloor A \rfloor \vee \lfloor B \rfloor$$

We add the two following cases to Lemma 3.4.3 and Lemma 3.4.4:

- If $C^+\boxed{A ⦶ B} \to D$ then $\lfloor D \rfloor \Rightarrow C^+\boxed{A \vee B}$.
- If $C^-\boxed{A ⦶ B} \to D$ then $C^-\boxed{A \vee B} \Rightarrow \lfloor D \rfloor$.

The proof of Lemma 3.4.3 is easily extended by inspecting each parallel rule, and we already mentioned that the backward rule $\mathsf{L}\supset_1$ is sound classically. Note that now sequents have multiple conclusions, thus one needs to use rules from a multi-conclusion calculus such as G3c [183].

Polarity preservation (Fact 3.4.1) is also true with $\obar$, we just need to add the missing cases from Figure 5.6:

- If $C\boxed{A \obar B} \to C'\boxed{A' \obar B'}$ then $C\square$ and $C'\square$ have the same polarity.
- If $C\boxed{A \ogreaterthan B} \to C'\boxed{A' \obar B'}$ (resp. $C\boxed{A \obar B} \to C'\boxed{A' \ogreaterthan B'}$) then $C\square$ and $C'\square$ have the same polarity.

The proof of Lemma 3.4.4 is also extended straightforwardly. We only write the added case for $\obar$ in the proof of the first statement:

- $@ = \obar$: by Fact 3.4.1, C' must be positive. Therefore by induction hypothesis $\lfloor D \rfloor \Rightarrow C'\boxed{A' \vee B'}$. By Lemma 3.4.3 we have $C'\boxed{A' \vee B'} \Rightarrow C^+\boxed{A \supset B}$. Thus by transitivity $\lfloor D \rfloor \Rightarrow C^+\boxed{A \supset B}$.

Regarding completeness (Theorem 3.7.1), we already noticed that our rules now allow us to prove Peirce's law, which is known to be sufficient to recover classical logic from intuitionistic logic.

The proof of productivity (Theorem 3.5.2) is again extended straightforwardly, by considering the additional case of parallel linkages and using arguments "dual" to those used for forward linkages. There is even less work to do regarding the preservation of Condition 3.2.1 since we dropped the intuitionistic restriction.

Finally about focusing (Section 3.6), we can just remark that some rules that were not invertible in intuitionistic logic become invertible in classical logic. Therefore the dynamics of focusing should be different, and it might be interesting to compare the behaviors of intuitionistic and classical DnD actions on specific examples.

Integration in a Proof Assistant 6.

6.1 Actema 83
6.2 Why a plugin? 84
6.3 The coq-actema system 84
6.3.1 Actema web app 85
6.3.2 Coq plugin 86
6.4 Interaction protocol . . 87
6.4.1 Translating goals 91
6.4.2 Retrieving actions 91
6.5 Compiling actions . . . 94
6.5.1 The `action` datatype . . 95
6.5.2 Deep inference semantics 96
6.6 Future works 97
6.6.1 Higher-order logic 97
6.6.2 Notations 98
6.6.3 Lemma search 99
6.6.4 Proof evolution 99

> God does not care about our mathematical difficulties. He integrates empirically.
>
> **Albert Einstein**

In the previous chapters, we introduced the Proof-by-Action paradigm (Chapter 2), and tried to convince the reader that it is both theoretically sound with its firm grounding in deep inference proof theory (Chapter 3 and Chapter 5), and practically useful by analyzing proofs of mathematical problems expressed within it (Chapter 4). We also mentioned multiple times our prototype of interface implementing PbA called `Actema`, and in particular the fact that it exists as a *standalone* web application with its own proof engine [69]. This is convenient for distributing it online as a publicly available website, so that people can immediately try it out without the hassles of installation procedures. However due to both historical choices in its design and lack of human resources for development, `Actema`'s proof engine is quite limited in its features:

- it can only handle goals expressed in many-sorted *intuitionistic first-order logic* (iFOL), whereas all state-of-the-art PAs support *higher-order* logic in one form or another; and higher-order features are crucial for formalizing many mathematical notions in a concise way, as witnessed by the example of Section 4.2;
- it does not implement a *certified kernel* for checking proof objects, which makes it hard to trust and interoperate with;
- it has no mechanism for adding new *mathematical notations*, only ad hoc support for arithmetical expressions. Thus formulas become very quickly impossible to read and manipulate;
- it has poor support for managing *libraries* of definitions, lemmas and proofs, partly because of the previous items.

To address these limitations, and thus enable a confrontation of the PbA paradigm to real mathematical developments, we decided to build `coq-actema`, a `Coq` plugin that directly connects `Actema` to a running instance of the `Coq` proof assistant [68]. The idea is that `Actema` should act as an enhanced graphical, interactive proof view that integrates in the usual text-based workflow of proof scripts. Thus instead of trying to turn `Actema` into a full-fledged PA, we exploit the over 30 years of effort that have been put in the development of `Coq`, and limit the role of `Actema` to that of a novel frontend for building proofs in `Coq`. This shall open the way to more advanced experimentations through the huge body of theories already developed in `Coq`, and make the PbA paradigm visible to the large community of existing users of this popular PA.

Remark 6.0.1 The same approach should be applicable in principle to any ITP that supports at least iFOL, and provides an interaction protocol for building proofs in a goal-directed manner. This includes other popular PAs such as `Lean` and `Isabelle`, but also more specialized software like the `Why3` platform for deductive program verification [21], the `Meta-F*` framework in the proof-oriented programming language `F*` [165], or the `EasyCrypt` toolset for the verification of cryptographic protocols [14].

The chapter is organized as follows: we start in Section 6.1 by explaining the architecture of the `Actema` web application, which follows the standard conceptual separation between frontend and backend. In Section 6.2, we reflect on some considerations that led us to the specific choice of a `Coq` plugin, in order to integrate `Actema` with `Coq`. Then in Section 6.3 we present the architecture of the `coq-actema` system, which structures all interactions between the user, `Coq` and `Actema`. Section 6.4 describes in more details the main usage scenario of `coq-actema`, following the flow of data and control between the different processes involved. Section 6.5 explains how the various graphical actions performed by the user in `Actema` are compiled into `Coq` tactics, ultimately producing certified proof terms. Finally in Section 6.6, we discuss possible avenues for extending the usability of `coq-actema` to a broader class of `Coq` goals, as well as prospective solutions to the problem of proof evolution in our graphical paradigm.

6.1. Actema

At its core, `Actema` is a web application made of two components: a *frontend* that implements the graphical interface with which the user interacts, written in `HTML`/`CSS`/`JavaScript` with the `Vue.js` framework [240]; and a *backend* that implements the proof engine, written in `OCaml` and compiled to `JavaScript` (`JS`) with `js_of_ocaml` [246]. The two components interact through an object-oriented API written in `OCaml`, which is loaded at runtime in the form of a `JS` object called `engine`, and whose methods can be called from the `Vue` components in the frontend.

[246]: Vouillon et al. (2014), 'From bytecode to JavaScript'

The `engine` object provides various high-level methods for handling the current *proof state*. Common operations include getting the list of open subgoals, querying available proof actions on a subgoal, or applying a given proof action. Lower-level methods are also available in other objects to inspect the data of the proof state. For instance,

```
engine.subgoals[0].context[0]
```

will return an object representing the first hypothesis of the first subgoal; and this object itself exposes an `html` method, which returns a string holding the `HTML` code used to display the statement of the hypothesis.

In the standalone version of `Actema`, the proof engine takes care of computing the new subgoals stemming from actions performed by the user. It is thus responsible for defining the *semantics* of proof actions. It is also in charge of various other tasks that process the logical data of the proof state, typically checking the validity of linkages during a DnD action, which requires the use of a unification algorithm (see Section 3.2).

6.2. Why a plugin?

Usually, *integrated development environments* (IDEs) for `Coq` live in an independent process, and exchange data with `Coq` through a high-level communication protocol: either the command line interface provided by `coqtop`, `Coq`'s default `XML` protocol, or its improved superset `SerAPI` [85]. In particular, `SerAPI` emerged from the development of `jsCoq` [7], an IDE that runs entirely in web browsers by embedding a version of `Coq` compiled with `js_of_ocaml`. Since `Actema` is also web-based and uses `js_of_ocaml`, our first idea was essentially to fork `jsCoq` and replace its interface by that of `Actema`. However as noted by E. J. G. Arias, the `SerAPI` protocol — and in fact all the other protocols turn out to be too high-level for our purpose. Typically we need to (partially) translate `Coq` goals into iFOL goals, which can be done much more easily with a direct access to `Coq`'s low-level API for manipulating kernel terms.

[85]: Gallego Arias (2016), 'SerAPI: Machine-Friendly, Data-Centric Serialization for COQ'

[7]: Arias et al. (2017), 'jsCoq: Towards Hybrid Theorem Proving Interfaces'

Now, remember that `Actema` is not meant as a full-fledged IDE that can manage the edition and execution states of the proof script, but only as an enhanced proof view for manipulating already-parsed goals. One should think of `Actema`'s actions simply as a graphical frontend for invoking a new set of tactics. And this is precisely what the plugin system of `Coq` has been designed for: extending `Coq` with new tactics. Thus the solution of a `Coq` plugin made a lot more sense, with the important benefit of ensuring compatibility with all existing IDEs. This would also entail easier adoption of `Actema` into existing `Coq` developments and workflows.

In this setting, it is now the `Coq` plugin which implements the semantics of proof actions as new tactics, instead of `Actema`'s backend. This allows us to leverage the facilities already provided by `Coq` to handle the proof state and generate proof terms in the calculus of inductive constructions. This does not make the backend of `Actema` completely irrelevant however: we still need it so that `Actema` can maintain its own, first-order version of the proof state, with additional metadata used to display and interact graphically with objects and statements. Also tasks related to the querying of both display data and proof actions, like the `html` method and unification algorithm mentioned in the previous section, are at the time of writing of this thesis still performed in `Actema`'s backend. It is unclear to what extent this should rather be a responsibility of the `Coq` plugin, relegating `Actema` to a pure role of frontend to the PA[1].

1: For instance in the `ProofWidgets` framework [12], all these tasks are implemented in the meta-programming language of `Lean` (which is `Lean` itself), making it more extensible by (expert) users of the PA.

6.3. The coq-actema system

Let us now give the full picture of the `coq-actema` system that integrates both `Actema` and the `Coq` plugin. A schematic view of its overall architecture, including the various components and their relationships, is provided

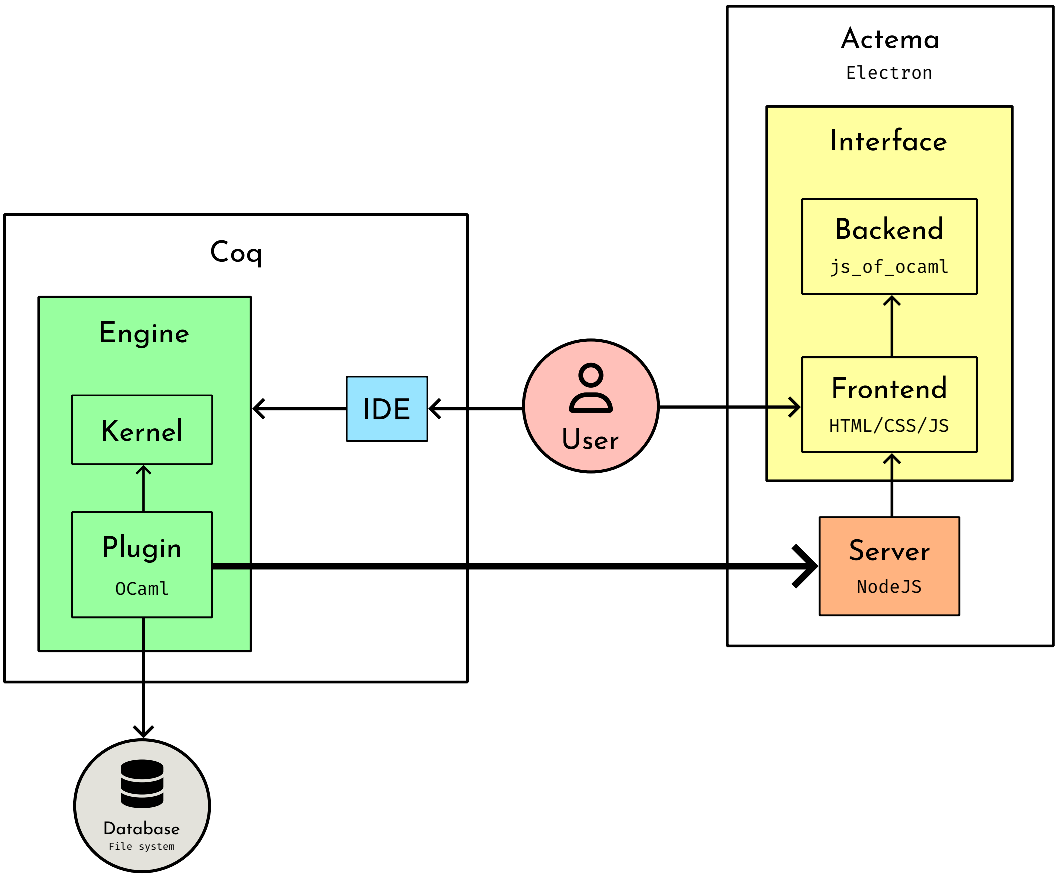


Figure 6.1.: Architecture of the `coq-actema` system

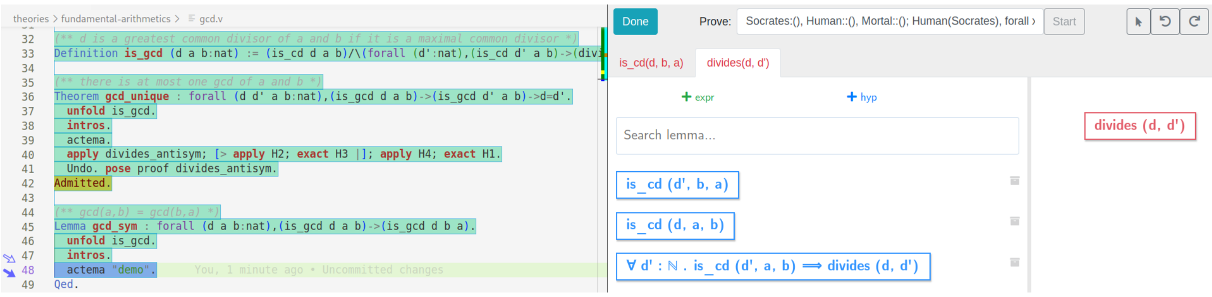
in Figure 6.1. The different *processes/agents* involved are represented by shapes of different *colors*, and we add a directed *arrow* whenever two of them communicate with each other, where the *source* requests data from or sends instructions to the *target*.

The User (pink circle) is the only human agent in the system, and drives all interactions. She interacts with the `Coq` and `Actema` subsystems (transparent rectangles), through the interfaces provided by her `Coq` IDE of choice (blue rectangle) and `Actema`'s Frontend (yellow rectangle). This will typically take the form of a two-windows layout, as depicted by the screenshot of Figure 6.2.

6.3.1. Actema web app

The `Actema` web app runs in a process independent from `Coq`, represented by the yellow Interface rectangle. We add a third layer to the Frontend and Backend described in Section 6.1, namely a `HTTP` Server (orange rectangle) that handles requests from, and responses to the `Coq` Plugin. Thus we implement interprocess communication between `Actema` and `Coq` through the network layer of the operating system, rather than a more local mechanism such as Unix pipelines. There are a few reasons behind our choice of the `HTTP` protocol:

- it provides useful abstractions when working with a client/server



On the left, the usual interactive view of the proof script, in the VsCoq IDE [50]. On the right, the graphical proof view of Actema.

Figure 6.2.: Graphical layout of the `coq-actema` system

architecture structured around requests;

- it is a widely spread standard, especially in web technologies. Thus we were able to reduce development time by reusing generic implementations of both client and server from standard libraries;
- more anecdotically, this makes it easy to run Coq and Actema on different machines connected on the same network. This could be used for instance to offload heavy computations in a proof to the machine running Coq, while still being able to interact with Coq through Actema on the weaker machine.

The Server runs in a process separate from the Interface, in order to avoid any delay in the latter. Then we bundle everything in an Electron application [195], so that Actema can easily be run locally on most operating systems. This also allows us to exploit the multi-process architecture of Electron [194], where the so-called *main* process runs the server and has the ability to issue system calls for networking through the Node.js HTTP library [196]; and the so-called *renderer* process runs the Interface in the Chromium browser.

6.3.2. Coq plugin

The Plugin is loaded dynamically in Coq's Engine (green rectangle) by executing the following command in a proof script:

```
From Actema Require Import Loader.
```

It exposes a single tactic called `actema`, which can run in two distinct modes:

Interactive The Plugin sends the current subgoals to Actema, and the user applies a sequence of actions on them. Each time an action is performed, it is sent back to Coq, compiled into the appropriate tactic call, and then executed to generate new subgoals that are sent again to Actema. The `actema` tactic finishes its execution either when:

- all subgoals are proved (in Actema);
- the User decides to stop and give back control to the IDE;

- in some rare cases, an unrecoverable error occurs.

Non-interactive If the `actema` tactic has already been executed on the subgoal under focus, then the Plugin automatically saved the sequence of actions performed by the User in a Database (gray circle). Currently for ease of development, the Database is implemented as a simple directory on the local filesystem, where each file encodes an entry as follows:

- the filename is a hash code that uniquely identifies the goal by both its *content*, i.e. the statements of the hypotheses and conclusion, and an optional *identifier*, which can be given as argument to the tactic in the form of an arbitrary string;
- the contents of the file is a `Base64` encoding of the data specifying each action, whose format will be detailed in Section 6.5.

Then the tactic will load the sequence of actions from the appropriate file, recompile it into one big tactic, and execute it on the current subgoal.

One can also force the execution in interactive mode by using a variant of the tactic named `actema_force`. We provide details of the complete interaction protocol followed by the `actema` tactic in the following section.

Regarding the implementation of the Plugin, we chose to do it in the standard way by interfacing with the `coq-core` API in `OCaml` [237], although it has been encouraged in recent versions of `Coq` to interface with more stable APIs such as those provided by `Coq-Elpi` [235] and `MetaCoq` [228][2]. The main reason is that our plugin performs *side effects* by interacting with an external environment: the file system when saving and retrieving graphical proofs, and the network when issueing `HTTP` requests to `Actema`. Those cannot be implemented in the aforementioned frameworks.

[228]: Sozeau et al. (2020), 'The MetaCoq Project'

2: Indeed, breaking changes are frequently introduced in `coq-core` with newer versions of `Coq`, which requires more maintenance efforts from plugin developers.

6.4. Interaction protocol

We will now unroll the details of a complete interaction in `coq-actema`, starting from the User calling the `actema` tactic in her IDE, and ending with her viewing the new subgoals displayed in the IDE. We chose to represent this with a *sequence diagram*, as specified by the `UML` standard [254]. This kind of diagram is used to depict runtime behavior of a system, showing interactions between objects and the messages they exchange in the order they occur chronologically. In our case, the objects are the different processes described in the previous section, as well as the User. Since the full interaction is quite involved, we split the diagram in three parts: Figure 6.3 includes the beginning of the interaction, focusing on the non-interactive mode of the `actema` tactic where the Plugin communicates with `Coq`'s kernel and the Database. Figure 6.4 and Figure 6.5 tackle the interactive mode of the `actema` tactic: Figure 6.4 focuses on the conditions for breaking out of the interaction loop, and Figure 6.5 on the interactions at work when the User performs a proof action in `Actema`.

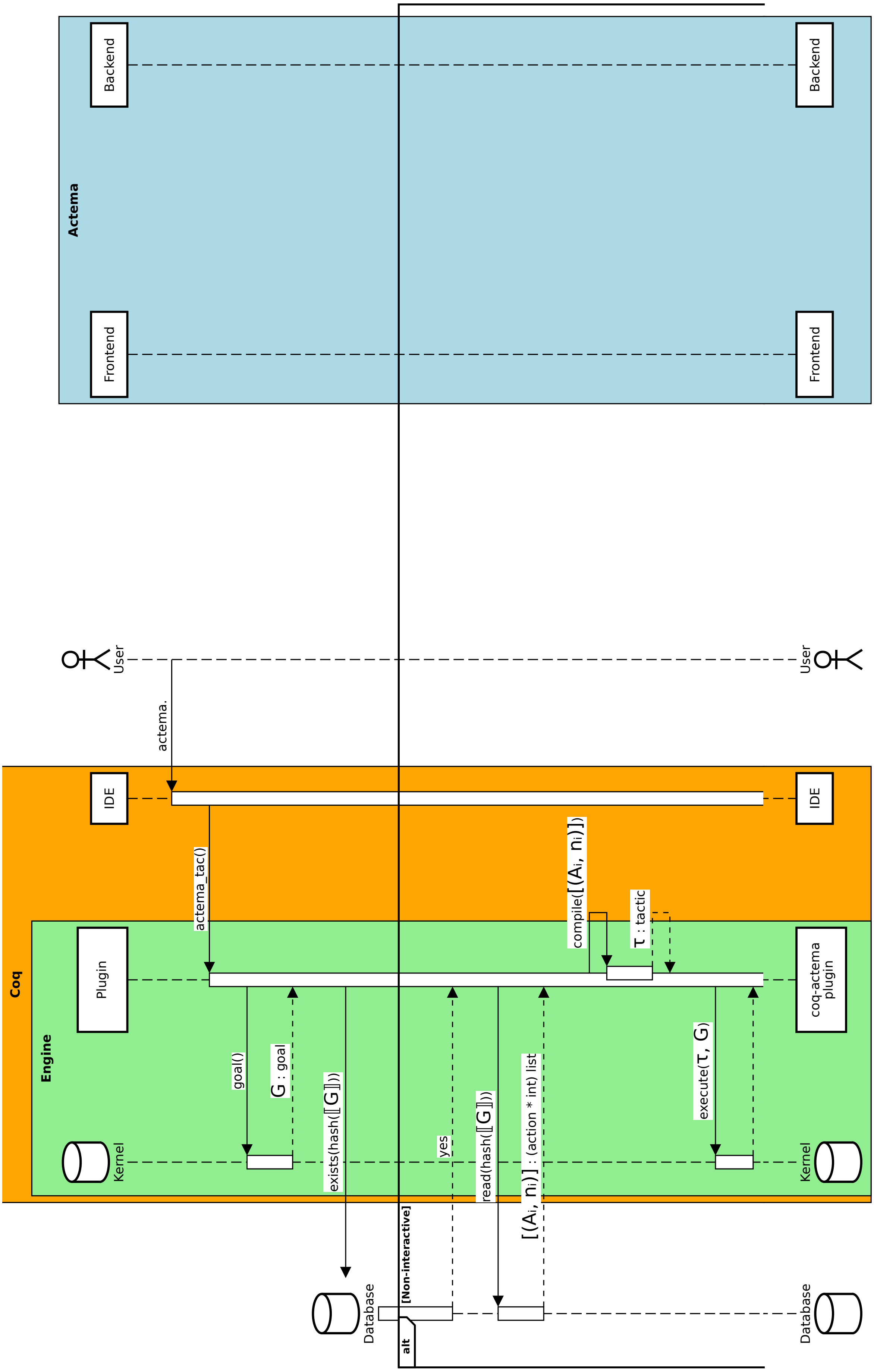


Figure 6.3.: Sequence diagram of `coq-actema`'s interaction protocol — non-interactive mode

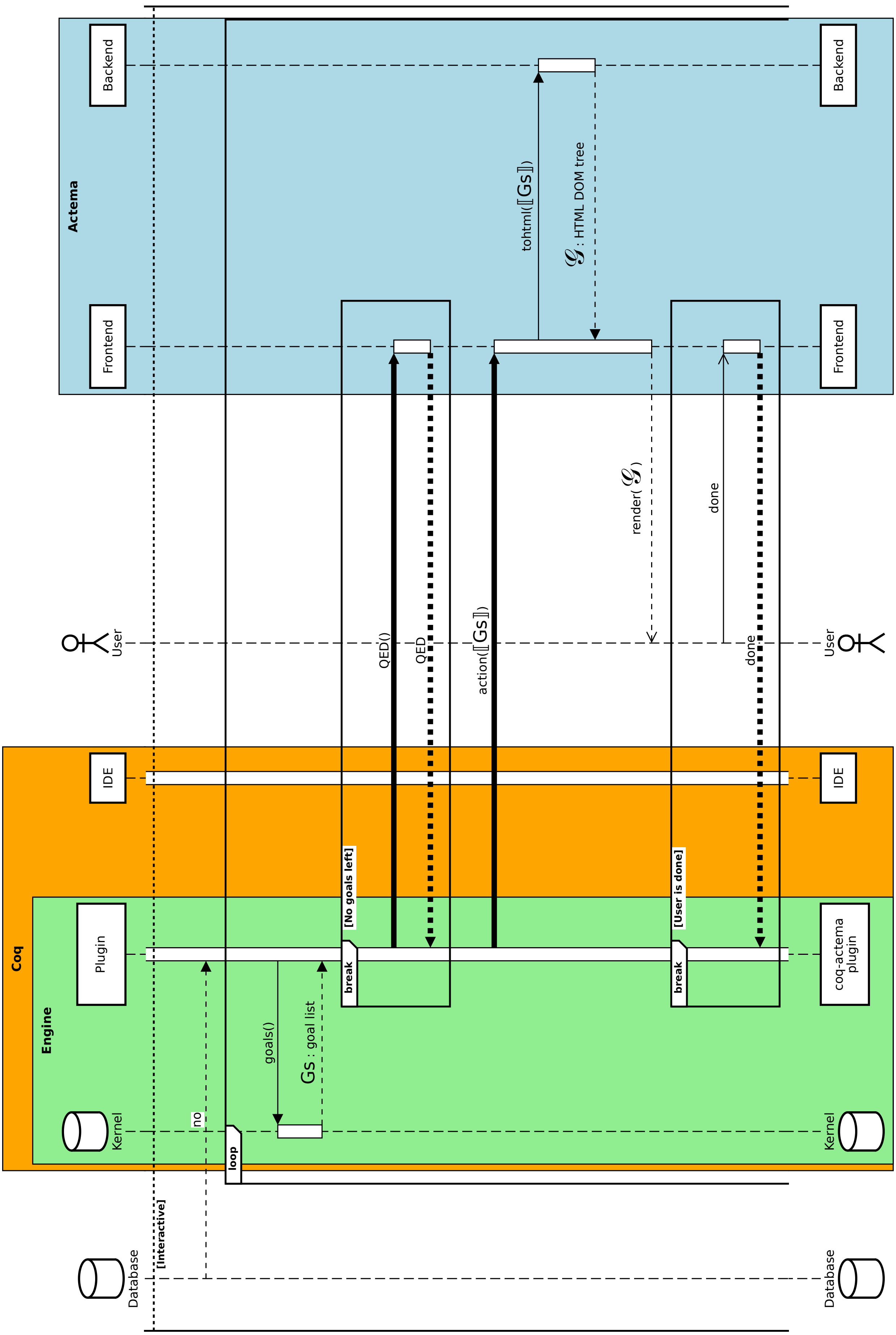


Figure 6.4.: Sequence diagram of `coq-actema`'s interaction protocol — breaking out of the interaction loop

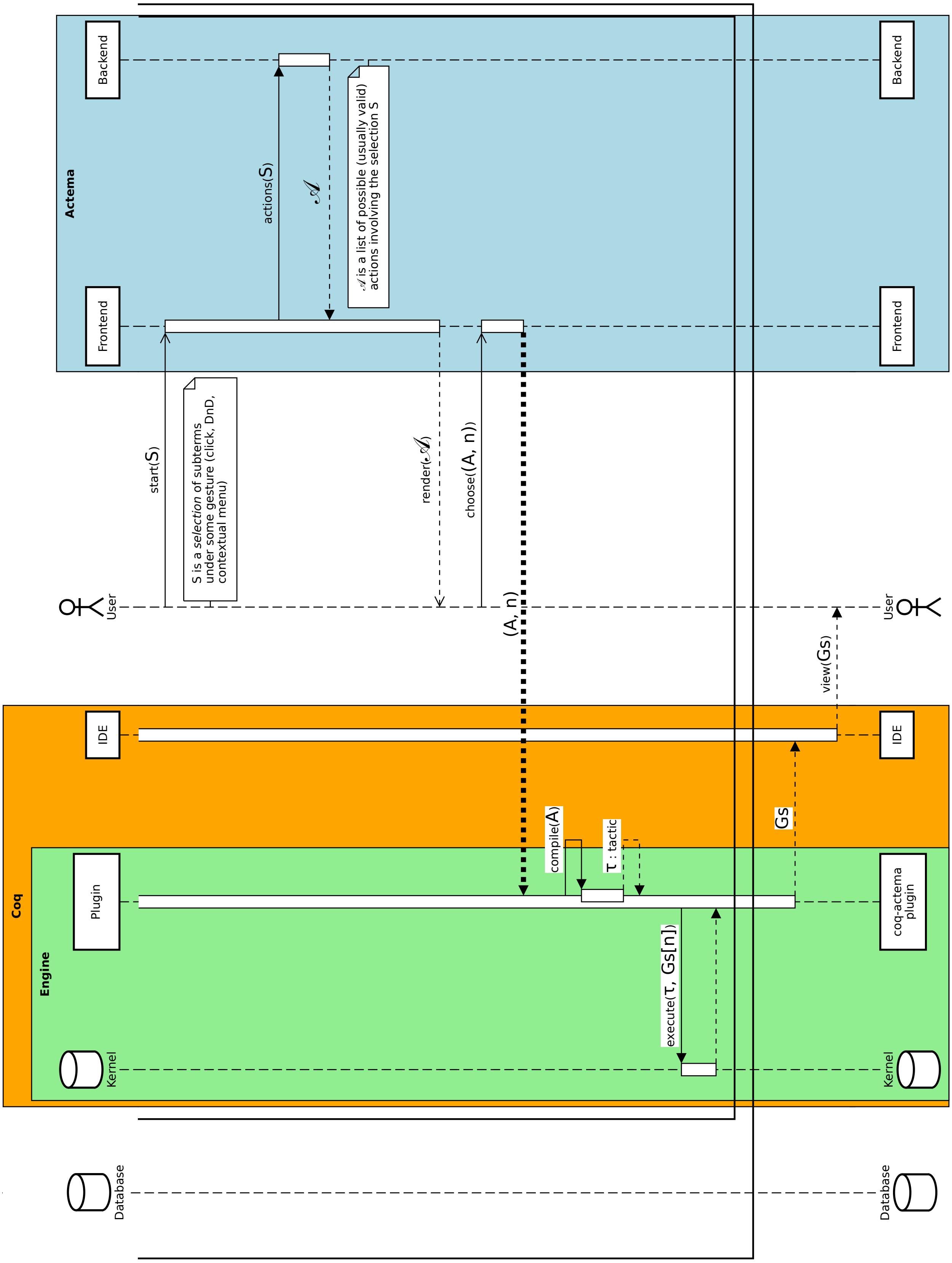


Figure 6.5.: Sequence diagram of `coq-actema`'s interaction protocol — applying an action

6.4.1. Translating goals

The first task performed by the Plugin when calling the `actema` tactic is to ask the kernel for the data of the subgoal G currently under focus. Then for the goal G to be understandable by the Backend of `Actema`, the Plugin will translate it into a new representation $[\![G]\!]$ in a custom datatype. In order to share the definition of this datatype across implementations of the Plugin and Backend, we decided to use the `ATD` data specification language [163]. It provides a set of tools to automatically generate idiomatic datatype definitions in a few target languages — including `OCaml` — along with (de)serialization and validation helpers. This is particularly fit for our usecase, where we need to serialize complex data like $[\![G]\!]$ in order to transmit it over `HTTP` messages. Since both the Plugin and Backend are written in `OCaml`, it also allows us to share across implementations our own domain-specific helpers for manipulating this data.

The `ATD` definition of goals is given by the `goal` type in Figure 6.7[3]. It relies on the `ATD` types `form` of *formulas*, and `env` of *environments* of available constants and variables. In our setting, these correspond respectively to the formulas and *signatures* of many-sorted FOL, whose `ATD` definitions are given in Figure 6.6. But one could imagine using formulas and environments in higher-order logic instead, and this would not change the structure of the `goal` datatype. Note that hypotheses are encoded with a `h_id` attribute corresponding to their string identifier in the `Coq` goal, even though we do not display it in `Actema`'s interface. This is required later by the plugin to compile actions into tactics, because we need to identify which `Coq` hypotheses correspond to those designated graphically by the User.

3: The syntax is almost the same as that of `OCaml` datatypes, for the reader already acquainted with this language.

6.4.2. Retrieving actions

The next step for the Plugin is to check if there already exists a graphical proof associated to $[\![G]\!]$ in the Database. If so, then it retrieves it in the form of a list A of *actions*, whose data format will be precised in Section 6.5. There is also a positive integer n_i associated to each action A_i in the list, corresponding to the index of the goal to which A_i applies in the list of subgoals. Then each A_i is compiled into a tactic $(\!|A_i|\!)$, and all the $(\!|A_i|\!)$ are composed into a unique tactic τ, which is executed by the kernel on G to apply the full sequence of actions.

Remark 6.4.1 Currently the translation $[\![-]\!]$ is not *injective*: it might map two different `Coq` goals to the same `Actema` goal, because strictly higher-order subterms are translated into a `dummy` atomic predicate/-function. Thus one might retrieve a proof from a different goal when calling the `actema` tactic. However this is not problematic, since the User cannot perform any actions involving the parts of the two goals that make them distinct; they might as well be seen as the same goal from her point of view. It can thus be considered as a feature that maximizes proof reuse.

```
(* ------------------------------------------------------------------ *)
(** Identifiers *)

type name = string

(* ------------------------------------------------------------------ *)
(** Types *)

type type_ = [
  | TVar of name
]

type arity = type_ list
type sig_  = (arity * type_)

(* ------------------------------------------------------------------ *)
(** Expressions *)

type expr = [
  | EVar of name
  | EFun of (name * expr list)
]

(* ------------------------------------------------------------------ *)
(** Formulas *)

type logcon = [ And | Or | Imp | Equiv | Not ]
type bkind  = [ Forall | Exist ]

type form = [
  | FTrue
  | FFalse
  | FPred of (name * expr list)
  | FConn of (logcon * form list)
  | FBind of (bkind * name * type_ * form)
]

(* ------------------------------------------------------------------ *)
(** Terms = Formulas + Expressions *)

type term = [F of form | E of expr]

(* ------------------------------------------------------------------ *)
(** Environments *)

(* Body of a variable declaration, holding its type and eventually an expression
   in the case of a local definition *)
type bvar = (type_ * expr option)

type varenv = (name * bvar) list

type env = {
  env_sort      : name                      list; (* Sorts, i.e. atomic types *)
  env_prp       : (name * arity           ) list; (* Predicate symbols *)
  env_fun       : (name * sig_            ) list; (* Function symbols *)
  env_sort_name : (name * name            ) list;
  env_prp_name  : (name * name            ) list;
  env_fun_name  : (name * name            ) list;
  env_var       : varenv;                         (* Variable declarations *)
}

(* Local environment, only maps abstract variables to their type *)
type lenv = (name * type_) list
```

Figure 6.6.: ATD definitions for first-order formulas and environments

```

(* ------------------------------------------------------------------- *)
(** Goals *)

(* Unique identifier *)
type uid = string

(* Hypothesis *)
type hyp = {
  h_id : uid;
  h_form : form;
}

(* Goal *)
type goal = {
  g_env : env;
  g_hyps : hyp list;
  g_concl : form;
}

type goals = goal list

(* Abstract goal, without the signature *)
type agoal = {
  a_vars : varenv;
  a_hyps : hyp list;
  a_concl : form;
}

```

Figure 6.7.: ATD definitions for goals

If there is no saved proof for $\llbracket G \rrbracket$, then the `actema` tactic has never been executed on $\llbracket G \rrbracket$, and thus we let the User provide a (partial) proof in `Actema`. First the Plugin retrieves the list Gs of all subgoals, instead of just the one under focus. If Gs is empty, which would happen after all subgoals have been solved from `Actema`, then it sends a `QED` request to `Actema` so that the latter can update its view accordingly, and the interaction loop with `Actema` stops here[4].

4: In Figure 6.4 and Figure 6.5, we depict requests as being sent to the Frontend of `Actema`. This is an imprecision for trading in some readability, since as reviewed in Section 6.3 it is the Server which handles communication with the Plugin, and in particular forwards requests to the Frontend.

If Gs is not empty then there is at least one subgoal, and we send an `action` `HTTP` request to `Actema`, whose body contains the translated subgoals $\llbracket Gs \rrbracket$. To do this, we chose to serialize $\llbracket Gs \rrbracket$ with the `Biniou` helpers autogenerated by `atdgen`, the `OCaml` backend of `ATD`. According to its authors: “Biniou is a binary format extensible like JSON but more compact and faster to process” [162]. This data is then deserialized and compiled into a set $\mathcal{G}$ of `HTML` `DOM` nodes by the Backend, so that the goals can be rendered by the Frontend and exposed to the User. Then the User has two options:

- she can apply either a click, DnD or contextual action[5]. The precise protocol followed for applying an action is summarized in Table 6.1. Let us focus on the more complex case of DnD actions. The **Start** column describes how the User *starts* the action, here by dragging some item I of the current subgoal. Then the Frontend asks the Backend for the set $\mathcal{A}$ of all available DnD actions involving I. The **Selection** column describes how the computation of $\mathcal{A}$ is impacted by the set S of subterms that are selected in the current subgoal. For DnD actions, we essentially filter out all linkages that do not match the selection. The **Render** column describes how $\mathcal{A}$ is *rendered* to the User: here we highlight the set Q of all valid drop targets, which correspond to

5: See Section 4.2 for an introductory example of contextual action with Unfold.

Table 6.1.: Protocol for applying an action in Actema

Kind	**Start**	**Selection** S	**Render**	**End**
Click	Hover item I	Ignored	Highlight $P \subseteq I\lightning$	Click on some $p \in P$
DnD	Drag item I	If $\exists p \in S \cap I\lightning$ then match only $p @ q$ where $q \in \overline{I\lightning}$ If $\exists q \in S \cap \overline{I\lightning}$ then match only $p @ q$ where $p \in I\lightning$	Highlight $Q \subseteq \overline{I\lightning}$	Drop on some $q \in Q$
Contextual	Open menu	Populate menu only with actions applicable on S	Show menu	Choose an item in the menu

We introduced some notations for conciseness:

- p, q denote paths to subterms of the current goal
- P, Q and the selection S denote sets of paths
- $I\lightning$ denotes the set of paths within item I
- $\overline{I\lightning}$ denotes the complement of $I\lightning$, i.e. all paths in all items $J \neq I$
- $p@q$ is a linkage as introduced in Section 3.1

subterms of the current goal located in other items[6]. Lastly, the **End** column describes how the user chooses a specific action $A \in \mathscr{A}$ to apply, here by dropping I on a given target q. Then A is serialized and sent in the body of the response to the `action` request, together with the index n of the subgoal under focus in `Actema`. The Plugin can therefore compile A into a tactic $(\!|A|\!)$ which is executed on the n^{th} Coq subgoal, giving a new list of subgoals which is sent again to `Actema` for another round of the interaction loop.

- or she can click on a `Done` button in `Actema`'s interface: this has the effect of answering the `action` request from the Plugin with a `done` response, and the interaction loop with `Actema` stops here. This will happen when the User wants to go back to editing the proof script, either because she is satisfied with the new subgoals obtained from previous actions, or because she is stuck and wants to try native `Coq` tactics instead. Indeed our protocol is *synchronous*: the IDE's interface is stuck until the `actema` tactic has finished its execution, and thus one cannot edit the proof script *and* build a proof in `Actema` at the same time.

6: Highlighting is here understood in a *visual* sense: in the current implementation of `Actema`, subterms are indicated graphically by squaring them. But one could imagine other modalities for highlighting, typically *spelling* the subterms with a speech synthesis algorithm, e.g. for users with impaired vision.

6.5. Compiling actions

Once the Plugin has received the actions to execute, either from the Database or the User, it will compile them with the function $(\!| - |\!)$ which translates any action A into a `Coq` tactic $(\!|A|\!)$. This function actually has access to some other data: `Coq`'s goal G, its `Actema` translation $[\![G]\!]$, and a bijective mapping Σ between `Coq` constants in the environment of G and the corresponding `Actema` symbols in the first-order signature of $[\![G]\!]$.

```

(* ------------------------------------------------------------------- *)
(** Actions *)

(* A path refers to a subterm in the current subgoal, through a [handle]
    identifying an item of kind [kind], and a list of integers [sub] designating
    the specific subterm of the item *)
type pkind = [Hyp | Concl | Var of [Head | Body]]
type ctxt = { kind : pkind; handle : uid }
type ipath = { ctxt : ctxt; sub : int list }

(* Trace of a subformula linking, from which the list of rewrite rules to apply
   can be reconstructed *)
type choice = (int * (lenv * lenv * expr) option)
type itrace = choice list

type action = [
  | AId                                   (* The empty action which does nothing *)
  | ADef of (name * type_ * expr)         (* Introduction of a local definition *)
  | AIntro of (int * (expr * type_) option) (* Click on a conclusion *)
  | AExact of uid                         (* Proof by assumption *)
  | AElim of (uid * int)                  (* Click on a hypothesis *)
  | AInd of uid                           (* Click on a variable of inductive type *)
  | ASimpl of ipath                       (* Simplify contextual action *)
  | ARed of ipath                         (* Unfold contextual action *)
  | AIndt of ipath                        (* Induction contextual action *)
  | APbp of ipath                         (* Proof-by-Pointing contextual action *)
  | ACase of ipath                        (* Case contextual action *)
  | ACut of form                          (* Click on +hyp button *)
  | AGeneralize of uid                    (* Generalization of a hypothesis *)
  | AMove of (uid * uid option)           (* Reordering of a hypothesis *)
  | ADuplicate of uid                     (* Duplication of a hypothesis *)
  | ALink of (ipath * ipath * itrace)     (* DnD action for subformula linking *)
  | AInstantiate of (expr * ipath)        (* DnD action for instantiating a quantifier *)
]

(* An action identifier is a pair of an abstract goal and an arbitrary string identifier *)
type aident = (string * agoal)

```

Figure 6.8.: ATD definitions for actions

6.5.1. The action datatype

The `action` datatype is described thoroughly in the ATD specification provided in Figure 6.8. It is a big algebraic datatype, where each constructor encodes a specific *type* of action. An action's type is equivalent to the *signature* of a tactic, i.e. its name and the types of its arguments. In particular, the translation function $(\!|-|\!)$ is defined as a big pattern-matching on the action's type[7]. The arguments in action types rely on most datatypes defined previously in Figure 6.6 and Figure 6.7, and on two new datatypes: the type `ipath` of *paths*, which is used pervasively to designate subterms of the current subgoal (that are typically indicated by the User through pointing); and the type `itrace` of *subformula linking traces*, which is used in the compilation of DnD actions that perform subformula linking, to be described soon.

7: Note that an action's type is orthogonal to what we referred to as its *kind* in Table 6.1, that is the interface mechanism through which it is accessible. One might for example want to map some action types to *vocal commands* instead of click or DnD gestures.

Most click and contextual actions have a straightforward translation as Coq tactics. For instance, the `AIntro` action that corresponds to a click on the conclusion C will be mapped to the Coq tactic that introduces the main connective of C, and is thus defined by case on the latter: `intro` for $\supset$ and $\forall$, `split` for $\wedge$, etc. The actions `AInd`, `ASimpl` and `ARed` correspond respectively to the contextual actions Induction, Simplify and Unfold introduced in Chapter 4, and are mapped almost directly to the equivalent `Coq` tactics `induction`, `simpl` and `red`. The only difference is that they have a *deep inference* flavor, since they can all be applied on an arbitrary subterm selected by the User. This relies on our implementation of deep inference semantics directly in `Coq`, that we now briefly describe.

6.5.2. Deep inference semantics

In a deep inference setting, one can reason on subterms located arbitrarily *deep* inside statements, usually by applying some kind of rewriting rules on them. In particular, the semantics of DnD actions described in Chapter 3 are based on the rules of SFL (Figure 2.4). To implement them, we chose to do a *deep embedding* of FOL inside CoIC. Here the word "deep" has a different meaning, related to the fact that we encode the statements of FOL with our own custom datatypes, instead of reusing the statements of CoIC. This makes it easier to define the SFL rewriting rules, in particular because we need to manipulate *contexts* (Definition 3.1.1) explicitly, and those are not available for CoIC propositions.

Then we use a technique called *computational reflection* in order to apply the embedded deep inference semantics to `Coq` goals. Originating from the *small scale reflection* methodology supported by the `SSReflect` framework [100], it consists in:

1. translating `Coq` objects into their equivalent formulation in the deep embedding with a `reify` function;
2. reasoning on the deep embedding with the help of `Coq` programs (also called *fixpoints*);
3. translating objects back into `Coq` with a `reflect` function.

It is easy to implement the `reflect` function because the datatypes in a deep embedding are almost always defined as *inductive* types, and thus one can easily do pattern-matching on them. It is a different story for the `reify` function, especially in our case: indeed we want to translate the statements of `Coq` goals into first-order propositions. But `Coq` statements are objects of type `Prop`, and thus cannot be pattern-matched on inside CoIC[8]. Thus we need to have recourse to a *meta-programming* language in order to inspect the structure of `Coq` goals. Here we use the standard `Ltac` language, which provides powerful constructs for pattern-matching on goals[9].

8: For reasons of consistency of the logic, well-known in the literature on type theory.

9: A downside of `Ltac` is that it is an *untyped* language, whose programs are notoriously hard to debug and maintain. One might consider a cleaner implementation with more recent alternatives in the `Coq` ecosystem, such as the successor to `Ltac` `Ltac2` [239], or the `MetaCoq` project [228].

The most complex tactics are those implementing backward and forward DnD actions, called respectively `backward` and `forward`. They rely on two `Coq` fixpoints `b3` and `f3` which respectively compute the new conclusion `b3`(p, q, T) from a backward linkage $p \circleddash q$, and the new hypothesis `f3`(p, q, T) from a forward linkage $p \circledast q$, where T is the so-called *subformula linking trace* mentioned earlier. Of course the paths p, q and the trace T are all expressed with custom `Coq` datatypes relying on our deep embedding of FOL. The role of the trace in particular is to provide the list of SFL rewriting rules to apply, as well as the `Coq` terms instantiating quantifiers that were computed in `Actema` by unification of the two linked

```
Theorem b3_corr : forall p q T,
  coerce p -> coerce (b3 p q T) -> coerce q.

Theorem f3_corr : forall p q T,
  coerce p -> coerce q -> coerce (f3 p q T).
```

Figure 6.9.: Soundness theorems of DnD fixpoints in `Coq`

subformulas. Then we formulate in `Coq` two theorems that guarantee the logical *soundness* of `b3` and `f3` (Figure 6.9), corresponding to Theorem 3.4.1. Note that the theorems are formulated using the native implication connective `->` of `Coq`, thanks to the `reflect` function. The final tactics `backward` and `forward` can thus modify the goal by applying these theorems, first reifying the goal with the `reify` function, and then relying on the fact (also proved in `Coq`) that `reflect` is the inverse of `reify`.

Remark 6.5.1 There exist a few other approaches to the computer implementation of deep inference systems. Ozan Kahramanoğulları has pioneered the field, by implementing various calculi of structures inside frameworks like Maude [140] and Tom [138], that are dedicated to the specification of rewriting systems. For an integration within modern proof assistants, we only know of Chaudhuri's recent work [46] that explores different techniques in addition to reflection, like *combinators* and some more powerful usages of *metaprogramming*. He also provides an effective implementation of the techniques in his `Profint` tool [39], that allows to export subformula linking derivations built with `Profint`'s GUI as proof scripts directly executable in various proof assistants (`Coq`, `Lean`, `Isabelle/HOL`).

[46]: Chaudhuri et al. (2022), *Certifying Proof-By-Linking*

6.6. Future works

The `coq-actema` system described in this chapter has been successfully implemented and tested on various simple examples, including those of Section 4.1 and Section 4.3. But there are many `Coq` goals that cannot be properly handled in `Actema`, which still hinders the usability of the system in real mathematical developments, even in an educational setting. Typically, the example of Section 4.2 cannot be completely performed in `Actema`, in this case because of the lack of support for *higher-order* functions and predicates, but also because of the poor support for user-defined *notations*. Those are only a few of the current limitations of `coq-actema`, and we describe in the following pages how they could be overcome, both to widen the scope and improve the UX of the system.

6.6.1. Higher-order logic

The importance of being able to express and manipulate higher-order functions and predicates has been stressed multiple times before. The fact that `Actema` is limited to first-order logic is mostly a historical contingency, motivated by the fact that some algorithms like unification are more tractable in this setting. But now that we rely on Coq's proof engine, there is no fundamental reason for maintaining this choice. Because the language of statements is at the foundation of a logical framework, many other components of a proof assistant will depend on it. Thus the switch to higher-order logic should be done as soon as possible, to limit the amount of refactoring work to perform in the future. This will require changes to Actema's Backend and Frontend, but also to the Coq Plugin[10] and the ATD datatype definitions enabling communication between the two.

10: The Coq theory implementing the tactics for deep inference-based actions already has partial support for higher-order goals, thus work remains mostly on the side of the translation module for Actema written in OCaml.

One central question in the transition to higher-order logic is how unification of subterms will be handled. Algorithms in this setting are known to be incomplete because of undecidability [126], and their implementation can be very tricky. The most sensible option seems to rely on the implementation already provided by Coq, which is the fruit of years of development and improvements. But this would require changing the interaction protocol of coq-actema, by allowing the Backend of Actema to make unification requests to the Plugin. This might be doable without changing the current client-server architecture, but will probably involve some intricate design decisions.

[126]: Huet (1973), 'The undecidability of unification in third order logic'

A more radical solution would be to replace the `actions` request from the Frontend to the Backend by a `start` response from the Frontend to the Plugin, with the data of the selection in its body. Then we could completely delegate the computation of available actions to the Plugin, allowing us to freely use Coq's unification. This might not be too hard since we should be able to directly reuse OCaml code from the Backend, but is a deeper structural change to the interaction protocol, that makes the Plugin responsible for an important part of Actema's behavior. And this would induce a lot of unnecessary reimplementation efforts if we were to port the Plugin to other PAs.

6.6.2. Notations

Another big limitation already mentioned in the introduction of this chapter, is that we do not handle custom *notations* for displaying terms[11]. It is however a crucial feature for making proofs in a specific domain tractable, especially in the PbA paradigm where one needs to manipulate directly statements in the goal. Now that we are connected to Coq, we can in principle reuse the notation system already implemented within Coq. The coq-core OCaml API indeed exposes methods for pretty-printing Coq terms using their assigned notations. The problem is that these methods only return *strings*, but in order to manipulate terms interactively in Actema we also need access to *trees* mapping their subterm structure to the pretty-printed string. At the time of writing there is no support for the latter, but the Coq development team informed us that they plan to add this feature. The same problem was met by the developers of the ProofWidgets framework in Lean, and they had to modify the pretty-printer of Lean upstream[12].

11: Apart from expressions in Peano arithmetic, for which we have ad hoc support.

12: Section 4.1 of [12].

Once one has support for custom notations displayed in an HTML page, it is tempting to also allow for arbitrary `HTML/JS` code, instead of just textual notations. This opens the space for very rich graphical and interactive representations of mathematical objects, which could greatly improve the accessibility of PAs, but also their expert usage by enabling domain-specific interfaces targetting non-standard methods of reasoning. A typical example is the *diagrammatic* reasoning pervasive in *category theory*, which is very hard and cumbersome to express as manipulation of logical statements. Actually a system very similar to `coq-actema` is currently being developed by Luc Chabassier [36], for the very purpose of integrating diagrammatic proofs in category theory to the traditional proof script workflow. One could imagine in the long-term embedding this system as a subsystem of `coq-actema`, through an advanced protocol for interactive notations.

In fact the `ProofWidgets` framework has been designed with this usecase in mind from the outset. But they rely on a very different architecture compared to that of `coq-actema`, where the methods generating the `HTML/JS` code of pretty-printed terms are directly implemented in a DSL embedded in the meta-programming language of the PA. While this allows easy access to all meta-programming facilities for manipulating terms, this makes their framework only usable within `Lean`, while `Actema` could in principle be used with any PA that implements a corresponding plugin (for example with a `lean-actema` variant of our system).

6.6.3. Lemma search

We already described in Section 4.2 the *lemma search* feature of `Actema`. Currently it is implemented only in its standalone version. Adding support for it in `coq-actema` would require additional efforts compared to other contextual actions like `Induction`. Indeed we do not only need access to the current goal, but also to the full global environment of `Coq` where lemmas are stored. While in the standalone version we had a toy lemma database with very few entries, the standard library of `Coq` contains thousands of lemmas. And to use our selection-based filtering algorithm implemented in `Actema`, we would need to translate the entire library into statements understandable by `Actema`'s Backend, and then send it over `HTTP`. Thus it will be important to implement some cache mechanism to remember which lemmas have already been exported to `Actema`'s own database, to avoid recomputing the translation each time.

6.6.4. Proof evolution

An important question when designing a proving environment is how users will be able to manipulate an *existing* (partial) proof, either one they have built in the past, one that was built by other people, or a mix of both in a collaborative context. This is a complex problem spanning various activities that are involved in the lifecycle of a proof: modifying it while it is being constructed; reading it for the first time, or many months/years after it was written; updating it after slight changes to the statement of its theorem; etc[13].

13: Not very surprisingly, those activities are commonly found in the context of *programming* environments. Thus one might get insight by cross-pollinating ideas from both domains, in the spirit of the Curry-Howard correspondence (which also underlies the design of `Coq`).

Digression

One might even argue that thinking about the best way to *represent* a proof leads to more fundamental questions, that have been much debated both in proof theory and the history and philosophy of mathematics: what is the essence of a proof, seen as a meta-mathematical object [233]? What are the roles played by informal and formal proofs, both in the teaching of mathematics, and the social and scientific practice of mathematicians [15]?

In the literature and community of people designing proof assistants, these various problematics are generally regrouped under the term of *proof evolution*. A fundamental remark about the PbA paradigm, and thus about the `coq-actema` system, is that it has not been designed with proof evolution in mind from the outset. Indeed, a proof built with the `actema` tactic will provide the least possible amount of information in the proof script, since we can just witness the call to that tactic. And currently there are no facilities to visualize the associated sequence of actions stored in the Database of graphical proofs.

The first question that should be answered is: how do we represent statically a sequence of graphical actions, let alone a single action? For a machine representation, we can just dump the data of the action invokation, and this is indeed what we do with the Database. But finding a human-readable representation that an average user can quickly manipulate and reason about is a lot more delicate. The most direct way may be to abandon text altogether, and just replay the action on the interface through a graphical *animation*. This is an intrisically temporal and dynamic representation, akin to a mathematician unfolding her demonstration on the blackboard. One could then imagine an interface dedicated to richly-structured navigation inside this sequence of animations, in the style of an improved video player.

A more conservative solution would be to find a systematic way to translate a sequence of actions into a proof text. The question of generating declarative proof texts from imperative proof scripts has already been explored by some authors, especially in the case of proofs expressed in natural language[14]. Our hope is that the structure of proofs in the PbA paradigm might be well-suited to the generation of readable and concise proof texts, thanks notably to the subformula linking semantics of DnD actions that exhibit clearly the flow of argumentation.

14: See for example section 3.6 of E. Ayers' thesis [11]. We can also mention ongoing work of Patrick Massot in the `Lean` proof assistant [166].

An even more pragmatic solution, that should be straightforward to implement in the short-term, consists in inserting tactic invokations in the proof script that are in one-to-one correspondence with graphical actions. Since we actually compile actions into tactics, this is in principle easy to implement. However, there are currently two drawbacks to this approach:

Leaking SFL data Since most tactics are deep inference-based, they take as arguments the paths to manipulated subterms, in the form of lists of integers. Those are hard to read by humans, and very sensible to small changes in the shape of the goal. This is even worse for the `backward` and `forward` tactics, because they also take as argument the subformula linking trace, which is a very complex data structure expressed in our deep embedding of FOL, and hence should not leak into the user interface. Hopefully, relying on `Coq`'s unification instead of `Actema`'s should mitigate the complexity of the trace, by removing the need to incorporate full substitutions. There is also the possibility of replacing integer-based paths by *patterns* in the `SSReflect` language, which are known to be a more robust way to designate subterms. But this would require the design of some clever algorithm, able to generate patterns that correctly generalize the User's intent from the sole data of selected paths.

Editor integration The interaction protocol described in Section 6.4 does not provide any way to send requests to the IDE, which would be necessary to actually insert the tactic invokation at the right location in the proof script, and this as soon as the action is performed by the User. A "brutal" solution would be to reimplement `coq-actema` as an extension of a specific IDE, typically `VsCoq` which is also based on web technologies. But this would require some big implementation efforts, in addition to locking the User into this specific IDE. A better option might be to directly interact with a *language server* implementing the Language Server Protocol (`LSP`) [51]. The `coq-lsp` project aims to provide such a server for `Coq`, but at the time of writing of this thesis does not implement yet all methods of the `LSP` standard. The one that interests us in particular is the `textDocument/codeAction` method, for which support is currently planned [86]. Then `coq-actema` would stay compatible with all IDEs that run `coq-lsp`.

Iconic Manipulations

Asymmetric Bubble Calculus 7.

> Leibniz sought to make the form of a symbol reflect its content. “In signs,” he wrote, “one sees an advantage for discovery that is greatest when they express the exact nature of a thing briefly and, as it were, picture it; then, indeed, the labor of thought is wonderfully diminished.”
>
> **Frederick Kreiling**, *Leibniz*, 1968

7.1 The chemical metaphor 103
7.2 Bubbles and solutions 104
7.3 Asymmetric calculus 106
7.3.1 Interpreting solutions . . 106
7.3.2 Sequent-style rules 106
7.3.3 Proof-as-trace 109
7.3.4 Graphical rules 109
7.4 Back to Proof-by-Action 113

We introduce a new kind of nested sequent proof system dubbed bubble calculus. Inspired by the *membrane* mechanism of the *chemical abstract machine* (Cham) [17], so-called *bubbles* internalize the notion of subgoal inside sequents, rather than through the tree structure induced by traditional inference rules. This allows for a more hierarchical representation of the proof state, where contexts can be shared between different subgoals. In addition to the usual textual syntax, the bubble calculus can be expressed in a graphical syntax, where logical meaning is captured by *physical* constraints on diagrammatic manipulations, instead of *virtual* restrictions on available inference rules.

[17]: Berry et al. (1989), ‘The chemical abstract machine’

We start in Section 7.1 with the genesis of the idea of bubble calculus, coming from the observation that our Proof-by-Action paradigm (Chapter 2) lends itself quite naturally to a metaphorical interpretation, where actions are seen as *chemical* reactions. In Section 7.2 we introduce the concept of *bubble* as a way to control the scope of hypotheses inside nested sequents that we call *solutions*. In Section 7.3 we describe our proof system for intuitionistic logic dubbed *asymmetric bubble calculus*, based on multiset rewriting rules over solutions comprising at most one conclusion. Finally in Section 7.4, we import ideas from this bubble calculus back to the realm of GUIs for interactive proof building, analyzing their possible impact for UX improvements.

7.1. The chemical metaphor

The Proof-by-Action paradigm introduced in Chapter 2 offers multiple ways to the user to attack the proof of a theorem. DnD actions for subformula linking and equality rewriting are the main mechanism, but they only work in a goal comprising multiple items. Since it is customary in proof assistants to specify the goal to be proved as a single logical formula, one needs a way to decompose it into many items for further processing through DnD. This is precisely what the introduction rules for logical connectives in sequent calculus do, and following the Proof-by-Pointing paradigm [18] we map them to click actions (see Section 2.3).

So visually, a proof in `Actema` consists in breaking logical items into subitems positioned freely in space, and then bringing those subitems together to make them interact and produce a new item. This is quite evocative of a *chemical reaction* controlled by the user, where logical formulas are akin to molecules made of propositional atoms linked together by logical connectives[1]. Click actions are then a mean to "heat" molecules to the point of breaking these chemical bonds. The most canonical examples are the right introduction rule for implication $\supset$R, which breaks a conclusion/positive ion into a new hypothesis/negative ion and a new conclusion; and the left introduction rule for conjunction $\wedge$L, which breaks a hypothesis into two hypotheses. In fact, we strongly conjecture that these are the only click actions needed to obtain a complete deductive system for propositional logic: breaking red implications allows for backward DnDs, and blue conjunctions for forward DnDs[2].

1: This precise metaphor about the molecular structure of propositions can already be found in Russell's introduction to Wittgenstein's *Tractatus Logico-Philosophicus* [255, p. 11], which was the main inspiration to his philosophy of *logical atomism* [142]. Even earlier in the history of logic, C. S. Peirce took inspiration from chemical diagrams to devise his *existential graphs* — see [214, pp. 17–18] or our own presentation in Section 9.7 for more details.

2: In predicate logic, one would also need the right (resp. left) introduction rule for $\forall$ (resp. $\exists$). It might also be the case that backward DnDs alone are sufficient for completeness, since a linkage of the form $A \circleddash \boxed{B} \supset C$ will involve a forward phase. In this case only the right introduction rules for $\supset$ and $\forall$ would be required.

Rather than completeness, the issue here is *consistency* of the user interface: if the user is allowed to decompose red $\supset$ and blue $\wedge$, she will assume naturally that she can also decompose blue $\supset$ and red $\wedge$, as well as $\vee$ of any color. While red $\vee$ can be handled by pointing directly at the disjunct to be proved, other configurations correspond to rules of sequent calculus with multiple premisses. In `Actema`, this corresponds to creating a new subgoal for each premise, where subgoals are displayed one at a time in different *tabs*: this new interface mechanism breaks the chemical metaphor. The root cause lies in the way sequent calculus implements *context-scoping*: each subgoal will share the same initial context of hypotheses, but future hypotheses "buried" in the conclusions must be available only in their respective subgoals. The tabs mechanism implements this by forcing the user to focus on exactly one tab/subgoal, thus making it impossible to display items from different subgoals on the same screen: this renders interaction between items *physically* impossible.

7.2. Bubbles and solutions

In order to accomodate context-scoping within the chemical metaphor, we were led to explore a notion of *bubble* inspired by the *membranes* of the Cham [17]. The latter are used to delineate zones of *local* interaction, which are still porous to external data. This is precisely what we want to do here: let us consider that the user tries to prove the sequent $\Gamma \Rightarrow A \wedge B$. By clicking on the red item $A \wedge B$, she will break it into two bubbles $(\!| \Rightarrow A |\!)$ and $(\!| \Rightarrow B |\!)$. Then she might decompose A and B further into sequents $\sigma_A = \Gamma_A \Rightarrow C_A$ and $\sigma_B = \Gamma_B \Rightarrow C_B$, and use hypotheses from Γ by dragging them inside either $(\!| \sigma_A |\!)$ or $(\!| \sigma_B |\!)$. However, hypotheses from Γ_A and Γ_B cannot be dragged out from their respective bubble, since then they could be used in the other bubble and violate context-scoping.

[17]: Berry et al. (1989), 'The chemical abstract machine'

This situation is illustrated in Figure 7.1, where bubbles are represented by gray circles, and possible drag moves of formulas by arrows. More specifically, green and orange arrows symbolize respectively valid and invalid moves. Notice how this graphical depiction of bubbles exhibits their *topological* behavior: while objects can enter inside bubbles from the outside, they get blocked by the membrane in the opposite direction. Indeed the only relevant feature of the circle representation is that it divides the space into an *interior* and an *exterior*. Then the *nesting* of circles and the *positions* of formulas relative to them encode respectively

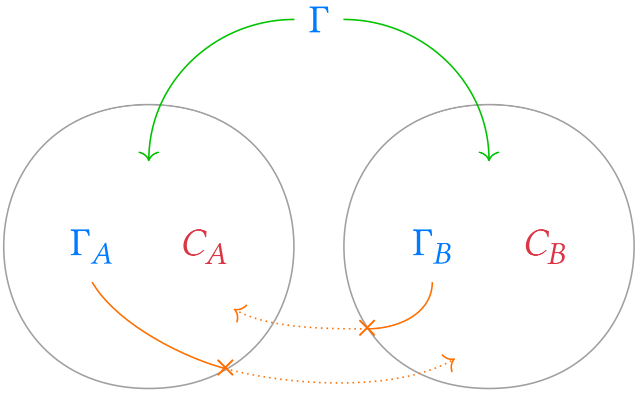


Figure 7.1.: Context-scoping in bubbles as topological constraints

the *tree* structure of the proof, and the scope of hypotheses in it.

Bubbles can also be seen as a way to internalize in the syntax of sequents the notion of *subgoal*, which requires in turn to allow nesting of sequents inside each other. The proof state is not a set of subgoals anymore, but a single nested sequent of this sort, that we call a *solution*[3]. In textual syntax, solutions S are generated by the following grammar:

$$S, T, U ::= \Gamma \langle S_1 ; \dots ; S_n \rangle \Delta \qquad\qquad \Gamma, \Delta ::= A_1, \dots, A_n$$

where the left-hand Γ and right-hand Δ in solutions represent respectively *hypotheses* and *conclusions*, and the A_i are usual formulas of FOL. Thus solutions are just like sequents, except that we add a collection of nested solutions S_i that will represent subgoals, or premisses of usual inference rules. To be more precise, the collections of formulas A_i and solutions S_i are *multisets*, which gives the following mutually recursive definitions:

Definition 7.2.1 (Ion) *An ion is a formula charged either negatively (hypothesis) or positively (conclusion).*

Definition 7.2.2 (Bubble) *A bubble is a solution enclosed in a membrane.*

Definition 7.2.3 (Solution) *A solution S is a multiset of ions and bubbles. It is single-conclusion if it contains at most one positive ion. We will use letters $\mathcal{S}, \mathcal{T}, \mathcal{U}$ to denote multisets of solutions.*

Note that in the above definitions, bubbles play a purely metaphorical role and could be dispensed with. But it will be useful later on to distinguish them conceptually from solutions.

3: The term "solution" refers here to the metaphor of a *chemical solution* made up of an unordered collection of molecules. Which is quite ironic, since we use it to denote goals waiting to be proved, that is problems lacking a solution...

7.3. Asymmetric calculus

7.3.1. Interpreting solutions

A natural way to give logical meaning to a solution is to translate it into a formula. In the following we provide one such translation, which will play a determining role in the design of inference rules for manipulating solutions. We qualify it of *asymmetric* because it only works for single-conclusion solutions, in the same way that LJ only works for single-conclusion sequents.

Remark 7.3.1 In this section we only deal with single-conclusion solutions, but the more general case will be studied in Chapter 8.

Just like a sequent, a solution is semantically equivalent to an implication, except that we add the *conjunction* of all subgoals to the consequent:

Definition 7.3.1 (Asymmetric interpretation) *The* asymmetric interpretation $\llbracket - \rrbracket$ *on solutions is defined recursively by:*

$$\llbracket \Gamma \, \langle S_1 \, ; \, \dots \, ; \, S_n \rangle \, \Delta \rrbracket = \bigwedge \Gamma \supset \left(\bigwedge \Delta \wedge \bigwedge_i \llbracket S_i \rrbracket \right)$$

Note that we join formulas in Δ conjunctively: since we do not consider solutions with more than one conclusion, this is just to handle the case where $\Delta = \emptyset$, and thus $\bigwedge \Delta = \top$. This subtle detail is in fact essential to the way we encode the tree structure of proofs inside solutions:

- a solution with one conclusion corresponds to a *leaf* of the proof tree, i.e. a subgoal;
- a solution with no conclusion corresponds to a *node* of the proof tree, i.e. a branching point where we created multiple subgoals.

This will soon become clearer with examples of derivations in our calculus. In Section 8.2, we will consider a different interpretation of solutions that entails a different encoding of the proof structure in them.

7.3.2. Sequent-style rules

Our initial idea for a proof system based on solutions was quite simple: we take the inference rules of LJ, and turn them each into an unary rule by encoding premisses as bubbles. This gives the basis for the set of rules presented in Figure 7.2, which defines our asymmetric bubble calculus for intuitionistic logic dubbed BJ. It is divided in five groups:

- The IDENTITY, RESOURCE and HEATING groups correspond respectively to the identity, structural and introduction rules of sequent calculus, following the terminology of [94]. More precisely, rules i↓

[94]: Girard (2011), *The blind spot*

Identity

$$\frac{\Gamma\ \langle\mathcal{S}\rangle}{\Gamma, A\ \langle\mathcal{S}\rangle\ A}\ \mathsf{i}{\downarrow} \qquad \frac{\Gamma\ \langle\mathcal{S};\ \langle\rangle\ A;\ A\ \langle\rangle\ \Delta\rangle}{\Gamma\ \langle\mathcal{S}\rangle\ \Delta}\ \mathsf{i}{\uparrow}$$

Resource

$$\frac{\Gamma\ \langle\mathcal{S}\rangle\ \Delta}{\Gamma, A\ \langle\mathcal{S}\rangle\ \Delta}\ \mathsf{w} \qquad \frac{\Gamma, A, A\ \langle\mathcal{S}\rangle\ \Delta}{\Gamma, A\ \langle\mathcal{S}\rangle\ \Delta}\ \mathsf{c}$$

Flow

$$\frac{\Gamma\ \langle\mathcal{S}; \Gamma', A\ \langle\mathcal{S}'\rangle\ \Delta'\rangle\ \Delta}{\Gamma, A\ \langle\mathcal{S}; \Gamma'\ \langle\mathcal{S}'\rangle\ \Delta'\rangle\ \Delta}\ \mathsf{f}{-}$$

Membrane

$$\frac{\Gamma\ \langle\mathcal{S}\rangle\ \Delta}{\Gamma\ \langle\mathcal{S}; \langle\rangle\ \rangle\ \Delta}\ \mathsf{p}$$

Heating

$$\frac{\Gamma\ \langle\mathcal{S}\rangle\ \Delta}{\Gamma, \top\ \langle\mathcal{S}\rangle\ \Delta}\ \top{-} \qquad \frac{\Gamma\ \langle\mathcal{S}\rangle}{\Gamma\ \langle\mathcal{S}\rangle\ \top}\ \top{+}$$

$$\frac{\Gamma\ \langle\mathcal{S}\rangle}{\Gamma, \bot\ \langle\mathcal{S}\rangle\ \Delta}\ \bot{-}$$

$$\frac{\Gamma, A, B\ \langle\mathcal{S}\rangle\ \Delta}{\Gamma, A \wedge B\ \langle\mathcal{S}\rangle\ \Delta}\ \wedge{-} \qquad \frac{\Gamma\ \langle\mathcal{S};\ \langle\rangle\ A;\ \langle\rangle\ B\rangle}{\Gamma\ \langle\mathcal{S}\rangle\ A \wedge B}\ \wedge{+}$$

$$\frac{\Gamma\ \langle\mathcal{S}\rangle\ A}{\Gamma\ \langle\mathcal{S}\rangle\ A \vee B}\ \vee{+_1}$$

$$\frac{\Gamma\ \langle\mathcal{S}; A\ \langle\rangle\ \Delta; B\ \langle\rangle\ \Delta\rangle}{\Gamma, A \vee B\ \langle\mathcal{S}\rangle\ \Delta}\ \vee{-}$$

$$\frac{\Gamma\ \langle\mathcal{S}\rangle\ B}{\Gamma\ \langle\mathcal{S}\rangle\ A \vee B}\ \vee{+_2}$$

$$\frac{\Gamma\ \langle\mathcal{S};\ \langle\rangle\ A; B\ \langle\rangle\ \Delta\rangle}{\Gamma, A \supset B\ \langle\mathcal{S}\rangle\ \Delta}\ \supset{-} \qquad \frac{\Gamma, A\ \langle\mathcal{S}\rangle\ B}{\Gamma\ \langle\mathcal{S}\rangle\ A \supset B}\ \supset{+}$$

$$\frac{\Gamma, A\{t/x\}\ \langle\mathcal{S}\rangle\ \Delta}{\Gamma, \forall x.A\ \langle\mathcal{S}\rangle\ \Delta}\ \forall{-} \qquad \frac{\Gamma\ \langle\mathcal{S}\rangle\ A}{\Gamma\ \langle\mathcal{S}\rangle\ \forall x.A}\ \forall{+}$$

$$\frac{\Gamma, A\ \langle\mathcal{S}\rangle\ \Delta}{\Gamma, \exists x.A\ \langle\mathcal{S}\rangle\ \Delta}\ \exists{-} \qquad \frac{\Gamma\ \langle\mathcal{S}\rangle\ A\{t/x\}}{\Gamma\ \langle\mathcal{S}\rangle\ \exists x.A}\ \exists{+}$$

In the $\forall{+}$ and $\exists{-}$ rules, x is not free in Γ, Δ and $\mathcal{S}$.

Figure 7.2.: Sequent-style presentation of the asymmetric bubble calculus BJ

and i↑ correspond to the axiom and cut rules; rules w and c to the weakening and contraction rules; and every rule of the form ∘− (resp. ∘+) to the left introduction rule (resp. right introduction rule) for the logical connective ∘.

- The FLOW and MEMBRANE groups are new, and define the behavior of bubbles. More specifically, F-rules characterize how information flows inside solutions by specifying what kinds of objects can traverse bubbles, and in which direction. They play the same role as *switch* rules in formalisms based on CoS [110], which includes our own subformula linking rules (Figure 2.4). In the asymmetric bubble calculus there is only one F-rule f− allowing hypotheses to flow inside bubbles.

  As their name suggests, M-rules handle the behavior of the *membrane* of bubbles, but independently from other items as opposed to F-rules. In the asymmetric bubble calculus there is only one M-rule p allowing to *pop* any empty bubble, which can be interpreted as the action of dismissing a solved subgoal. In CoS it would correspond to congruence rules handling the truth unit ⊤, and in subformula linking to the unit elimination rules (Figure 2.5).

Now that we have rules for manipulating solutions, and since solutions can be nested through bubbles, we need a notion of *context* for applying rules on subsolutions of arbitrary depth:

Definition 7.3.2 (Context) *A* context $S\square$ *is a solution which contains exactly one occurrence of a special solution written* $\square$*, called its* hole*. Given another solution* T*, we write* $S\boxed{T}$ *to denote the solution equal to* $S\square$ *where* $\square$ *has been replaced by* T*.*

Then every rule of Figure 7.2 is applicable in any context $U\square$. That is:

$$\frac{S}{T} \quad \text{should be read as} \quad \frac{U\boxed{S}}{U\boxed{T}} \quad \text{for all } U\square$$

Definition 7.3.3 (Derivation) *We write* $S \to T$ *to indicate a* rewrite step*, that is an instance of some rule from Figure 7.2 with* T *as premiss and* S *as conclusion*[4]*. A* derivation $S \to^n T$ *is a sequence of rewrite steps* $S_0 \to S_1 \ldots \to S_n$ *with* $S_0 = S$*,* $S_n = T$ *and* $n \geq 0$*. Generally the length* n *of the derivation does not matter, and we just write* $S \to^* T$*. Finally, derivations are closed under arbitrary contexts: for every context* $U\square$*,* $S \to T$ *implies* $U\boxed{S} \to U\boxed{T}$*. We write* $S \rightharpoonup T$ *to denote a* shallow step*, i.e. a direct instance of a rule in the empty context.*

4: The direction of the arrow is from conclusion to premiss, to stay consistent with our interactive proof building setting where inference rules are seen as goal-modifying actions.

Definition 7.3.4 (Proof) *A* proof *of a solution* S *in* BJ *is a derivation* $S \to^* \langle\rangle$ *that reduces* S *to the empty solution, which denotes the proof state where there are no subgoals left.*

$$\begin{array}{cl}
\langle\rangle & \\ \hline
\boxed{\langle\,\langle\rangle\,\rangle} & \mathsf{p} \\ \hline
\langle\boxed{\langle\,\langle\rangle\,\rangle}\rangle & \mathsf{p} \\ \hline
\langle\,\langle\boxed{B\,\langle\rangle\,B}\rangle\,\rangle & \mathsf{i}\!\downarrow \\ \hline
\langle\boxed{B\,\langle\,\langle\rangle\,B\rangle}\rangle & \mathsf{f}- \\ \hline
\langle B\,\langle\boxed{\langle\rangle\,A \vee B}\rangle\,\rangle & \vee+_2 \\ \hline
\langle\boxed{B\,\langle\,\langle\rangle\,A \vee B;\,\langle\rangle\,\rangle}\rangle & \mathsf{p} \\ \hline
\langle B\,\langle\,\langle\rangle\,A \vee B;\,\boxed{C\,\langle\rangle\,C}\rangle\,\rangle & \mathsf{i}\!\downarrow \\ \hline
\langle\boxed{B, A \vee B \supset C\,\langle\rangle\,C}\rangle & \supset- \\ \hline
\boxed{\langle\,\langle\rangle;\,B, A \vee B \supset C\,\langle\rangle\,C\rangle} & \mathsf{p} \\ \hline
\langle\boxed{\langle\,\langle\rangle\,\rangle};\,B, A \vee B \supset C\,\langle\rangle\,C\rangle & \mathsf{p} \\ \hline
\langle\,\langle\boxed{A\,\langle\rangle\,A}\rangle;\,B, A \vee B \supset C\,\langle\rangle\,C\rangle & \mathsf{i}\!\downarrow \\ \hline
\langle\boxed{A\,\langle\,\langle\rangle\,A\rangle};\,B, A \vee B \supset C\,\langle\rangle\,C\rangle & \mathsf{f}- \\ \hline
\langle A\,\langle\boxed{\langle\rangle\,A \vee B}\rangle;\,B, A \vee B \supset C\,\langle\rangle\,C\rangle & \vee+_1 \\ \hline
\langle\boxed{A\,\langle\,\langle\rangle\,A \vee B;\,\langle\rangle\,\rangle};\,B, A \vee B \supset C\,\langle\rangle\,C\rangle & \mathsf{p} \\ \hline
\langle A\,\langle\,\langle\rangle\,A \vee B;\,\boxed{C\,\langle\rangle\,C}\rangle;\,B, A \vee B \supset C\,\langle\rangle\,C\rangle & \mathsf{i}\!\downarrow \\ \hline
\langle\boxed{A, A \vee B \supset C\,\langle\rangle\,C};\,B, A \vee B \supset C\,\langle\rangle\,C\rangle & \supset- \\ \hline\hline
\boxed{A \vee B \supset C\,\langle A, A \vee B \supset C\,\langle\rangle\,C;\,B\,\langle\rangle\,C\rangle} & \mathsf{f}- \\ \hline\hline
\boxed{A \vee B \supset C, A \vee B \supset C\,\langle A\,\langle\rangle\,C;\,B\,\langle\rangle\,C\rangle} & \mathsf{f}- \\ \hline
\boxed{A \vee B \supset C\,\langle A\,\langle\rangle\,C;\,B\,\langle\rangle\,C\rangle} & \mathsf{c} \\ \hline
A \vee B \supset C\,\langle A\,\langle\rangle\,C;\,\boxed{\langle\rangle\,B \supset C}\rangle & \supset+ \\ \hline
A \vee B \supset C\,\langle\boxed{\langle\rangle\,A \supset C};\,\langle\rangle\,B \supset C\rangle & \supset+ \\ \hline
\boxed{A \vee B \supset C\,\langle\rangle\,(A \supset C) \wedge (B \supset C)} & \wedge+ \\ \hline
\boxed{\langle\rangle\,(A \vee B \supset C) \supset (A \supset C) \wedge (B \supset C)} & \supset+
\end{array}$$

Figure 7.3.: Example of sequent-style proof in BJ

7.3.3. Proof-as-trace

An example of proof in BJ is shown in Figure 7.3, where the focused subsolution is squared for each inference. Notice that many rules could have been applied in a different order: for instance all applications of the p rule could have been postponed to the top/end of the derivation. This is generally true of all formalisms based on CoS, which is known in the deep inference literature for its "bureaucracy". In BJ, ℍ-rules aggravate the matter by adding all inessential rule permutations from sequent calculus to those of CoS. As our wording suggests, this is usually perceived negatively in deep inference proof theory, where a central question is that of finding *canonical* representations of proof objects [233].

[233]: Straßburger (2019), 'The problem of proof identity, and why computer scientists should care about Hilbert's 24th problem'

However in our interactive proof building setting, it should rather be seen as a *desirable* property of the system. Indeed, one consequence is that the user has more freedom to organize her reasoning in whichever order she wants, in an incremental and guided way. One should remember that in the Proof-by-Action paradigm, the focus is not the proof object, which is implicit and hidden to the user, but the *process* of building it. Then a BJ-derivation is better understood as the *trace* of this building process, rather than the constructed proof[5]. And the fact that this trace corresponds, or can be transformed into a more canonical representation is of no concern to the user. What matters for a good proof building interface is to be as flexible as possible, in order to match the user's own mental process of argumentation.

5: The idea of *proof-as-trace* is relatively common in logic programming [174], but not so much in deep inference proof theory. It is Jean-Baptiste Joinet who shared with us his idea of applying it in this setting, based on his own work interpreting the CoS for MLL as a system for building *multiplicative proof nets* [135].

Of course flexibility comes at a price, and the rules of BJ are probably too numerous and low-level to be mapped directly into individual proof actions in a user interface. Some of these concerns will be tackled in Subsection 8.8.2, but we think a better answer might have been found with the proof system introduced in Chapter 10, and its associated prototype of GUI presented in Section 10.8.

7.3.4. Graphical rules

While the sequent-style presentation of BJ clearly shows its filiation with sequent calculus, its syntax is quite heavy, and obscures an important property of the rules: they almost always preserve the contexts Γ, Δ of formulas and $\mathcal{S}$ of bubbles. That is, the rules of BJ are *local*. This enables a more economical and graphical presentation of the rules in Figure 7.4, where BJ is seen as a multiset rewriting system just like the Cham thanks to Definition 7.2.3. Instead of relying on a notion of context, we define formally what it means to be a *subsolution*:

Definition 7.3.5 (Subsolution) *S is a subsolution of T, written $S \prec T$, if either $S \subseteq T$ or $S \prec T_0$ for some $T_0 \in T$, where $\subseteq$ denotes multiset inclusion.*

Then a multiset rewriting rule $S \to_r T$ can be applied in a solution U whenever $S \prec U$, by replacing one occurrence of S by T inside U. The notions of derivation (Definition 7.3.3) and proof (Definition 7.3.4) stay unchanged, by observing that the rewriting rule $S \to_r T$ from S to T and the inference rule $r : S \to T$ with premiss T and conclusion S denote the same rule r.

Figure 7.5 shows the graphical presentation of the same BJ-proof as in Figure 7.3. Whereas in Figure 7.3 we squared the whole subsolutions corresponding to the conclusions of inference rules, here we squared on each line the redex modified by the associated rewriting rule. This example highlights the greater locality of the rewriting approach, by indicating more precisely which parts of the proof state are changed by the rules.

But it still over-approximates the modifications that really need to be performed to carry the transformations. Indeed, by only exposing the data of a redex S and a reddendum T, a rewriting rule $S \to_r T$ can only be interpreted as the deletion of S followed by the insertion T. Taking for instance the $\supset-$ rule in Figure 7.4, one can describe its graphical behavior more finely as resulting from the following sequence of *edits*:

1. Erase the $\supset$ connective;
2. Change the polarity of A from hypothesis to conclusion;
3. Insert a new empty bubble;
4. Move A in this bubble;
5. Insert a new empty bubble;
6. Move B in this bubble;
7. If Δ is not empty, also move Δ in this bubble.

It would be interesting to consider the question of finding a minimal set of edit operations like these, that can simulate all the rules of BJ[6]. Note however that most of the above edits are *unsound* as reasoning steps. If not for logical insight, such an edit calculus could still be relevant *computationally*, typically by enabling efficient implementations of the rules with a small memory footprint.

6: As will become apparent in Section 8.7, BJ itself provides a finer-grained simulation of the rules of sequent calculus, which in turn is known to be a more detailed variant of *natural deduction*. Interestingly through the Curry-Howard isomorphism, this would correspond to a *chain of compilation*, starting from the higher-level λ-calculus (natural deduction), going into abstract machines (sequent calculus) [71], down to something akin to *assembly language* with `jump` instructions [108, Section 6.3.1].

Identity

$A\ A$	$\to$		i↓
Δ	$\to$	A $\quad A\ \Delta$	i↑

Resource

A	$\to$		w
A	$\to$	$A\ A$	c

Flow

$A\ S$	$\to$	$A\ S$	f−

Membrane

	$\to$		p

Heating

$\top$	$\to$		$\top-$	$\top$	$\to$		$\top+$
$\bot\ \Delta$	$\to$		$\bot-$				
$A \wedge B$	$\to$	$A\ B$	$\wedge-$	$A \wedge B$	$\to$	A $\quad B$	$\wedge+$
$A \vee B\ \ \Delta$	$\to$	$A\ \Delta$ $\quad B\ \Delta$	$\vee-$	$A \vee B$	$\to$	A	$\vee+_1$
				$A \vee B$	$\to$	B	$\vee+_2$
$A \supset B\ \ \Delta$	$\to$	A $\quad B\ \Delta$	$\supset-$	$A \supset B$	$\to$	$A\ B$	$\supset+$
$\forall x.A$	$\to$	$A\{t/x\}$	$\forall-$	$\forall x.A$	$\to$	$A\{y/x\}$	$\forall+$
$\exists x.A$	$\to$	$A\{y/x\}$	$\exists-$	$\exists x.A$	$\to$	$A\{t/x\}$	$\exists+$

In the rules i↑, $\bot-$, $\vee-$ and $\supset-$, Δ is either empty, or a singleton of one positive ion.
In the rules $\forall+$ and $\exists-$, y is fresh.

Figure 7.4.: Graphical presentation of the asymmetric bubble calculus BJ

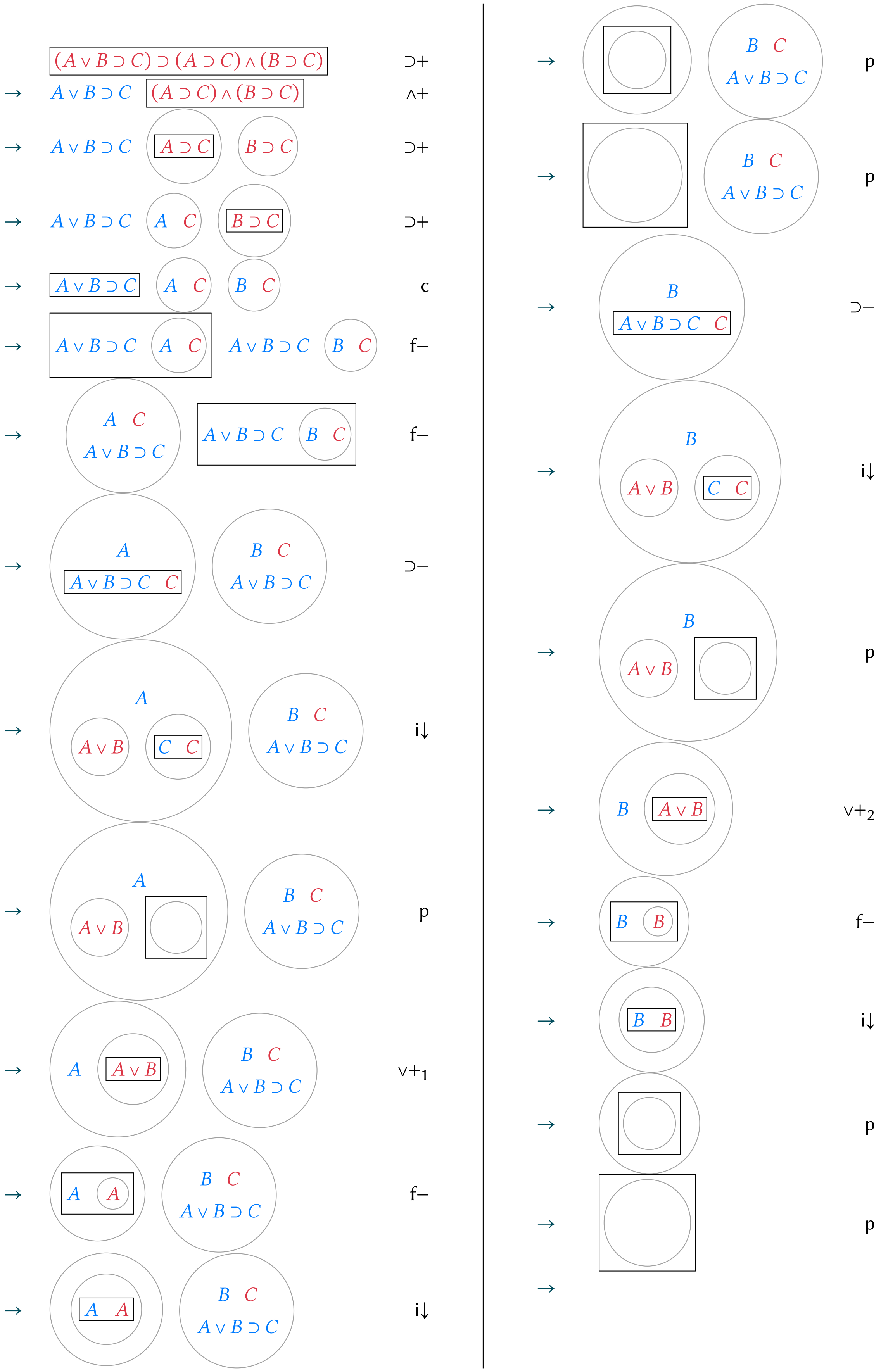


Figure 7.5.: Example of graphical proof in BJ

7.4. Back to Proof-by-Action

When looking at the BJ-proof of Figure 7.5, the astute reader might have been reminded of the Proof-by-Action paradigm as introduced in Chapter 2, by seeing redexes as the items involved in a graphical action — there are always at most two such items. More precisely, ℍ-rules correspond to *click* actions on blue (∘− rules) or red items (∘+ rules), and the i↓ rule corresponds to the most basic DnD action between dual occurrences of a formula.

As mentioned earlier when comparing BJ to LJ, the novelty here lies with ℍ-rules, 𝔽-rules and 𝕄-rules that deal with *bubbles*. Remember that the goal behind the idea of bubble calculus was precisely to provide a new way to manipulate subgoals through bubbles instead of tabs, which are more in line with the chemical metaphor. It is quite easy to imagine a GUI presenting the proof state as a solution, in a graphical layout close to that of Figure 7.5[7]. Like formulas in blue and red items, whole subgoals could now be shown on the same screen in their respective bubbles, and be freely moved around with a pointing device. Following are some ideas for mapping the remaining rules of BJ in such a GUI:

7: Although there might be some challenges in implementing an efficient layouting algorithm for bubbles, typically to make solutions fit into the screen.

𝔽**LOW** The f− rule plays a special role, in that it would not be mapped to any particular action. Indeed it captures the way information flows in solutions, and we already described in Section 7.2 how this is reflected in the topological behavior of bubbles. Thus it could be implemented in the graphical interface as a kind of *physics engine*, like those found in video games: when dragging an item around the proof canvas, it would get stuck on the membrane of bubbles, except when the item is blue and the drag movement goes inward. This of course would provide a level of interactivity unseen before in a proving interface, making it very discoverable and playful. It also combines nicely with DnD actions in general: for instance a sequence of applications of f− followed by i↓ could be performed as a single DnD action, where the dragged hypothesis crosses successively the various bubbles on the way.

𝕄**EMBRANE** The p rule can be mapped very straightforwardly to the action of clicking on the area of an empty bubble, in order to pop it. It could also be entirely automated, by letting the proof engine eagerly pop empty bubbles as soon as they appear in a solution. Note that in this graphical setting, the p rule can be understood as resulting from a process of *contraction*[8] of the membrane into a single point: if the bubble contains some items Δ, then this process fails because the membrane gets stuck on the boundaries of Δ. This is a topological way to check the emptiness of a bubble, which has the benefit of being completely *continous*, in addition to being very clear visually.

ℝ**ESOURCE** The contraction rule c could be mapped to a specific triggering input when starting to drag a blue item A (e.g. a shortkey if a keyboard is available, or a long press on the item on a touchscreen), which has the effect of keeping a copy of A at its original location in addition to moving the item[9]. As for the weakening rule w, it could be available as a contextual action when selecting blue items.

8: Not to be confused with the contraction rule c.

9: This mechanism is quite standard in GUIs that manipulate duplicable resources like file managers, where one maintains the `CTRL` key to enable copy mode. It was also chosen by K. Chaudhuri to implement contraction in his `Profint` prototype for subformula linking in intuitionistic logic [40].

$$
\begin{array}{cccccc}
A & B & \rightarrow & A \circledast B & & \circledast \\
A & B & \rightarrow & A \oslash B & & \oslash
\end{array}
$$

Figure 7.6.: Linkage creation rules in BJ

Identity Although the i↓ rule only corresponds to the base case of DnD actions, it would be easy to integrate the full SFL semantics of DnD actions directly in BJ. Indeed our SFL rules (Figure 2.4) are already expressed as rewriting rules, just like the graphical rules of BJ (Figure 7.4). Thus it is just a matter of adding linkage creation rules like those of Section 3.7, but between adjacent formulas in a solution (Figure 7.6).

The cut rule was handled in `Actema` with a separate +hyp button, which adds the cut formula A (input by the user in a dialog box) as a new hypothesis in the current goal, and as the conclusion in a new subgoal (see Subsection 2.2.1). Since subgoals are now reified as bubbles, the i↑ rule could be mapped instead to a contextual action available on any red item Δ, which would have the effect of spawning a bubble around it with a blue item A, and another bubble nearby it with a red item A.

Heating For ℍ-rules that spawn bubbles like ∧+, it is important that bubbles stay close to the item being clicked, in order to make the transformation visually clear. One could even imagine a small animation that smoothly turns the main connective into bubbles, to convey more effectively the intuition that heating rules break logical connectives seen as chemical bonds.

Beyond the recovered uniformity of the user interface in terms of the chemical metaphor, BJ exhibits some features that are interesting both on the proof-theoretical and user-experience levels:

Factorization It implements a form of *context-sharing* between subgoals: that is, one can perform transformations on shared hypotheses (forward reasoning) without going back to a proof state anterior to the splitting of said subgoals. This should simplify the navigation in the proof as it is being constructed, by avoiding the need to locate these splitting points. In fact often beginners (but also occasionally seasoned users) do not have the reflex to do this, precisely because the interface makes it difficult. This results in proofs with a lot of duplicated arguments, since splitting goals systematically duplicates the context of hypotheses. Thus bubbles can be seen as a mechanism that favors by default a style of proof with better factorization of subproofs.

Parallelism The locality of rewriting rules makes it possible for multiple users to reason on different subgoals of the same proof state *at the same time*, without compromising soundness. Combined with the above factorization property, this enables *asynchronous* collaborative setups, where various users can work on the same proof in parallel (e.g. through an online web interface), while still benefitting from the knowledge built by collaborators in shared contexts.

Navigation The tree structure of subgoals is immediately apparent in the proof state through the nesting of bubbles. Thus part of the information on the proof construction process, which was made implicit and temporal in the proof state history, is now made explicit and spatial in the proof state itself[10].

There are multiple ways to visualize trees on a planar surface, but if we are to maintain the bubble metaphor, *zoomable user interfaces* (ZUI)

10: This concern of finding an explicit graphical representation of the "motions of reasoning *in actu*", and not only the states of mind, can be found already in the works of Peirce on his existential graphs [214, pp. 112–113]. We will come back to this in Chapter 9.

seem to be a right fit: they allow for efficient space management and navigation, and *zooming in* intuitively conveys the idea of focusing on a specific subgoal. One could also *zoom out* to have an overview of the different subgoals and their shared context, something which is hard to do in current proof assistants.

When zoomed in on a subgoal, the shared contexts around it will not be visible anymore. While this is useful to focus attention and avoid being distracted by other subgoals, it can quickly become cumbersome for the user to always have to zoom out in order to retrieve hypotheses from these shared contexts. One solution would be to rebrand the context zone of `Actema` as a *global* context zone, where all the shared contexts available in the subgoal under focus are merged in a single list, and immediately accessible for manipulation. Of course actions performed in the context zone would be reflected in the proof canvas, and vice versa.

Goal diffing From a user perspective, the locality of rules means that applying some action to one or two items will not involve other items[11]. Non-local rules are less natural for a beginner because they modify a global state (here other items and subgoals) which is not clearly correlated to the transformed data, often because it is not immediately visible.

11: The only exceptions are clicks on blue $\perp$, $\vee$ and $\supset$, but the only extra item they involve is the conclusion.

For instance in `Actema`, many users have reported difficulties in understanding the effect of click actions that create new subgoals. A first reason that can easily be remedied, is that there was not enough visual feedback to indicate the newly created tabs. But a deeper limitation is that the user needs to explicitly focus on these subgoals to show their content, which they might not do immediately. And then it gets difficult to keep track of the origin of said subgoals without a way to visualize the tree structure of the proof.

All these concerns can be addressed within the bubble metaphor: since bubbles are items freely positioned on the proof canvas, all the new items produced by an action can stay near the location where the action was initiated (i.e. the click or drop location); and since all transformations are local, all items not involved in the action can have their locations preserved. In other words, bubbles make it easier to understand the *difference* between a goal and subgoals generated by a proof action, which is crucial when learning the semantics of actions through practice.

Symmetric Bubble Calculi | 8.

8.1 Non-determinism and iconicity 117
8.2 Symmetric interpretation 118
8.2.1 Distributing conclusions 118
8.2.2 Saturated solutions 119
8.2.3 Backward factorization . 120
8.3 Coloring bubbles . . . 121
8.3.1 Red bubbles 121
8.3.2 Blue bubbles 122
8.3.3 Polarized interpretation . 123
8.4 Designing for properties 124
8.5 Symmetric calculus . 127
8.5.1 Graphical presentation . 127
8.5.2 Sequent rules 130
8.5.3 Bubble rules 130
8.6 Soundness 133
8.6.1 Heyting and Brouwer algebras 133
8.6.2 Duality 135
8.6.3 Shallow soundness 137
8.6.4 Contextual soundness . . 140
8.7 Completeness 143
8.8 Invertible calculus . . 148
8.8.1 Modifying rules 148
8.8.2 Semi-automated proof search 151
8.8.3 Failure of full iconicity . . 154

Each city receives its form from the desert it opposes; and so the camel driver and the sailor see Despina, a border city between two deserts.

Italo Calvino, *Invisible Cities*, 1972

In this chapter, we explore to what extent the bubble calculus of Chapter 7 can be made more *symmetric*, by relaxing the restriction that solutions must contain at most one conclusion. At a surface level, our approach is similar to that of Gentzen, who went from his single-conclusion sequent calculus LJ to the multi-conclusion calculus LK. Like him, we will uncover beautiful dualities that were hidden by the asymmetry of the initial calculus. But by sticking unwaveringly to intuitionism, we will be led to the exotic territory of bi-intuitionistic logic, an intermediate logic that conservatively extends intuitionistic logic, and in particular does not prove the law of excluded middle.

An underlying thread of our investigation will be the quest for a *fully iconic* proof system, where all logical connectives can be replaced by appropriate new kinds of bubbles. This will make us rediscover many principles already studied in the deep inference literature, with topological intuitions of the bubble metaphor shedding a new light on them. We will end up with two symmetric bubble calculi, each with its own tradeoff on the properties satisfied by inference rules. In particular, the ability of the calculi to factorize *backward* in addition to forward proof steps might prove useful to build concise proofs, all through direct manipulation.

The chapter is organized as follows: in Section 8.1 we motivate our quest for a system where all introduction rules for logical connectives are *invertible*, to reduce non-determinism in proof search and enable a fully *iconic* approach to proof building. To that effect, we relax in Section 8.2 the restriction to single-conclusion solutions, which requires a new distinction between *saturated* and *unsaturated* solutions. This gives rise in Section 8.3 to an extension of the syntax of solutions, where bubbles can themselves be *polarized*. In Section 8.4 we identify key properties that will guide the design of inference rules, some of which were already aimed for implicitly throughout the evolution of our concept of bubble.

In Section 8.5 we introduce a core *symmetric bubble calculus* for classical logic called system B, in reference to the symmetric system L of Herbelin [118]. Then in Section 8.6 we prove the soundness of system B, and show that by removing selectively some inference rules that define the *porosity* of polarized bubbles, one gets intuitionistic, dual-intuitionistic and bi-intuitionistic logic as fragments. In Section 8.7 we support this claim by showing that the bi-intuitionistic fragment is not only sound, but also *cut-free complete* with respect to the cut-free nested sequent calculus DBiInt of Postniece [206].

[118]: Herbelin (2008), *Duality of computation and sequent calculus : a few more remarks*

[206]: Postniece (2009), 'Deep Inference in Bi-intuitionistic Logic'

Finally in Section 8.8, we introduce a fully invertible variant of system B that we conjecture to be complete, and present a canonical way to search for proofs in this system. Unfortunately, invertibility does not entail the full iconicity of the system, and we reflect on the fundamental reasons that might prevent any variant of system B from being fully iconic.

Note

Although we include rules for quantifiers, in this thesis we only treat the soundness and completeness of bubble calculi for *propositional* logic. Indeed quantifiers would make the algebraic semantics more involved when proving soundness, and during our literature review we found very few proof systems for bi-intuitionistic logic supporting them, at least none suitable for our syntactic completeness proof. More generally, bi-intuitionistic logic has received less attention in the setting of FOL, probably because it is *not* a conservative extension of intuitionistic FOL, but only of *constant-domain* intuitionistic FOL (see [8, 53]).

8.1. Non-determinism and iconicity

In all known sequent calculus formulations of intuitionistic logic, there are at least two rules which are invariably *non-invertible*:

1. a left introduction rule for $\supset$ (there might be many ones, as in the calculus LJT of Dyckhoff [73]);
2. the right introduction rule for either:
   - $\vee$ when sequents have at most or exactly one conclusion;
   - $\supset$ when sequents have multiple conclusions, e.g. in the multi-conclusion variant of LJT in [73].

[73]: Dyckhoff (1992), 'Contraction-Free Sequent Calculi for Intuitionistic Logic'

In BJ, this means that click actions on blue $\supset$ and red $\vee$ need to be performed in a specific order to be able to complete proofs.

In his thesis [108], Guenot introduced a specific kind of nested sequent system where — like in BJ — inference rules can be expressed as rewriting rules. An interesting feature of these systems is that they satisfy a *decomposability* property: all introduction rules for connectives are *invertible*, and formulas can be completely decomposed by using them until atoms are reached, before applying other rules. Thus introduction rules are in a sense *admissible* in these systems, because every formula can be translated into an equivalent pure nested sequent with the same number of atoms[1]. Non-determinism then arises in the choice of atoms that are to be connected in axiom rule instances, as well as the choice of subsequents to be duplicated for reuse.

[108]: Guenot (2013), 'Nested Deduction in Logical Foundations for Computation'

1: As far as we know, the admissibility of introduction rules is not proved, let alone mentioned in [108]. This is our own observation which lacks a proper formal proof, and is thus subject to caution.

In our graphical setting, this would translate into an interface where all click actions are redundant. Although we already considered this possibility in Section 3.7, it goes further here by making even *logical connectives* superfluous, since all other rules work purely on the structure of sequents.

This means that all logical connectives could be replaced by metaphorical constructs like bubbles, which suggest *physically* the possible transformations on the proof state. Unfortunately, the systems in [108] only handle classical logic, and the implicative fragment of intuitionistic logic. Thus began our quest for a bubble calculus in the style of Guenot capturing full intuitionistic logic[2].

2: Other nested sequent systems for full intuitionistic logic exist [78, 206], but they are based on tree-shaped proofs, and thus ignore the whole *raison d'être* of our concept of bubble.

8.2. Symmetric interpretation

8.2.1. Distributing conclusions

The first direction we followed was to relax the constraint that solutions must be single-conclusion. Indeed as already noted in Section 5.1, a notable property of sequent calculi with multiple conclusions is that their right introduction rule for $\vee$ is invertible.

The main difficulty lies in the way one should interpret a multi-conclusion solution S as a formula $[\![S]\!]$. If we just take the asymmetric interpretation (Definition 7.3.1) and group conclusions disjunctively instead of conjunctively, we get

$$[\![\Gamma \langle \mathcal{S} \rangle \Delta]\!] = \bigwedge \Gamma \supset \bigvee \Delta \wedge \bigwedge_{S \in \mathcal{S}} [\![S]\!] \tag{8.1}$$

But this interpretation breaks on the 0-ary case when Δ is empty: instead of seeing $\Gamma \langle \mathcal{S} \rangle$ as a node of the proof tree with hypotheses Γ and subgoals $\mathcal{S}$, it trivializes it to $[\![\Gamma \langle \mathcal{S} \rangle\ ]\!] = \bigwedge \Gamma \supset \bot$, i.e. a goal where one has to find a contradiction in Γ; which is obviously not what we have in mind.

A key observation was that in the rules of multi-conclusion sequent calculi, one usually distributes the context Δ of conclusions in all premisses: this restores a perfect symmetry with respect to the context of hypotheses Γ, as illustrated by the $\wedge$R$*$ rule (Figure 8.1). Then our idea was that instead of implementing distribution/sharing of conclusions inside inference rules, we could do it implicitly in the interpretation of solutions. This is already what happens in the asymmetric interpretation for hypotheses; indeed the context Γ is shared among subgoals, because:

$$\frac{\Gamma \Rightarrow A, \Delta \quad \Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow A \wedge B, \Delta} \wedge\mathsf{R}*$$

Figure 8.1.: Multi-conclusion right introduction rule for conjunction

1. it appears on the left side of an implication $\supset$;
2. bubbles are joined conjunctively;
3. implication distributes over conjunction thanks to the equivalence $A \supset B \wedge C \simeq (A \supset B) \wedge (A \supset C)$.

But what does it mean precisely to share conclusions among subgoals? If we consider the two solutions of Figure 8.2, we would like to have $[\![S]\!] \simeq [\![T]\!] \simeq (A \supset B \vee E) \wedge (C \supset D \vee E)$. Since disjunction distributes over conjunction, a first naive try would give the following interpretation, where we just replace $\wedge$ by $\vee$ compared to Equation 8.1:

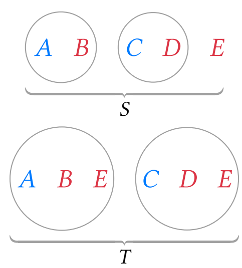


Figure 8.2.: Distributing conclusions in bubbles

$$[\![\Gamma \langle \mathcal{S} \rangle \Delta]\!] = \bigwedge \Gamma \supset \bigwedge_{S \in \mathcal{S}} [\![S]\!] \vee \bigvee \Delta \tag{8.2}$$

But this immediately fails whenever $\mathcal{S} = \varnothing$, because it trivializes to $\bigwedge \Gamma \supset \top \vee \bigvee \Delta \simeq \top$ instead of $\bigwedge \Gamma \supset \bigvee \Delta$.

8.2.2. Saturated solutions

The only way we found around the above defect was to internalize *syntactically* a distinction between two kinds of solutions, by assigning them one of two statuses[3]:

3: In the terminology of Martin-Löf, we could say that we now have two distinct forms of *judgment*.

- *saturated* solutions $\Gamma \ \langle \mathcal{S} \rangle \ \Delta$ correspond to branching nodes in the proof tree, or to closed leaves when $\mathcal{S} = \varnothing$ (i.e. solved subgoals). Thus it becomes sensical to have $[\![\Gamma \ \langle\rangle \ \Delta]\!] = \top$. In the asymmetric interpretation, saturated solutions were encoded by solutions with no conclusions;
- *unsaturated* solutions $\Gamma \Rightarrow \Delta$ correspond to open leaves in the proof tree (i.e. unsolved subgoals). In the asymmetric interpretation, they were encoded by solutions with one conclusion.

Then we keep the last proposed interpretation given in Equation 8.2 for saturated solutions, and interpret unsaturated solutions like usual sequents:

$$[\![\Gamma \Rightarrow \Delta]\!] = \bigwedge \Gamma \supset \bigvee \Delta$$

To be able to abstract from the saturation status of solutions, we reframe the syntax of solutions with so-called *branching operators* $\triangleright$:

$$\begin{aligned} S, T, U &::= \Gamma \triangleright \Delta \\ \triangleright, \blacktriangleright &::= \Rightarrow \mid \langle \mathcal{S} \rangle \end{aligned}$$

Graphically, saturated solutions with no bubbles can be distinguished from unsaturated solutions by painting their *background* on the proof canvas in green, the intent being to suggest that they have already been solved. A pathological example is the distinction between the saturated empty bubble ◯ and the unsaturated empty bubble ◯, who are interpreted respectively by $[\![\, \langle \, \langle\rangle \, \rangle \,]\!] = \top$ and $[\![\, \langle \Rightarrow \rangle \,]\!] = \bot$.

Remark 8.2.1 The terminology of "saturation" is also inspired by *chemistry*: in this context, a solution is saturated when it has reached *equilibrium*, meaning that the chemical reaction of *dissolution* cannot happen anymore. The analogy applies to our logical setting: a solution is saturated when it has reached *truth*, meaning that the logical reaction of *backward linking* cannot happen anymore. *Forward* linking is still possible though and may produce additional shared knowledge, akin to solid sugar accumulating at the bottom of a *supersaturated* container of water.

8.2.3. Backward factorization

Now coming back to our motivating example of Figure 8.2, the interpretation of Equation 8.2 still fails, because we associate two non-equivalent formulas to S and T. To show this, let us try to derive the equivalence through some algebraic developments:

$$\begin{aligned}
[\![S]\!] &= \top \supset ((A \supset B) \wedge (C \supset D)) \vee E \\
&\simeq ((A \supset B) \wedge (C \supset D)) \vee E \\
&\simeq ((A \supset B) \vee E) \wedge ((C \supset D) \vee E) \\
&\simeq (A \supset B \vee E) \wedge (C \supset D \vee E) \qquad (8.3) \\
&\simeq ((A \supset B) \wedge (C \supset D)) \vee E \\
[\![T]\!] &= \top \supset ((A \supset B \vee E) \wedge (C \supset D \vee E)) \vee \bot
\end{aligned}$$

Wait, we did manage to prove it! The trick resides in Equation 8.3, which uses twice the equivalence $(A \supset B) \vee C \simeq A \supset (B \vee C)$. It turns out that this equivalence is true in classical logic, but *not* in intuitionistic logic. More precisely, it is the implication $G \triangleq (A \supset (B \vee C)) \supset ((A \supset B) \vee C)$ which is not provable intuitionistically, since it can easily be shown equivalent to the law of excluded middle[4]. Thus according to this interpretation, S entails T but T does not entail S, which means that it is not able to account for the *factorization* of common conclusions in distinct subgoals.

4: This was already noticed in [49], with the linear version $(A \multimap (B \parr C)) \multimap ((A \multimap B) \parr C)$ of G called *Grishin (a)* and its converse *Grishin (b)*. More precisely, it is affirmed that while Grishin (b) is valid in FILL, which is the extension of intuitionistic multiplicative linear logic (IMLL) with $\parr$, adding Grishin (a) makes FILL collapse to MLL.

To remedy this situation, we opted for a different strategy: instead of finding a logical formula capturing the distributive semantics of conclusions over subgoals, we hardcode this semantics by defining the interpretation function over saturated solutions through *non-structural* recursion. This gives the following final definitions:

Definition 8.2.1 (Mix operator) *The commutative* mix operator $\uplus$ *on solutions is defined by:*

$$(\Gamma \rhd \Delta) \uplus (\Gamma' \Rightarrow \Delta') = \Gamma, \Gamma' \rhd \Delta, \Delta'$$
$$(\Gamma \langle \mathcal{S} \rangle \Delta) \uplus (\Gamma' \langle \mathcal{S}' \rangle \Delta') = \Gamma, \Gamma' \langle \mathcal{S}; \mathcal{S}' \rangle \Delta, \Delta'$$

Definition 8.2.2 (Symmetric interpretation) *The* symmetric interpretation $[\![-]\!]$ *of a solution is defined recursively by:*

$$[\![\Gamma \langle \mathcal{S} \rangle \Delta]\!] = \bigwedge_{S \in \mathcal{S}} [\![S \uplus (\Gamma \Rightarrow \Delta)]\!]$$
$$[\![\Gamma \Rightarrow \Delta]\!] = \bigwedge \Gamma \supset \bigvee \Delta$$

This is the right approach for interpreting solutions with multiple conclusions, as will be demonstrated formally in Section 8.6.

8.3. Coloring bubbles

8.3.1. Red bubbles

With our new symmetric interpretation, we can start generalizing the rules of BJ to multiple conclusions. While for most rules one just has to replace single-conclusion (resp. no-conclusion) solutions with unsaturated (resp. saturated) ones[5], the ⊃+ rule stands out as particularly problematic. Indeed if we content ourselves with the natural generalization ⊃+c of Figure 8.3, then we can easily build a proof of the excluded middle like in Figure 5.2, and thus collapse to classical logic. This fact is well-known in the literature on multi-conclusion intuitionistic sequent calculi, and the solution is usually to discard the context of conclusions Δ, as in the ⊃R∗i rule of Figure 5.3. But this would make our rule both non-local and non-invertible.

5: More details will be given in the next section

$$\frac{\Gamma, A \rhd B, \Delta}{\Gamma \rhd A \supset B, \Delta}\ {\supset}{+}\mathsf{c}$$

Figure 8.3.: Classical multi-conclusion version of ⊃+

A better solution comes from the nested sequent systems of Fitting [78] and Clouston et al. [49], where sequents can appear as *conclusions* of other sequents. In our chemical metaphor, this corresponds to having *red bubbles*. Then the key idea is to allow *hypotheses* to flow into sequents that appear as conclusions[6], but to forbid *conclusions* to do so. Graphically, this means that blue items can enter red bubbles (rule f−+↓ of Figure 8.4), but red items cannot: this is reminiscent of the electromagnetic phenomemon of *repulsion* between objects charged with the same polarity.

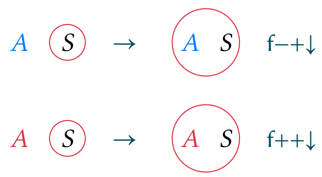


Figure 8.4.: F-rules for red bubbles

[78]: Fitting (2014), 'Nested Sequents for Intuitionistic Logics'

[49]: Clouston et al. (2013), 'Annotation-Free Sequent Calculi for Full Intuitionistic Linear Logic'

6: This corresponds to the *Lift* rule of [78] and pl_1 rule of [49].

To illustrate why this works, let us consider how one can manipulate with red bubbles the classical equivalence $(A \supset B) \vee C \simeq A \supset (B \vee C)$, that we already encountered in the previous section. The beggings of the proofs for both directions of the equivalence are depicted parallelly in Figure 8.5. Indeed both proofs have a very similar structure:

1. the first step is to decompose the conclusion with the new version of the rules ∨+ and ⊃+. While the former simply splits disjunctions in two, the latter encapsulates the antecedant and consequent of implications in a red bubble: the aim is to forbid the use of the antecedant to prove conclusions other than the consequent, as will become apparent later;
2. then in both cases we want to apply the hypothesis A in a forward step, either with $A \supset B$ or $A \supset (B \vee C)$. To do so, we need to bring the two hypotheses together in the same solution. And since items are trapped within bubbles, the only way to go is to move the blue hypothesis inside the red bubble with the f−+↓ rule;
3. this time we decompose the hypothesis with the new version of the rules ∨− and ⊃−. They are basically a local variant of those of BJ: we encapsulate both subformulas in separate bubbles, but without touching the conclusions of the ambient solution;
4. now that all formulas have been decomposed, it only remains to bring together dual atoms for annihilation, and pop all empty bubbles. In Grishin (b) this is easy, because all necessary movements (indicated by green arrows) are valid: they only cross gray bubbles inward. In Grishin (a) this works for A and B, but not for C (orange dotted arrow): it would cross the red bubble, which is expressedly forbidden.

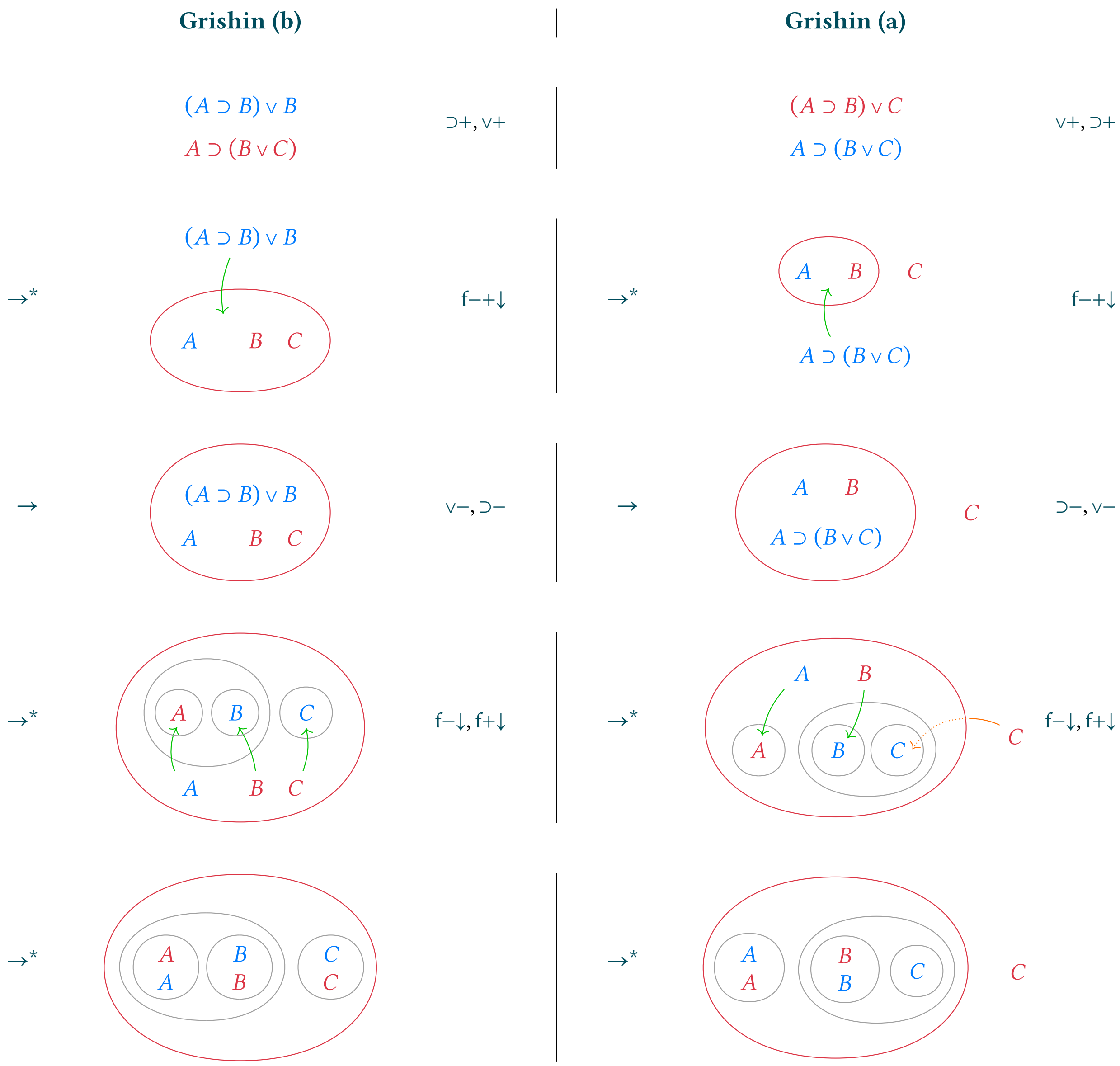


Figure 8.5.: Proof attempts for Grishin (a) and Grishin (b)

Thus in order to prove Grishin (a) and recover classical logic, it suffices either to add the f++↓ rule allowing red items to enter red bubbles (Figure 8.4), or to use the ⊃+c rule which avoids red bubbles altogether. In the following we will settle for the first option: we find it more elegant, because it explains the distinction between intuitionistic and classical logic as a kind of *physical law* independent of logical connectives.

8.3.2. Blue bubbles

Now it is only natural to wonder: since bubbles can be colored in red, or charged positively, would it also make sense to have *blue* bubbles charged *negatively*? The answer is *yes*, but we need to broaden our logical view and consider more exotic beasts: the adequately named *dual-intuitionistic* logic, and *bi-intuitionistic* logic.

For the moment, let us stay at a purely syntactic level. The idea is very simple, and can be summarized in two words: *color swapping*. Thus the law that "blue items can enter red bubbles, but red items cannot" becomes a new law that "red items can enter blue bubbles, but blue items cannot", which is enforced by allowing only the use of the f+−↓ rule in Figure 8.6. Well this is neat, but will not be of much use if there is no way to spawn blue bubbles. Be it as it may: we can just craft a new logical connective! Since red bubbles are produced by the implication connective $A \supset B$, we define a dual *exclusion* connective $A \subset B$ (read "A excludes B"[7]), whose heating rules are those of $\supset$ where blue and red have been swapped (Figure 8.7).

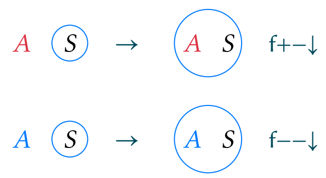


Figure 8.6.: F-rules for blue bubbles

7: We ask for the reader's leniency regarding our choice of symbol and terminology: in set theory this would be total nonsense, since $A \subset B$ would read "A is included in B". Even worse, in the boolean algebra induced by set operations, $A \subset B$ is interpreted as A *implies* B... But we want to emphasize the duality between exclusion and implication by mirroring the symbol, as it is traditionally done with conjunction $\wedge$ and disjunction $\vee$.

Not very surprisingly, the exclusion connective has already been studied in the literature on intuitionistic logic, starting with the seminal paper of Rauszer on *Heyting-Brouwer logic*, i.e. intuitionistic logic to which we add exclusion [210]. In this paper, exclusion was called *pseudo-difference*, to evoke its close connection with set-theoretical difference. Indeed given two sets A and B, one can define the set $A \setminus B$ by comprehension as $\{x \mid x \in A \wedge x \notin B\}$, which is the set A from which all elements of B have been *excluded*. With an interpretation in boolean algebras, this corresponds to the classical connective defined by the truth table of $A \wedge \neg B$, which is dual to the truth table of $\neg A \vee B$ defining material implication.

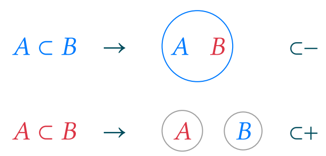


Figure 8.7.: H-rules for exclusion $\subset$

While the first paper of Rauszer [210] belongs to the Polish tradition of algebraic logic, she also explored in later works the proof-theoretic [209] and model-theoretic [211] sides of the question. Many authors have then deepened the proof theory of exclusion, whether in isolation from implication in *dual-intuitionistic* logic [102, 245], or with both connectives in *bi-intuitionistic* logic as in Rauszer's original work[8] [205, 207]. In particular, we are going to rely in Section 8.7 on the deep inference calculus developed by Postniece in her thesis [207] to get completeness and cut admissibility of our symmetric bubble calculus introduced in the next section.

[210]: Rauszer (1974), 'Semi-Boolean Algebras and Their Applications to Intuitionistic Logic with Dual Operations'

[209]: Rauszer (1974), 'A Formalization of the Propositional Calculus of H-B Logic'

[211]: Rauszer (1977), 'Applications of Kripke Models to Heyting-Brouwer Logic'

[102]: Goré (2000), 'Dual Intuitionistic Logic Revisited'

[245]: Urbas (1996), 'Dual-Intuitionistic Logic'

8: Crolard [53] and Aschieri [8] have also explored the computational counterpart of exclusion through the Curry-Howard correspondence, which is claimed by the first author to be a typing operator for *first-class coroutines*.

[205]: Pinto et al. (2011), 'Relating Sequent Calculi for Bi-intuitionistic Propositional Logic'

[207]: Postniece (2010), 'Proof theory and proof search of bi-intuitionistic and tense logic'

8.3.3. Polarized interpretation

Let us now extend the formal definition of bubbles so that they can be colored:

Definition 8.3.1 (Bubble) *A bubble is a solution enclosed in a membrane. The membrane can be either unpolarized (neutral), charged positively, or charged negatively.*

Neutral bubbles are the usual ones depicted in gray, while positive and negative bubbles correspond respectively to red and blue bubbles. We also update the definition of solutions, which can now be unsaturated or saturated:

Definition 8.3.2 (Solution) *A solution is a multiset of ions and bubbles. It can be either saturated or unsaturated, and unsaturated solutions cannot contain neutral bubbles. Solutions S can be represented textually with the following syntax:*

$$S, T, U ::= \Gamma \rhd \Delta \qquad \mathcal{S} ::= S_1 ; \ldots ; S_n$$
$$I, J, K ::= A \mid S \qquad \Gamma, \Delta ::= I_1, \ldots, I_n$$
$$\rhd, \blacktriangleright ::= \Rightarrow \mid \langle \mathcal{S} \rangle$$

In particular, an item I can be either a formula A or a bubble/solution S.

Note that in the textual syntax, bubbles are identified with *subsolutions* (Definition 7.3.5), and their polarity is determined by their position relative to branching operators; that is, for any solutions S, T, U such that $T \prec U$, S is either:

- *neutral* if $T = \Gamma \langle \mathcal{S} \rangle \Delta$ and $S \in \mathcal{S}$;
- *positive* if $T = \Gamma \rhd \Delta$ and $S \in \Delta$;
- *negative* if $T = \Gamma \rhd \Delta$ and $S \in \Gamma$.

Then we need to split our symmetric interpretation accordingly, so that positive bubbles are mapped to implications, and negative bubbles to exclusions[9]:

9: Here we took inspiration from the work of Clouston et al. on nested sequents for FILL [49].

Definition 8.3.3 (Polarized symmetric interpretation) *The symmetric positive interpretation and negative interpretation of solutions $[\![-]\!]^+$ and $[\![-]\!]^-$ are defined by mutual recursion as follows:*

$$
\begin{aligned}
[\![A]\!]^+ &= A & [\![A]\!]^- &= A \\
[\![\Gamma \langle \mathcal{S} \rangle \Delta]\!]^+ &= \bigwedge_{S \in \mathcal{S}} [\![S \uplus \Gamma \Rightarrow \Delta]\!]^+ & [\![\Gamma \langle \mathcal{S} \rangle \Delta]\!]^- &= \bigvee_{S \in \mathcal{S}} [\![S \uplus \Gamma \Rightarrow \Delta]\!]^- \\
[\![\Gamma \Rightarrow \Delta]\!]^+ &= [\![\Gamma]\!]^- \supset [\![\Delta]\!]^+ & [\![\Gamma \Rightarrow \Delta]\!]^- &= [\![\Gamma]\!]^- \subset [\![\Delta]\!]^+ \\
[\![\Gamma]\!]^+ &= \bigvee_{I \in \Gamma} [\![I]\!]^+ & [\![\Gamma]\!]^- &= \bigwedge_{I \in \Gamma} [\![I]\!]^-
\end{aligned}
$$

One can easily check that the interpretation of a solution that has no negative (resp. positive) subsolution will not contain any occurrence of the exclusion (resp. implication) connective. This will be crucial later to represent proofs of both intuitionistic, dual-intuitionistic and bi-intuitionistic logic in the same system.

8.4. Designing for properties

With our new syntax and interpretation of solutions at hand, we can design a new proof calculus that includes the rules previously discussed for manipulating polarized bubbles. The rich structure of solutions offers many possibilities in the precise formulation of rules, depending on the properties we expect from the calculus. We identified *six* of these properties, whose consequences range from aesthetic and theoretical considerations on paper, to concrete usability matters in a graphical proof building interface. Let us summarize them, going from the *practical* to the *theoretical*:

Invertibility A rule is invertible when it could in principle be applied in the converse direction, while staying logically sound[10]. In other words, it corresponds to a logical *equivalence*: when all rules in a (bubble) calculus are invertible, we get that $S \to T$ implies $[\![S]\!] \simeq [\![T]\!]$. This entails in particular that a user can apply the rule without fear of turning a provable goal into an unprovable one[11], eliminating an important source of non-determinism in proof search: the need for *backtracking*[12].

10: The *"in principle"* part is important: more often than not, adding the converse of a rule only brings unnecessary complexity in proof search, especially in a user interface that aims for simplicity.

11: Assuming that the calculus is *complete* (Section 8.7).

12: See also Section 5.1 for a discussion on this matter.

Decomposability We already mentioned this property in Section 8.1 as one of the main motivations for this chapter: the ability to decompose all logical connectives "for free", and thus reason solely on solutions that comprise only bubbles and atomic formulas. As far as we know, it has never been identified explicitly in the literature before, although it can loosely be seen as an extension of the *decomposition theorem* in the calculus of structures [242, Theorem 4.1.3].

[242]: Tubella et al. (2019), 'Introduction to Deep Inference'

One reason is that logical connectives are widely considered as *primitive* in the tradition of mathematical logic: they *are* the objects of the reasoning activity, rather than a tool for representing and structuring arguments. Thus the idea of an alternative does not even occur. But even if it does, it is not clear that it would bring any interesting viewpoint on the problems usually studied in proof theory. In our case, it is motivated by a very concrete application: making formal proofs accessible to a broader audience, by replacing symbolic and linguistic means of representation by iconic and directly manipulable ones.

Digression

One could argue that more "semantic" approaches in proof theory have achieved connective-free explanations of proofs, like strategies in game semantics or the combinatorial proofs of D. Hughes [117]. But this is more of a side effect than a goal of these approaches, which intentionally abstract from the syntactic process of building proofs. A notable exception is the Girardian line of works starting from *ludics* [93] and culminating in *transcendental syntax* [76], where both frameworks are founded upon the syntactic mechanisms of proof search (focusing in sequent calculus, and unification in the resolution algorithm of Robinson, respectively). Here the aim to rid proofs of connectives is greatly emphasized by Girard, but the focus is again on *proofs* and not *proof states*. Also Girard embraces the full space of incomplete but also *incorrect* proofs, while we still want a framework where proofs are correct by construction.

Factorizability We say that a proof calculus is *factorizable* when it makes it easier to avoid duplicating arguments in subproofs. In Section 7.4, we already remarked that the ability to share hypotheses between subgoals in BJ enables the factorization of *forward* reasoning steps at any stage of the proof construction. With our new symmetric interpretation of multi-conclusion solutions, we will now be able to factorize *backward* reasoning steps as well, which was in fact the main motivation behind the example of Figure 8.2.

Locality There does not seem to be a general consensus on what it means precisely for an inference rule to be *local*. This terminology has been employed by various authors in proof theory, in ways that are often hard to compare. For instance in [183], rules are said to be local because the contexts of hypotheses involved in a rule are located in the sequents of that rule, by opposition to natural deduction rules in their labelled presentation where hypotheses are located in arbitrary distant leaves of the derivation. In the setting of deep inference, local rules are those that can be applied without "inspection of expressions of arbitrary size"[13]. Finally in his transcendental syntax, Girard evokes a related but more elusive notion, concerned with the *genericity* of logical objects involved in a rule[14].

[183]: Negri et al. (2001), *Structural Proof Theory*

13: Definition 2.1.1 in [242]. The same definition is used in [241].

14: See the section *Globality and locality in logical systems* in [76, Chapter 6].

Our conception of locality is related to all the previous ones, although it is guided by the idea of direct manipulation of logical entities by humans, rather than purely proof-theoretical considerations. For instance, BJ has some locality in the deep inference sense because all rules are applicable in arbitrary contexts; but we relax the *atomicity* constraint that reduces $\mathbb{I}$-rules and $\mathbb{R}$-rules to their atomic version, because it would be unnecessarily restrictive for the purpose of building proofs manually. Still, we want to avoid as much as possible referring to generic objects that are not directly related to the manipulated data, in the spirit of Girard's locality. A typical example is the elimination rule $\vee\mathsf{e}$ for disjunction in natural deduction, corresponding to the $\vee-$ rule of BJ that involves an arbitrary conclusion Δ. The benefits of locality from a UX point of view have already been discussed in Section 7.4.

Linearity We consider an inference rule to be *linear* when it preserves the number of atomic formulas in solutions. This is a strong requirement, which for instance excludes the $\mathbb{I}$-rules of BJ since they can insert or remove (even numbers of) atoms. Thus we cannot achieve full linearity in that sense, but it is still interesting to maximize it.

The first reason is *methodological*: by the words of its creator A. Guglielmi, *"[...] deep inference is obtained by applying some of the main concepts behind linear logic to the formalisms, i.e., to the rules by which proof systems are designed."* [111].

[111]: Guglielmi (2014), *Deep Inference*

The second reason is *computational*: linearity can enable a measure on solutions that is strictly decreasing with the application of rules, avoiding infinite loops during proof search as in the calculus LJT of Dyckhoff [73].

The third reason is *ergonomical*: as already remarked by the authors of the Proof-by-Pointing paradigm[15], rules that systematically duplicate formulas can quickly overload the goal with useless copies, making it harder to read and navigate.

15: Section 4.1 of [18].

Symmetry In classical logic, both sequent calculi like LK and deep inference systems like CoS are known for their very rich *symmetries*. In fact, one of our ambitions with bubbles was to bring back the symmetry of classical logic in a constructive setting, without resorting to linear logic. This chapter stems in great part from our lack of satisfaction with the asymmetry at work in the BJ calculus, which looked quite unnatural. Of course we will not be able to completely eliminate it, but it will be distilled into the flow rules governing the *porosity* of bubbles that were hinted at in Section 8.3, rather than through the arbitrary restriction of sequents to one conclusion[16].

16: Whether it is enforced in the syntax of sequents themselves, or through restriction on rules that manipulate conclusions like contraction or the right introduction rule for $\supset$.

Our treatment of dual-intuitionistic and bi-intuitionistic logic through blue bubbles is also motivated by this quest for symmetry. It should be noted that although we use naming conventions for rules that resemble those of CoS (e.g. with $\mathbb{I}$-rules), we do not aim for a perfect symmetry where one can get a complete calculus by simply taking the dual of each rule. Thus we will content ourselves with the hypothesis/conclusion symmetry coming from sequent calculus. Interestingly, the calculus ISgq of Tiu for intuitionistic predicate logic does the opposite, by having a perfect dual system cISgq, but no symmetries among its switch rules (the equivalent of our F-rules) [241].

[241]: Tiu (2006), 'A Local System for Intuitionistic Logic'

In the next section we present a core calculus called system B that maximizes *symmetry*, *linearity* and *locality*. In our opinion this makes for a good proof-theoretical foundation, around which variant calculi with different tradeoffs can be designed.

8.5. Symmetric calculus

8.5.1. Graphical presentation

Like the asymmetric bubble calculus BJ, the rules of system B enjoy both a sequent-style and a graphical presentation, given respectively in Figure 8.8 and Figure 8.9. The presence of saturated and unsaturated solutions complicates quite a bit the graphical representation of rules, thus some explanations are in order:

Saturated solutions In Section 8.2, we mentioned that saturated solutions with no neutral bubbles can be distinguished visually from unsaturated solutions by painting their background in a different color; we chose a light green, to suggest that they denote *solved* subgoals. In Figure 8.9, we emphasize systematically the distinction by extending this convention to all saturated solutions.

Generic statuses As can be seen in Figure 8.8, many rules of system B are *generic* over branching operators $\triangleright, \blacktriangleright$, which determine whether a solution is saturated or unsaturated, i.e. its *status*. The challenge is thus to find an iconic counterpart to the symbols $\triangleright, \blacktriangleright$, that fulfills the same function of *meta-variable* ranging over solution statuses. Since we already use the background color to represent the status of concrete solutions, we chose to do the same with abstract ones: each new color other than green will stand for the status of the solution associated to the given location of the canvas. For instance in the f−+↓ rule, the status of the ambient solution where the rule is applied is denoted by a light yellow background, while the status of the solution S enclosed in a red bubble is denoted by a light pink background.

Status changes Last but not least, many rules like i↓ change the status of the ambient solution from unsaturated to saturated: graphically, this means that the background must become green *everywhere*, not only in the portion of the canvas depicted by the rule. At first it might appear as breaking locality, but it should rather be understood as the result of a perfectly local and continuous process: one can imagine a literal *drop* of green paint that soaks a growing portion of the canvas, until it reaches an enclosing bubble — for the consistency of the metaphor, let us say a cut in the papersheet — that stops its progression[17].

17: We will come back to this "*paper cuts*" metaphor, first introduced by C. S. Peirce, in Chapter 9. When saturating the top-level solution — Peirce called it the *sheet of assertion*, the drop expansion process becomes *infinite*. I find it to be a beautiful allegory of the *unreachability* of global, unconditional truth: it is only by being confined to a finite, well-delimited space, that we can affirm unequivocally our certainty. As Wittgenstein famously said at the end of the Tractatus: *"Whereof one cannot speak, thereof one must be silent"*.

Identity

$$\frac{\Gamma\,\langle\rangle\,\Delta}{\Gamma, A \Rightarrow A, \Delta}\ \mathsf{i}{\downarrow} \qquad \frac{\Gamma\,\langle \Rightarrow A;\, A \Rightarrow \rangle\,\Delta}{\Gamma \Rightarrow \Delta}\ \mathsf{i}{\uparrow}$$

Resource

$$\frac{\Gamma \rhd \Delta}{\Gamma, I \rhd \Delta}\ \mathsf{w}{-} \qquad \frac{\Gamma \rhd \Delta}{\Gamma \rhd I, \Delta}\ \mathsf{w}{+}$$

$$\frac{\Gamma, I, I \rhd \Delta}{\Gamma, I \rhd \Delta}\ \mathsf{c}{-} \qquad \frac{\Gamma \rhd I, I, \Delta}{\Gamma \rhd I, \Delta}\ \mathsf{c}{+}$$

Flow

$$\frac{\Gamma\,\langle \mathcal{S};\, \Gamma'\,\langle \mathcal{S}' \rangle\,\Delta';\, S \rangle\,\Delta}{\Gamma\,\langle \mathcal{S};\, \Gamma'\,\langle \mathcal{S}';\, S \rangle\,\Delta' \rangle\,\Delta}\ \mathsf{f}{\uparrow}$$

$$\frac{\Gamma\,\langle \Gamma', I \blacktriangleright \Delta';\, \mathcal{S} \rangle\,\Delta}{\Gamma, I\,\langle \Gamma' \blacktriangleright \Delta';\, \mathcal{S} \rangle\,\Delta}\ \mathsf{f}{-}{\downarrow} \qquad \frac{\Gamma\,\langle \mathcal{S};\, \Gamma' \blacktriangleright I, \Delta' \rangle\,\Delta}{\Gamma\,\langle \mathcal{S};\, \Gamma' \blacktriangleright \Delta' \rangle\, I, \Delta}\ \mathsf{f}{+}{\downarrow}$$

$$\frac{\Gamma \rhd (\Gamma', I \blacktriangleright \Delta'), \Delta}{\Gamma, I \rhd (\Gamma' \blacktriangleright \Delta'), \Delta}\ \mathsf{f}{-}{+}{\downarrow} \qquad \frac{\Gamma, (\Gamma' \blacktriangleright I, \Delta') \rhd \Delta}{\Gamma, (\Gamma' \blacktriangleright \Delta') \rhd I, \Delta}\ \mathsf{f}{+}{-}{\downarrow}$$

$$\frac{\Gamma, I, (\Gamma' \blacktriangleright \Delta') \rhd \Delta}{\Gamma, (\Gamma', I \blacktriangleright \Delta') \rhd \Delta}\ \mathsf{f}{-}{-}{\uparrow} \qquad \frac{\Gamma \rhd (\Gamma' \blacktriangleright \Delta'), I, \Delta}{\Gamma \rhd (\Gamma' \blacktriangleright I, \Delta'), \Delta}\ \mathsf{f}{+}{+}{\uparrow}$$

$$\frac{\Gamma, I \rhd (\Gamma' \blacktriangleright \Delta'), \Delta}{\Gamma \rhd (\Gamma', I \blacktriangleright \Delta'), \Delta}\ \mathsf{f}{-}{+}{\uparrow} \qquad \frac{\Gamma, (\Gamma' \blacktriangleright \Delta') \rhd I, \Delta}{\Gamma, (\Gamma' \blacktriangleright I, \Delta') \rhd \Delta}\ \mathsf{f}{+}{-}{\uparrow}$$

$$\frac{\Gamma, (\Gamma', I \blacktriangleright \Delta') \rhd \Delta}{\Gamma, I, (\Gamma' \blacktriangleright \Delta') \rhd \Delta}\ \mathsf{f}{-}{-}{\downarrow} \qquad \frac{\Gamma \rhd (\Gamma' \blacktriangleright I, \Delta'), \Delta}{\Gamma \rhd (\Gamma' \blacktriangleright \Delta'), I, \Delta}\ \mathsf{f}{+}{+}{\downarrow}$$

Membrane

$$\frac{\Gamma\,\langle \mathcal{S} \rangle\,\Delta}{\Gamma\,\langle \mathcal{S};\, \langle\rangle \rangle\,\Delta}\ \mathsf{p}$$

$$\frac{\Gamma\,\langle\rangle\,\Delta}{\Gamma, (\langle\rangle) \Rightarrow \Delta}\ \mathsf{p}{-} \qquad \frac{\Gamma\,\langle\rangle\,\Delta}{\Gamma \Rightarrow (\langle\rangle), \Delta}\ \mathsf{p}{+}$$

$$\frac{\Gamma\,\langle S \rangle\,\Delta}{\Gamma\,\langle\langle S \rangle\rangle\,\Delta}\ \mathsf{a}$$

$$\frac{\Gamma, S \rhd \Delta}{\Gamma, (\langle S \rangle) \rhd \Delta}\ \mathsf{a}{-} \qquad \frac{\Gamma \rhd S, \Delta}{\Gamma \rhd (\langle S \rangle), \Delta}\ \mathsf{a}{+}$$

Heating

$$\frac{\Gamma \rhd \Delta}{\Gamma, \top \rhd \Delta}\ \top{-} \qquad \frac{\Gamma\,\langle\rangle\,\Delta}{\Gamma \Rightarrow \top, \Delta}\ \top{+}$$

$$\frac{\Gamma\,\langle\rangle\,\Delta}{\Gamma, \bot \Rightarrow \Delta}\ \bot{-} \qquad \frac{\Gamma \rhd \Delta}{\Gamma \rhd \bot, \Delta}\ \bot{+}$$

$$\frac{\Gamma, A, B \rhd \Delta}{\Gamma, A \wedge B \rhd \Delta}\ \wedge{-} \qquad \frac{\Gamma\,\langle \Rightarrow A;\, \Rightarrow B \rangle\,\Delta}{\Gamma \Rightarrow A \wedge B, \Delta}\ \wedge{+}$$

$$\frac{\Gamma\,\langle A \Rightarrow;\, B \Rightarrow \rangle\,\Delta}{\Gamma, A \vee B \Rightarrow \Delta}\ \vee{-} \qquad \frac{\Gamma \rhd A, B, \Delta}{\Gamma \rhd A \vee B, \Delta}\ \vee{+}$$

$$\frac{\Gamma\,\langle \Rightarrow A;\, B \Rightarrow \rangle\,\Delta}{\Gamma, A \supset B \Rightarrow \Delta}\ \supset{-} \qquad \frac{\Gamma \rhd (A \Rightarrow B), \Delta}{\Gamma \rhd A \supset B, \Delta}\ \supset{+}$$

$$\frac{\Gamma, (A \Rightarrow B) \rhd \Delta}{\Gamma, A \subset B \rhd \Delta}\ \subset{-} \qquad \frac{\Gamma\,\langle \Rightarrow A;\, B \Rightarrow \rangle\,\Delta}{\Gamma \Rightarrow A \subset B, \Delta}\ \subset{+}$$

$$\frac{\Gamma, A\{t/x\} \rhd \Delta}{\Gamma, \forall x.A \rhd \Delta}\ \forall{-} \qquad \frac{\Gamma \rhd A, \Delta}{\Gamma \rhd \forall x.A, \Delta}\ \forall{+}$$

$$\frac{\Gamma, A \rhd \Delta}{\Gamma, \exists x.A \rhd \Delta}\ \exists{-} \qquad \frac{\Gamma \rhd A\{t/x\}, \Delta}{\Gamma \rhd \exists x.A, \Delta}\ \exists{+}$$

In the $\forall{+}$ and $\exists{-}$ rules, x is not free in Γ, Δ and $\rhd$.

Figure 8.8.: Sequent-style presentation of system B

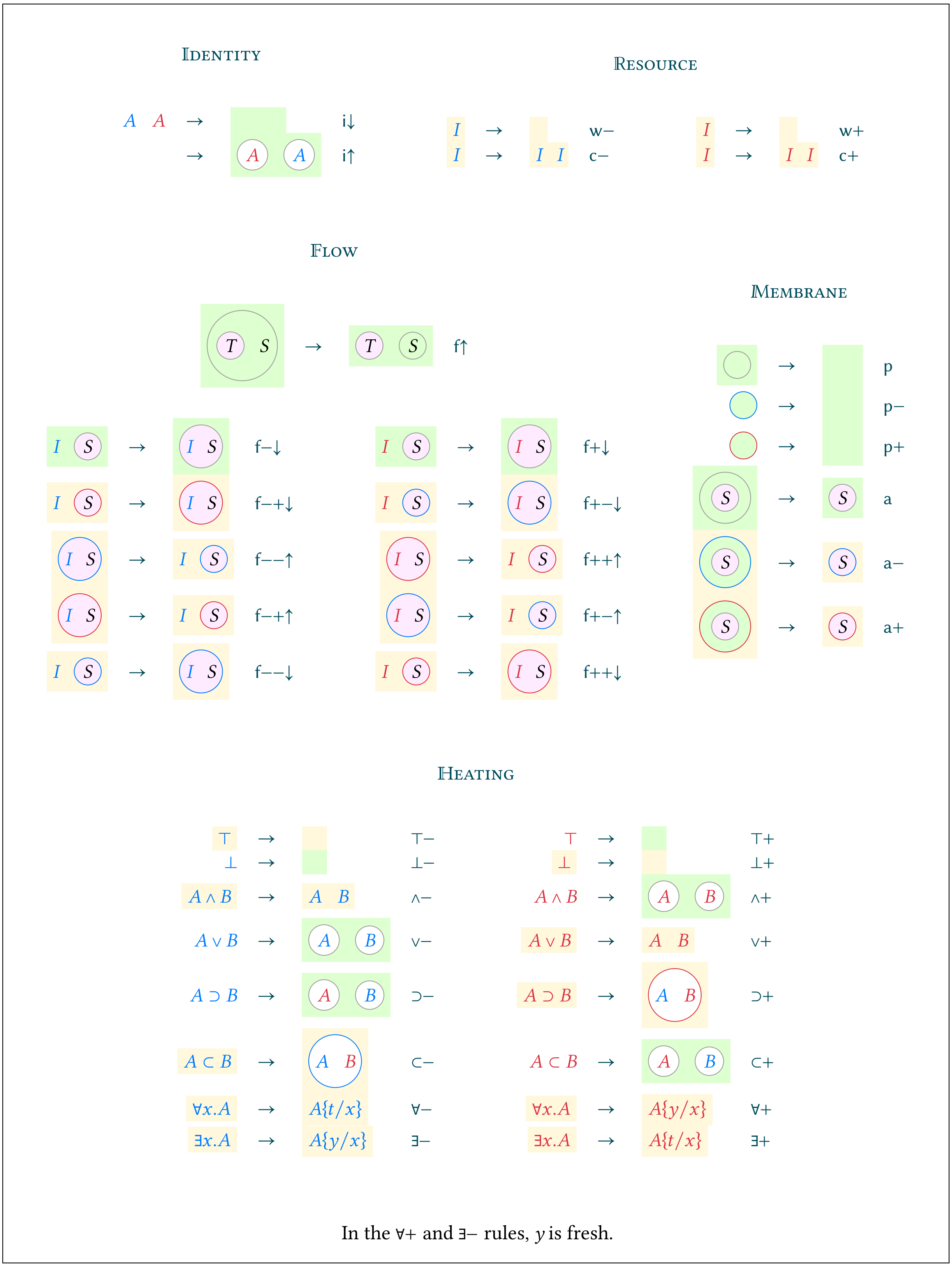


Figure 8.9.: Graphical presentation of system B

8.5.2. Sequent rules

We will now analyze the various groups of rules of system B, by comparing them to those of BJ. We start with the groups already found in sequent calculus:

Identity A first difference, that we will find in most rules of system B, is that we rely on the distributive interpretation of conclusions in solutions. For instance in the i↑ rule, Δ is available potentially in both subgoals, and we do not need to move it manually: this will be the role of the $\mathbb{F}$-rules for red items.

A second difference is that $\mathbb{I}$-rules are not applicable in arbitrary subsolutions, but only *unsaturated* ones. This will also be the case of some $\mathbb{M}$-rules and $\mathbb{H}$-rules. In the case of the i↓ rule, it guarantees its *locality*: if the conclusion was $\Gamma, A \ \langle \mathcal{S} \rangle \ A, \Delta$, then the distributive semantics would entail that all subgoals in $\mathcal{S}$ must be solved at once, despite the fact that they are not directly related to A[18]. As for the i↑ rule, restricting to unsaturated solutions makes the rule *invertible*, without sacrificing locality. This will in fact be the case of all rules that create multiple subgoals.

18: If we were to give up on locality, we could opt for this variant, which gives better *factorizability*. In fact we will precisely do that in Section 8.8.

Resource Here we still have weakening and contraction for negative items (hypotheses), and we also allow them for positive items (conclusions). Note that contrary to the $\mathbb{I}$-rules which apply only to a formula A, $\mathbb{R}$-rules apply to an arbitrary item I, which can either be a formula or a solution. Combined to the fact that the ambient solution can be either unsaturated or saturated, this gives the most general and expressive formulation of the rules. We believe that like in CoS, the atomic version where I is restricted to an atomic formula might be sufficient for completeness.

Heating Like the i↑ rule, the $\bot-$, $\vee-$ and $\supset-$ rules become truly local in system B by letting $\mathbb{F}$-rules handle the distribution of conclusions in subgoals. Together with their dual rules $\top+$, $\wedge+$ and $\supset+$, they constitute the *saturating* $\mathbb{H}$-rules of system B. All other $\mathbb{H}$-rules work in arbitrary solutions just as in BJ. But thanks to the ability to have multiple conclusions (Section 8.2) and positive bubbles (Section 8.3), both the $\vee+$ and $\supset+$ rules are now *invertible*: this was the initial motivation for designing the symmetric bubble calculus.

8.5.3. Bubble rules

We now describe in details the groups of rules that handle specifically the behavior of bubbles:

Flow Compared to BJ where we only had neutral bubbles, the presence of polarized bubbles in system B creates a mini-combinatorial explosion in the number of possible $\mathbb{F}$-rules. Indeed, the general scheme is to consider what *polarities* of items are allowed to flow through bubbles, either inwards or outwards. With p item polarities and b bubble polatities, this makes for a total of $p \times b \times 2$ possible rules. In BJ items consisted only of positive/negative formulas and neutral bubbles ($p = 3$ and $b = 1$), thus we had a total of 6 possible $\mathbb{F}$-rules. It turns out that only the f−↓ rule was necessary, and it is also present in system B. Now with positive and negative bubbles added to the mix ($b = 3$), we get up to a total of 18 possible $\mathbb{F}$-rules in system B. Out of these, 11 were identified as being sound logically, and thus we decided to include all of them in system B.

We have already encountered some of them in Section 8.3: first the f+↓ rule for distributing conclusions in subgoals; but also the f−+↓ and f+−↓ rules, which allow a polarized item to flow *into* a bubble of *opposite* polarity. However to get *cut-free* completeness, we will also need a sort of dual of these rules, f++↑ and f−−↑, which allow a polarized item to flow *out* of a bubble with the *same* polarity. Thus in addition to the duality that *swaps* polarities (f−+↓ versus f+−↓), we have this new duality which *reverses* at the same time the *direction* of the flow, and the *relationship* between polarities (f−+↓ versus f++↑).

Taken together, these 6 rules capture provability in *bi-intuitionistic* logic, as will be demonstrated by the soundness and completeness theorems for system B. By adding any one of the converses to the 4 rules that define the porosity of polarized bubbles (f−+↑, f+−↑, f−−↓, f++↓), the system collapses to *classical* logic. This situation is summarized in Figure 8.10: as in Figure 7.1, green and orange arrows represent respectively valid and invalid moves, but in bi-intuitionistic rather than intuitionistic logic. To recover the latter, one can just ignore all arrows that cross the blue bubble, which are only useful in dual-intuitionistic logic. Then the purple arrows represent moves that are valid only in classical logic. The reader can easily check that there is a total of 18 arrows, and map the green and purple arrows back to the corresponding $\mathbb{F}$-rules of Figure 8.9.

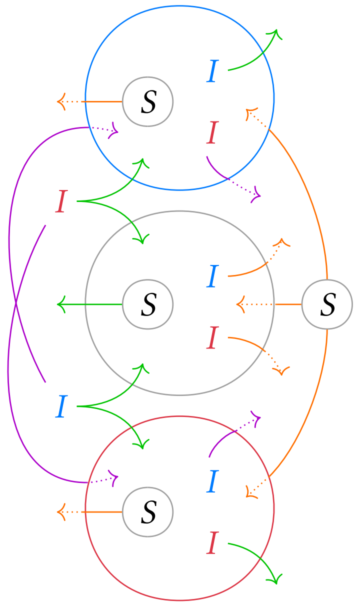


Figure 8.10.: Porosity of bubbles in system B

Remark 8.5.1 Since all polarized items can freely go in and out of polarized bubbles in classical logic, the latter are useless. In fact, one could restrict the syntax of solutions to neutral bubbles and only one polarity of formulas, say conclusions. This corresponds to the possibility of having one-sided formulations of sequent calculi for classical logic, by restricting negation to atomic formulas and extending it to arbitrary formulas through De Morgan dualities[19].

19: See for instance the one-sided sequent calculus in [94].

In their graphical representation, the bi-intuitionistic $\mathbb{F}$-rules of system B are equivalent to the three following *topological laws*, that we call the $\mathbb{F}$-*laws*[20]:

20: Hopefully, those are not *flaws* of our *flow* rules, but rather the opposite…

Fact 8.5.1 ($\mathbb{F}$-laws)

1. Polarized bubbles trap items with a different polarity, and repel items with the same polarity.
2. Neutral bubbles trap polarized items, and repel neutral items.
3. Polarized bubbles both trap and repel neutral bubbles.

In Figure 8.10, the ability of bubbles to *trap* or *repel* items correspond respectively to *outward* and *inward* orange arrows. $\mathbb{F}$-laws are thus the “negative” counterpart — in the grammatical sense — of $\mathbb{F}$-rules, represented by green arrows. The fact that purple arrows are demoted to orange arrows in bi-intuitionistic logic, can be interpreted as resulting from their violation of the first $\mathbb{F}$-law. The second and third $\mathbb{F}$-laws characterize the behavior of neutral bubbles, and are respected by all rules of system B.

In particular, they suggest the addition of a new $\mathbb{F}$-rule f↑, which allows to move neutral bubbles out of other neutral bubbles. When looking at it as a graphical rewriting rule in Figure 8.9, it can be seen as the act of *abstracting* the subgoal T from its parent subgoal S, since the hypotheses and conclusions of S cannot be brought to interact with those of T anymore. More generally in bi-intuitionistic logic, all $\mathbb{F}$-rules can be understood as *abstraction* moves, that strengthen the goal by moving irreversibly an item I out of its subgoal S. In the case of outward rules (whose name ends with ↑), I is brought closer to the *root* of the proof tree; and in the case of inward rules (whose name ends with ↓), I is brought closer to the *leaves* of the proof tree.

It would be interesting to try to formalize $\mathbb{F}$-laws, and more generally the graphical presentation of system B, with the rigorous tools of mathematical topology. This has been done for instance in [25] for the existential graphs of C. S. Peirce (see Chapter 10).

[25]: Brady et al. (2000), ‘A categorical interpretation of C.S. Peirce’s propositional logic Alpha’

Membrane We still have the popping rule p of BJ, which is now restricted to saturated empty bubbles. We add two popping rules p− and p+ for popping respectively negative and positive saturated empty bubbles. Like the i↓ rule, these have the effect of saturating the ambient solution, and for the same reasons we thus restrict them to unsaturated ambient solutions.

The novelty compared to BJ is that we also add so-called *absorption rules* {a, a−, a+} for membranes. These rules state that when a bubble contains only a single neutral bubble, the membrane of the latter can be absorbed into the membrane of the former. This is mainly useful when one wants to apply an outward $\mathbb{F}$-rule to an item that has the same polarity as the outer bubble, as witnessed by the use of the a+ rule in the proof of Uustalu’s formula in Figure 8.11. This formula was first introduced in [204] as a counter-example to the cut-elimination theorem of Rauszer’s sequent calculus for bi-intuitionistic logic [209], and our initial motivation for introducing absorption rules was precisely to provide a cut-free proof of this formula in system B.

$$
\begin{array}{cl}
\langle\rangle & \\ \hline
\langle\langle\rangle\rangle & \mathsf{p} \\ \hline
\langle\langle\rangle ; \langle\rangle\rangle & \mathsf{p} \\ \hline
\langle\langle\rangle ; q \Rightarrow q\rangle & \mathsf{i}\downarrow \\ \hline
\langle p \Rightarrow p ; q \Rightarrow q\rangle & \mathsf{i}\downarrow \\ \hline
\langle p \Rightarrow p ; q \Rightarrow \rangle\; q & \mathsf{f}{+}\downarrow \\ \hline
p\,\langle \Rightarrow p ; q \Rightarrow \rangle\; q & \mathsf{f}{-}\downarrow \\ \hline
p \Rightarrow q, p \subset q & \subset+ \\ \hline
p \Rightarrow q, p \subset q, (\Rightarrow) & \mathsf{w}+ \\ \hline
p \Rightarrow q, (\Rightarrow p \subset q) & \mathsf{f}{+}{+}\downarrow \\ \hline
p \Rightarrow q, (\langle \Rightarrow p \subset q\rangle) & \mathsf{a}+ \\ \hline
p \Rightarrow q, (\langle \Rightarrow p \subset q ; \langle\rangle\rangle) & \mathsf{p} \\ \hline
p \Rightarrow q, (\langle \Rightarrow p \subset q ; r \Rightarrow r\rangle) & \mathsf{i}\downarrow \\ \hline
p \Rightarrow q, (r\,\langle \Rightarrow p \subset q ; \Rightarrow r\rangle) & \mathsf{f}{-}\downarrow \\ \hline
p \Rightarrow q, (r \Rightarrow ((p \subset q) \wedge r)) & \wedge+ \\ \hline
p \Rightarrow q, r \supset ((p \subset q) \wedge r) & \supset+
\end{array}
$$

Figure 8.11.: A proof of Uustalu’s formula in system B

[204]: Pinto et al. (2009), ‘Proof Search and Counter-Model Construction for Bi-intuitionistic Propositional Logic with Labelled Sequents’

[209]: Rauszer (1974), ‘A Formalization of the Propositional Calculus of H-B Logic’

Later, we realized that there is an interesting *symmetry* at play between popping rules and absorption rules. As mentioned in Section 7.4, popping rules can be understood as resulting from a process of *contraction* of membranes into a single point. Dually, absorption rules can be seen as the result of a process of *expansion* of the inner bubble towards the outer bubble. While contraction gets stuck on polarized items because they cannot cross neutral membranes outwards, expansion gets stuck on neutral items because they cannot cross neutral membranes inwards. Thus there is a very natural interplay between $\mathbb{M}$-rules and $\mathbb{F}$-laws.

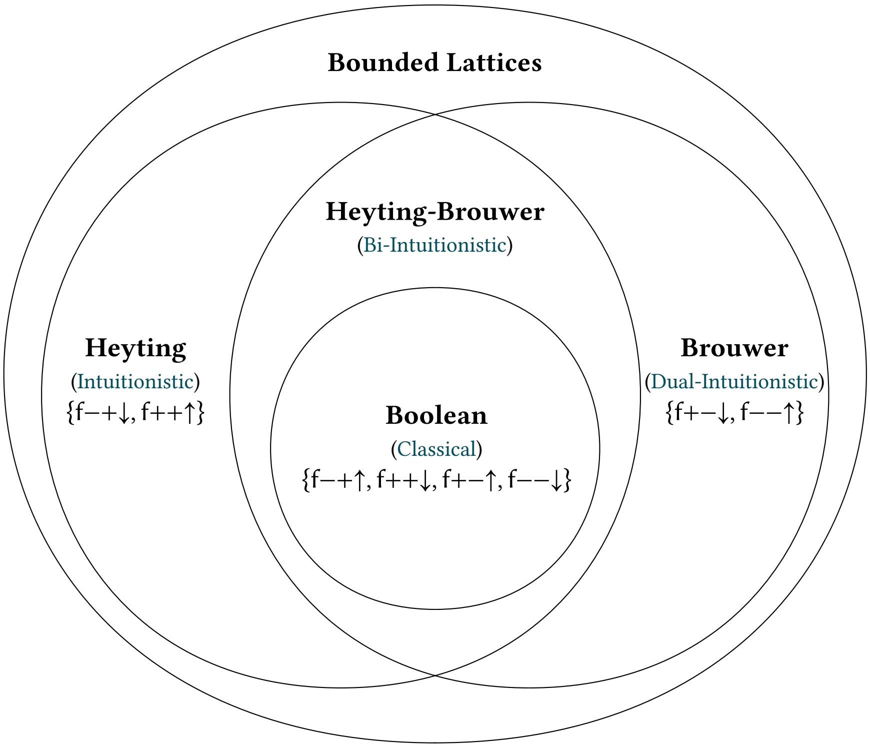


Figure 8.12.: Relationship between the various algebras interpreting system B

8.6. Soundness

8.6.1. Heyting and Brouwer algebras

We are now going to prove the soundness of system B with respect to various classes of *algebras*. While the full system is classical and thus sound only in *Boolean* algebras, most rules are sound in larger classes of algebras, namely: *Heyting* algebras for intuitionistic logic, *Brouwer* algebras for dual-intuitionistic logic, and *Heyting-Brouwer* algebras for bi-intuitionistic logic. These 4 classes are all instances of *bounded lattices*, and their relationship is summarized in the Venn diagram of Figure 8.12.

First we recall the definitions of the various algebras:

Definition 8.6.1 (Bounded lattice) *A* bounded lattice *is a structure* $(\mathscr{A}, \leq, \top, \bot, \wedge, \vee)$ *such that:*

- $(\mathscr{A}, \leq)$ *is a partial order, i.e. for every* $a, b, c \in \mathscr{A}$ *we have:*
  - $a \leq a$;
  - *if* $a \leq b$ *and* $b \leq a$ *then* $a = b$;
  - *if* $a \leq b$ *and* $b \leq c$ *then* $a \leq c$.
- $\bot$ *and* $\top$ *are respectively the smallest and greatest elements of* $(\mathscr{A}, \leq)$, *i.e. for every* $a \in \mathscr{A}$ *we have* $\bot \leq a$ *and* $a \leq \top$;
- *For every pair of elements* $a, b \in \mathscr{A}$, $a \vee b$ *is their join (least upper bound)*

and $a \wedge b$ their meet (greatest lower bound), that is:

- $a \leq a \vee b$, $b \leq a \vee b$ *and* $a \vee b \leq c$ *for all* $c \in \mathscr{A}$ *s.t.* $a \leq c$ *and* $b \leq c$;
- $a \wedge b \leq a$, $a \wedge b \leq b$ *and* $c \leq a \wedge b$ *for all* $c \in \mathscr{A}$ *s.t.* $c \leq a$ *and* $c \leq b$.

Remark 8.6.1 As mentioned in the introduction, we only conjecture the soundness of rules for quantifiers: this would require considering *complete* lattices, i.e. with meets and joins for arbitrary sets rather than just pairs[21].

21: See for instance [79, Section 4] for a concise treatment of the soundness and completeness of intuitionistic and classical natural deduction for first-order logic with respect to algebraic semantics.

As the notation strongly suggests, the greatest and smallest elements $\top$ and $\bot$ will model respectively truth and absurdity, while the meet $\wedge$ and join $\vee$ will model conjunction and disjunction. In fact the conditions of Definition 8.6.1 are very close to the rules of natural deduction for these connectives, by replacing the sequent operator $\Rightarrow$ with the partial order relation $\leq$. The same idea can be applied to the implication connective, and adding a corresponding exponential operation $\supset$ indeed gives the definition of a Heyting algebra:

Definition 8.6.2 (Heyting algebra) *A Heyting algebra is a structure* $(\mathscr{A}, \leq, \top, \bot, \wedge, \vee, \supset)$ *such that* $(\mathscr{A}, \leq, \top, \bot, \wedge, \vee)$ *is a bounded lattice and for every pair* $a, b \in \mathscr{A}$, *the exponential* $a \supset b$ *is the greatest element of the set* $\{c \in \mathscr{A} \mid c \wedge a \leq b\}$. *That is,* $(a \supset b) \wedge a \leq b$ *and* $c \leq a \supset b$ *for all* $c \in \mathscr{A}$ *s.t.* $c \wedge a \leq b$.

By dualizing this definition, we get a co-exponential operation $\subset$ that models the exclusion connective, and thus dual-intuitionistic logic in so-called Brouwer algebras:

Definition 8.6.3 (Brouwer algebra) *A Brouwer algebra is a structure* $(\mathscr{A}, \leq, \top, \bot, \wedge, \vee, \subset)$ *such that* $(\mathscr{A}, \leq, \top, \bot, \wedge, \vee)$ *is a bounded lattice and for every pair* $a, b \in \mathscr{A}$, *the co-exponential* $a \subset b$ *is the smallest element of the set* $\{c \in \mathscr{A} \mid b \leq a \vee c\}$. *That is,* $b \leq a \vee (b \subset a)$ *and* $b \subset a \leq c$ *for all* $c \in \mathscr{A}$ *s.t.* $b \leq a \vee c$.

Then we can model bi-intuitionistic logic, which comprises both implication and exclusion, by just taking pairs of a Heyting algebra and a Brouwer algebra on the same bounded lattice:

Definition 8.6.4 (Heyting-Brouwer algebra) *A Heyting-Brouwer algebra is a structure* $(\mathscr{A}, \leq, \top, \bot, \wedge, \vee, \supset, \subset)$ *such that* $(\mathscr{A}, \leq, \top, \bot, \wedge, \vee, \supset)$ *is a Heyting algebra and* $(\mathscr{A}, \leq, \top, \bot, \wedge, \vee, \subset)$ *is a Brouwer algebra.*

Finally, we recover classical logic by collapsing exponentials and co-exponentials to their classical definitions, giving a characterization of Boolean algebras:

Definition 8.6.5 *A Boolean algebra is a Heyting-Brouwer algebra* $(\mathscr{A}, \leq, \top, \bot, \wedge, \vee, \supset, \subset)$ *such that for every* $a, b \in \mathscr{A}$, $a \supset b = (\top \subset a) \vee b$ *and* $a \subset b = a \wedge (b \supset \bot)$.

Remark 8.6.2 Definition 8.6.5 can be shown equivalent to more usual definitions of Boolean algebras, that are based only on lattice operations and a primitive complement operation modelling negation; but including the proof here would lead us out of the scope of this chapter.

In the rest of this chapter, we will freely assimilate formulas with their interpretation in the various algebras. Indeed, since we only consider the abstract classes of all algebras and never deal with particular instances, they will stand in perfect bijection.

Definition 8.6.6 (Semantic entailment) *We write* $A \leq_{\mathcal{X}} B$ *(resp.* $A \simeq_{\mathcal{X}} B$*) to express that* $A \leq B$ *(resp.* $A \leq B$ *and* $B \leq A$*) in every algebra of the class* $\mathcal{X}$*. More precisely,* $\mathcal{X}$ *can be one of* $\mathcal{L}$, $\mathcal{H}$, $\mathcal{B}$, $\mathcal{HB}$ *or* $\mathcal{C}$*, which stand respectively for bounded lattices, Heyting, Brouwer, Heyting-Brouwer and Boolean algebras. We write* $A \leq B$ *(resp.* $A \simeq B$*) as a shorthand for* $A \leq_{\mathcal{H}} B$ *(resp.* $A \simeq_{\mathcal{H}} B$*).*

8.6.2. Duality

We now prove a number of lemmas that characterize *duality* both semantically, typically between Heyting algebras and Brouwer algebras, and syntactically in the rules of system B. This will be useful later on to shorten some proofs.

Definition 8.6.7 (Dual formula) *The* dual formula $A^\dagger$ *of a formula* A *is defined recursively as follows:*

$$
\begin{aligned}
a^\dagger &= a & & \\
\top^\dagger &= \bot & \bot^\dagger &= \top \\
(A \wedge B)^\dagger &= A^\dagger \vee B^\dagger & (A \vee B)^\dagger &= A^\dagger \wedge B^\dagger \\
(A \supset B)^\dagger &= B^\dagger \subset A^\dagger & (A \subset B)^\dagger &= B^\dagger \supset A^\dagger
\end{aligned}
$$

Fact 8.6.1 (Duality)

- $A \leq_{\mathcal{H}} B$ if and only if $B^\dagger \leq_{\mathcal{B}} A^\dagger$
- $A \leq_{\mathcal{B}} B$ if and only if $B^\dagger \leq_{\mathcal{H}} A^\dagger$
- $A \leq_{\mathcal{X}} B$ if and only if $B^\dagger \leq_{\mathcal{X}} A^\dagger$ when $\mathcal{X} \in \{\mathcal{HB}, \mathcal{C}\}$.

We omit the proof of Fact 8.6.1, but this can easily be obtained from the soundness and completeness of a symmetric sequent calculus for bi-intuitionistic logic; see for instance [212, Lemma 2].

[212]: Restall (1997), *Extending Intuitionistic Logic with Subtraction*

Definition 8.6.8 (Dual solution) *The* dual solution $S^\dagger$ *of a* solution S *is defined mutually recursively as follows:*

$$(\Gamma \rhd \Delta)^\dagger = \Delta^\dagger \rhd^\dagger \Gamma^\dagger \qquad (S_1 ; \dots ; S_n)^\dagger = S_1^\dagger ; \dots ; S_n^\dagger$$
$$A^\dagger = A^\dagger \qquad (I_1, \dots, I_n)^\dagger = I_1^\dagger, \dots, I_n^\dagger$$
$$\Rightarrow^\dagger = \Rightarrow \qquad \langle \mathcal{S} \rangle^\dagger = \langle \mathcal{S}^\dagger \rangle$$

For contexts*, the* hole *is self-dual:* $\Box^\dagger = \Box$*. This entails in particular that* $S^\dagger\boxed{T^\dagger} = S\boxed{T}^\dagger$.

Graphically, the dual of a solution S is S where the colors of items have been swapped — i.e. blue items become red and red items become blue — and formulas have been dualized (Definition 8.6.7).

Definition 8.6.9 *The* depth $|I|$ *of an* item I *is defined recursively as follows:*

$$|A| = 0$$
$$|\Gamma \Rightarrow \Delta| = 1 + \max_{J \in \Gamma \cup \Delta} |J|$$
$$|\Gamma \langle \mathcal{S} \rangle \Delta| = 1 + \max_{J \in \Gamma \cup \mathcal{S} \cup \Delta} |J|$$

Lemma 8.6.1 (Involutivity) $I^{\dagger\dagger} = I$.

Proof. By induction on $|I|$.

Formula Suppose $I = A$. Then we conclude by a straightforward induction on A.

Unsaturated solution Suppose $I = \Gamma \Rightarrow \Delta$. Then by definition we have $(\Gamma \Rightarrow \Delta^\dagger)^\dagger = (\Delta^\dagger \Rightarrow \Gamma^\dagger)^\dagger = \Gamma^{\dagger\dagger} \Rightarrow \Delta^{\dagger\dagger}$, and we conclude by IH.

Saturated solution Suppose $I = \Gamma \langle \mathcal{S} \rangle \Delta$. Then by definition we have $(\Gamma \langle \mathcal{S} \rangle \Delta^\dagger)^\dagger = (\Delta^\dagger \langle \mathcal{S}^\dagger \rangle \Gamma^\dagger)^\dagger = \Gamma^{\dagger\dagger} \langle \mathcal{S}^{\dagger\dagger} \rangle \Delta^{\dagger\dagger}$, and we conclude by IH.

□

Lemma 8.6.2 (Shallow rule duality) *If* $S \rightharpoonup T$ *then* $S^\dagger \rightharpoonup T^\dagger$.

Proof. There is a bijection among the rules of system B, that matches each rule $r : S \to T$ to its dual $r^\dagger : S^\dagger \to T^\dagger$. By involutivity (Lemma 8.6.1), this bijection is self-inverse: $r^{\dagger\dagger} = r$. It is most easily observed in the graphical presentation of the rules (Figure 8.9), where looking for the dual rule boils down to swapping red and blue (and mirroring logical connectives). The mapping goes as follows:

$$\begin{array}{ccc} \mathsf{i}{\downarrow} & \leftrightarrow & \mathsf{i}{\downarrow} \\ \mathsf{i}{\uparrow} & \leftrightarrow & \mathsf{i}{\uparrow} \end{array} \qquad \begin{array}{ccc} \mathsf{w}- & \leftrightarrow & \mathsf{w}+ \\ \mathsf{c}- & \leftrightarrow & \mathsf{c}+ \end{array}$$

$$\begin{array}{ccc} \mathsf{f}- & \leftrightarrow & \mathsf{f}+ \\ \mathsf{f}{-}{+}{\downarrow} & \leftrightarrow & \mathsf{f}{+}{-}{\downarrow} \\ \mathsf{f}{-}{-}{\uparrow} & \leftrightarrow & \mathsf{f}{+}{+}{\uparrow} \\ \mathsf{f}{-}{+}{\uparrow} & \leftrightarrow & \mathsf{f}{+}{-}{\uparrow} \\ \mathsf{f}{-}{-}{\downarrow} & \leftrightarrow & \mathsf{f}{+}{+}{\downarrow} \end{array} \qquad \begin{array}{ccc} \mathsf{p} & \leftrightarrow & \mathsf{p} \\ \mathsf{p}- & \leftrightarrow & \mathsf{p}+ \\ \mathsf{a} & \leftrightarrow & \mathsf{a} \\ \mathsf{a}- & \leftrightarrow & \mathsf{a}+ \end{array}$$

$$\begin{array}{ccc} \top- & \leftrightarrow & \bot+ \\ \bot- & \leftrightarrow & \top+ \\ \wedge- & \leftrightarrow & \vee+ \\ \vee- & \leftrightarrow & \wedge+ \\ \supset- & \leftrightarrow & \subset+ \\ \subset- & \leftrightarrow & \supset+ \\ \forall- & \leftrightarrow & \exists+ \\ \exists- & \leftrightarrow & \forall+ \end{array}$$

Notice that some rules are self-dual, namely the identity rules $\mathsf{i}{\downarrow}$ and $\mathsf{i}{\uparrow}$, and the membrane rules p and a. □

Lemma 8.6.3 (Rule duality) *If $S \to T$ then $S^\dagger \to T^\dagger$.*

Proof. Let $U\square$, S_0 and T_0 such that $S = U\boxed{S_0}$, $T = U\boxed{T_0}$ and $S_0 \rightharpoonup T_0$. By Lemma 8.6.2 we have $S_0^\dagger \rightharpoonup T_0^\dagger$, and thus $U^\dagger\boxed{S_0^\dagger} \to U^\dagger\boxed{T_0^\dagger}$, or equivalently $U\boxed{S_0}^\dagger \to U\boxed{T_0}^\dagger$. □

Lemma 8.6.4 (Interpretation duality) $[\![I]\!]^{+\dagger} = [\![I^\dagger]\!]^-$ *and* $[\![I]\!]^{-\dagger} = [\![I^\dagger]\!]^+$.

Proof. By a straightforward induction on $|I|$. □

Lemma 8.6.5 $[\![S^\dagger]\!]^+ \leq_{\mathcal{X}} [\![T^\dagger]\!]^+$ *if and only if* $[\![T]\!]^- \leq_{\mathcal{X}} [\![S]\!]^-$ *when* $\mathcal{X} \in \{\mathcal{HB}, \mathcal{C}\}$.

Proof. By duality (Fact 8.6.1) we have $[\![T^\dagger]\!]^{+\dagger} \leq_{\mathcal{X}} [\![S^\dagger]\!]^{+\dagger}$, and then by Lemma 8.6.4 $[\![T^{\dagger\dagger}]\!]^- \leq_{\mathcal{X}} [\![S^{\dagger\dagger}]\!]^-$. We conclude by involutivity (Lemma 8.6.1). □

8.6.3. Shallow soundness

In the following we give a number of (in)equalities that hold in the various classes of algebras. They can easily be checked by building derivations in an adequate sequent calculus.

Fact 8.6.2 (Commutativity) $A \vee B \simeq_{\mathcal{L}} B \vee A$ and $A \wedge B \simeq_{\mathcal{L}} B \wedge A$.

Fact 8.6.3 (Idempotency) $A \vee A \simeq_{\mathcal{L}} A$ and $A \wedge A \simeq_{\mathcal{L}} A$.

Fact 8.6.4 (Currying)

$$A \supset (B \supset C) \simeq (A \wedge B) \supset C$$
$$(A \subset B) \subset C \simeq_{\mathcal{B}} A \subset (B \vee C)$$

Fact 8.6.5 (Distributivity)

$$A \wedge (B \vee C) \simeq_{\mathcal{L}} (A \wedge B) \vee (A \wedge C)$$
$$A \vee (B \wedge C) \simeq_{\mathcal{L}} (A \vee B) \wedge (A \vee C)$$
$$A \supset B \wedge C \simeq (A \supset B) \wedge (A \supset C)$$
$$A \vee B \supset C \simeq (A \supset B) \wedge (A \supset C)$$
$$A \vee B \subset C \simeq_{\mathcal{B}} (A \subset B) \vee (A \subset C)$$
$$A \subset B \wedge C \simeq_{\mathcal{B}} (A \subset B) \vee (A \subset C)$$

Fact 8.6.6 (Weak distributivity)

$$(A \supset B) \vee C \leq A \supset (B \vee C)$$
$$A \supset (B \vee C) \leq_{\mathcal{C}} (A \supset B) \vee C$$
$$(A \wedge B) \subset C \leq_{\mathcal{B}} A \wedge (B \subset C)$$
$$A \wedge (B \subset C) \leq_{\mathcal{C}} (A \wedge B) \subset C$$

Fact 8.6.7

$$(A \vee B) \wedge (C \supset D) \leq (A \supset C) \supset (B \vee D)$$
$$(A \vee B) \wedge (C \supset D) \leq_{\mathcal{HB}} (A \subset C) \vee (B \vee D)$$
$$(A \vee B) \wedge (A \supset B) \simeq B$$

Fact 8.6.8 $(A \subset B) \supset C \leq_{\mathcal{HB}} A \supset B \vee C$.

The following definition will be used pervasively to reason by induction on the tree structure induced by branching operators:

Definition 8.6.10 *The depth $|\rhd|$ of a branching operator $\rhd$ is defined recursively as follows:*

$$|\Rightarrow| = 0$$
$$|\langle \mathcal{S} \rangle| = 1 + \max_{S \in \mathcal{S}} |S|$$

Now we can prove a few lemmas that generalize some semantic (in)equalities to the interpretation of solutions with arbitrary branching operators. All detailed proofs are available in appendix (Section A.1).

Lemma 8.6.6 (Generalized weakening) $[\![S]\!]^+ \leq [\![S \uplus (\Gamma \Rightarrow \Delta)]\!]^+$.

Proof. By induction on $|\rhd|$, with $S = \Gamma' \rhd \Delta'$. ☐

Lemma 8.6.7 (Generalized contraction) $[\![S \uplus (\Rightarrow I, I)]\!]^+ \simeq [\![S \uplus (\Rightarrow I)]\!]^+$ *and* $[\![S \uplus (I, I \Rightarrow)]\!]^+ \simeq [\![S \uplus (I \Rightarrow)]\!]^+$.

Proof. By induction on $|\rhd|$, with $S = \Gamma \rhd \Delta$. ☐

Lemma 8.6.8 (Generalized weak distributivity)

$$[\![\Gamma \rhd \Delta]\!]^+ \vee [\![I]\!]^+ \leq [\![\Gamma \rhd I, \Delta]\!]^+ \tag{8.4}$$

$$[\![\Gamma \rhd I, \Delta]\!]^+ \leq_{\mathcal{C}} [\![\Gamma \rhd \Delta]\!]^+ \vee [\![I]\!]^+ \tag{8.5}$$

$$[\![\Gamma, I \rhd \Delta]\!]^- \leq_{\mathcal{B}} [\![I]\!]^- \wedge [\![\Gamma \rhd \Delta]\!]^- \tag{8.6}$$

$$[\![I]\!]^- \wedge [\![\Gamma \rhd \Delta]\!]^- \leq_{\mathcal{C}} [\![\Gamma, I \rhd \Delta]\!]^- \tag{8.7}$$

Proof. (8.4) holds by induction on $|\rhd|$, using the corresponding inequality from Fact 8.6.6. The proof of (8.5) is the same, except that we use the converse inequality of Fact 8.6.6 that holds in Boolean algebras. (8.6) and (8.7) hold by duality from (8.4) and (8.5). ☐

Lemma 8.6.9 (Generalized currying)

$$[\![\Gamma, I \rhd \Delta]\!]^+ \simeq [\![I]\!]^- \supset [\![\Gamma \rhd \Delta]\!]^+ \tag{8.8}$$

$$[\![\Gamma \rhd I, \Delta]\!]^- \simeq_{\mathcal{B}} [\![\Gamma \rhd \Delta]\!]^- \subset [\![I]\!]^+ \tag{8.9}$$

Proof. (8.8) holds by induction on $|\rhd|$, and (8.9) by duality. ☐

Lastly, we mention a technical property of the rules that will be necessary for the final proof of soundness to go through:

Fact 8.6.9 (Top-level genericity) If $S \rightharpoonup T$, then $S \uplus (\Gamma \Rightarrow \Delta) \rightharpoonup T \uplus (\Gamma \Rightarrow \Delta)$.

All the previous facts and lemmas can now be used to prove *shallow soundness*, i.e. that the interpretation of each rule of system B maps to an (in)equality in some class of algebras:

Lemma 8.6.10 (Shallow soundness) *If $S \rightharpoonup T$ then $[\![T \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \leq_{\mathcal{C}} [\![S \uplus (\Gamma \Rightarrow \Delta)]\!]^+$.*

Proof. $S \rightharpoonup T$ implies $[\![T]\!]^+ \leq_{\mathcal{C}} [\![S]\!]^+$, which is shown by inspection of each rule of system B (see Section A.1). That we can mix an arbitrary top-level context $\Gamma \Rightarrow \Delta$ into S and T follows from Fact 8.6.9. □

Since some rules only hold classically, the statement for the full system is relative to Boolean algebras. But from the detailed proof in Section A.1, we can identify two fragments $\mathsf{B}_{\mathcal{H}}$ and $\mathsf{B}_{\mathcal{HB}}$ of system B that are sound respectively for Heyting algebras and Heyting-Brouwer algebras:

Corollary 8.6.11 *Let*

$$\mathsf{B}_{\mathcal{HB}} \triangleq \mathsf{B} \setminus \{\mathsf{f}{-}{+}{\uparrow}, \mathsf{f}{+}{+}{\downarrow}, \mathsf{f}{+}{-}{\uparrow}, \mathsf{f}{-}{-}{\downarrow}\}$$
$$\mathsf{B}_{\mathcal{H}} \triangleq \mathsf{B}_{\mathcal{HB}} \setminus \{\mathsf{f}{+}{-}{\downarrow}, \mathsf{f}{-}{-}{\uparrow}, \subset{-}, \subset{+}\}$$

Then we have:

- $S \rightharpoonup_{\mathsf{B}_{\mathcal{H}}} T$ *implies* $[\![T]\!]^+ \leq [\![S]\!]^+$
- $S \rightharpoonup_{\mathsf{B}_{\mathcal{HB}}} T$ *implies* $[\![T]\!]^+ \leq_{\mathcal{HB}} [\![S]\!]^+$

In order to get the last missing fragment $\mathsf{B}_{\mathcal{B}}$ sound with respect to Brouwer algebras, we need dual lemmas that are relative to the negative interpretation $[\![-]\!]^-$ instead of the positive interpretation $[\![-]\!]^+$, since implication is replaced by exclusion. To avoid verbosity, we only formulate the main lemma, and assume that its proof will go through mechanically:

Lemma 8.6.12 (Shallow co-soundness) *If $S \rightharpoonup T$ then $[\![S \uplus (\Gamma \Rightarrow \Delta)]\!]^- \leq_{\mathcal{C}} [\![T \uplus (\Gamma \Rightarrow \Delta)]\!]^-$.*

Then from the (assumed) proof of Lemma 8.6.12 we get:

Corollary 8.6.13 *Let $\mathsf{B}_{\mathcal{B}} \triangleq \mathsf{B}_{\mathcal{HB}} \setminus \{\mathsf{f}{-}{+}{\downarrow}, \mathsf{f}{+}{+}{\uparrow}, \supset{-}, \supset{+}\}$. Then $S \rightharpoonup_{\mathsf{B}_{\mathcal{B}}} T$ implies $[\![S]\!]^- \leq_{\mathcal{B}} [\![T]\!]^-$.*

The full situation is summarized in Figure 8.12.

8.6.4. Contextual soundness

Lemma 8.6.14 (Functoriality) *Let $\mathcal{X} \in \{\mathcal{H}, \mathcal{HB}, \mathcal{C}\}$.*

- $[\![I]\!]^+ \leq_{\mathcal{X}} [\![J]\!]^+$ *implies* $[\![(\Rightarrow I) \uplus S]\!]^+ \leq_{\mathcal{X}} [\![(\Rightarrow J) \uplus S]\!]^+$
- $[\![J]\!]^- \leq_{\mathcal{X}} [\![I]\!]^-$ *implies* $[\![(I \Rightarrow) \uplus S]\!]^+ \leq_{\mathcal{X}} [\![(J \Rightarrow) \uplus S]\!]^+$

Proof. Let $S = \Gamma \rhd \Delta$. We proceed by induction on $|\rhd|$.

Base case Suppose $|\rhd| = 0$. Then $\rhd = \Rightarrow$, and we have

$$\begin{array}{rcll}
[\![(\Rightarrow I) \uplus S]\!]^+ & = & [\![\Gamma \Rightarrow I, \Delta]\!]^+ & \\
& = & [\![\Gamma]\!]^- \supset [\![I]\!]^+ \vee [\![\Delta]\!]^+ & \\
& \leq_{\mathcal{X}} & [\![\Gamma]\!]^- \supset [\![J]\!]^+ \vee [\![\Delta]\!]^+ & \text{(Hypothesis)} \\
& = & [\![\Gamma \Rightarrow J, \Delta]\!]^+ & \\
& = & [\![(\Rightarrow J) \uplus S]\!]^+ &
\end{array}$$

$$\begin{array}{rcll}
[\![(I \Rightarrow) \uplus S]\!]^+ & = & [\![\Gamma, I \Rightarrow \Delta]\!]^+ & \\
& = & [\![\Gamma]\!]^- \wedge [\![I]\!]^- \supset [\![\Delta]\!]^+ & \\
& \leq_{\mathcal{X}} & [\![\Gamma]\!]^- \wedge [\![J]\!]^- \supset [\![\Delta]\!]^+ & \text{(Hypothesis)} \\
& = & [\![\Gamma, J \Rightarrow \Delta]\!]^+ & \\
& = & [\![(J \Rightarrow) \uplus S]\!]^+ &
\end{array}$$

Recursive case Suppose $|\rhd| > 0$. Then $\rhd = \langle \mathcal{S} \rangle$, and for all $S_0 = \Gamma_0 \blacktriangleright \Delta_0 \in \mathcal{S}$ we have that $|\blacktriangleright| < |\rhd|$. Thus we have

$$\begin{array}{rcll}
[\![(\Rightarrow I) \uplus S]\!]^+ & = & [\![\Gamma \langle \mathcal{S} \rangle I, \Delta]\!]^+ & \\
& = & \bigwedge_{S_0 \in \mathcal{S}} [\![(\Gamma \Rightarrow I, \Delta) \uplus S_0]\!]^+ & \\
& = & \bigwedge_{S_0 \in \mathcal{S}} [\![(\Rightarrow I) \uplus ((\Gamma \Rightarrow \Delta) \uplus S_0)]\!]^+ & \\
& \leq_{\mathcal{X}} & \bigwedge_{S_0 \in \mathcal{S}} [\![(\Rightarrow J) \uplus ((\Gamma \Rightarrow \Delta) \uplus S_0)]\!]^+ & \text{(IH)} \\
& = & \bigwedge_{S_0 \in \mathcal{S}} [\![(\Gamma \Rightarrow J, \Delta) \uplus S_0]\!]^+ & \\
& = & [\![\Gamma \langle \mathcal{S} \rangle J, \Delta]\!]^+ & \\
& = & [\![(\Rightarrow J) \uplus S]\!]^+ &
\end{array}$$

$$\begin{array}{rcll}
[\![(I \Rightarrow) \uplus S]\!]^+ & = & [\![\Gamma, I \langle \mathcal{S} \rangle \Delta]\!]^+ & \\
& = & \bigwedge_{S_0 \in \mathcal{S}} [\![(\Gamma, I \Rightarrow \Delta) \uplus S_0]\!]^+ & \\
& = & \bigwedge_{S_0 \in \mathcal{S}} [\![(I \Rightarrow) \uplus ((\Gamma \Rightarrow \Delta) \uplus S_0)]\!]^+ & \\
& \leq_{\mathcal{X}} & \bigwedge_{S_0 \in \mathcal{S}} [\![(J \Rightarrow) \uplus ((\Gamma \Rightarrow \Delta) \uplus S_0)]\!]^+ & \text{(IH)} \\
& = & \bigwedge_{S_0 \in \mathcal{S}} [\![(\Gamma, J \Rightarrow \Delta) \uplus S_0]\!]^+ & \\
& = & [\![\Gamma, J \langle \mathcal{S} \rangle \Delta]\!]^+ & \\
& = & [\![(J \Rightarrow) \uplus S]\!]^+ &
\end{array}$$

□

In order to ease reasoning by induction on contexts, we give a formulation equivalent to Definition 7.3.2 as a context-free grammar:

Fact 8.6.10 *Contexts* $S\square$ are generated by the following grammar:

$$S\square ::= \square \mid \Gamma \rhd S\square, \Delta \mid \Gamma, S\square \rhd \Delta \mid \Gamma \langle \mathcal{S}; S\square \rangle \Delta$$

Definition 8.6.11 *The depth $|S\square|$ of a context $S\square$ is defined recursively as follows:*

$$|\square| = 0$$
$$|\Gamma \rhd S\square, \Delta| = |\Gamma, S\square \rhd \Delta| = |\Gamma \langle \mathcal{S}; S\square \rangle \Delta| = 1 + |S\square|$$

Lemma 8.6.15 (Contextual soundness) *If* $S \rightharpoonup T$ *then* $[\![U\boxed{T} \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \leq_{\mathcal{C}} [\![U\boxed{S} \uplus (\Gamma \Rightarrow \Delta)]\!]^+$.

Proof. By induction on $|U\square|$.

Base case Suppose $|U\square| = 0$. Then $U\square = \square$, and we conclude by shallow soundness (Lemma 8.6.10).

Positive case Suppose $|U\square| > 0$ and $U\square = \Gamma' \rhd U_0\square, \Delta'$. Then by IH we have $[\![U_0\boxed{T}]\!]^+ \leq_{\mathcal{C}} [\![U_0\boxed{S}]\!]^+$, and thus

$$\begin{aligned}
[\![(\Gamma' \rhd U_0\boxed{T}, \Delta') \uplus (\Gamma \Rightarrow \Delta)]\!]^+ &= [\![(\Rightarrow U_0\boxed{T}) \uplus (\Gamma, \Gamma' \rhd \Delta', \Delta)]\!]^+ \\
&\leq_{\mathcal{C}} [\![(\Rightarrow U_0\boxed{S}) \uplus (\Gamma, \Gamma' \rhd \Delta', \Delta)]\!]^+ && \text{(Lemma 8.6.14)} \\
&= [\![(\Gamma' \rhd U_0\boxed{S}, \Delta') \uplus (\Gamma \Rightarrow \Delta)]\!]^+
\end{aligned}$$

Negative case Suppose $|U\square| > 0$ and $U\square = \Gamma', U_0\square \rhd \Delta'$. Then by Lemma 8.6.2 we have $S^\dagger \rightharpoonup T^\dagger$, and thus by IH $[\![U_0^\dagger\boxed{T^\dagger}]\!]^+ \leq_{\mathcal{C}} [\![U_0^\dagger\boxed{S^\dagger}]\!]^+$, or equivalently $[\![U_0\boxed{T}^\dagger]\!]^+ \leq_{\mathcal{C}} [\![U_0\boxed{S}^\dagger]\!]^+$. Then by Lemma 8.6.5 we get $[\![U_0\boxed{S}]\!]^- \leq_{\mathcal{C}} [\![U_0\boxed{T}]\!]^-$, and thus

$$\begin{aligned}
[\![(\Gamma', U_0\boxed{T} \rhd \Delta') \uplus (\Gamma \Rightarrow \Delta)]\!]^+ &= [\![(U_0\boxed{T} \Rightarrow) \uplus (\Gamma, \Gamma' \rhd \Delta', \Delta)]\!]^+ \\
&\leq_{\mathcal{C}} [\![(U_0\boxed{S} \Rightarrow) \uplus (\Gamma, \Gamma' \rhd \Delta', \Delta)]\!]^+ && \text{(Lemma 8.6.14)} \\
&= [\![(\Gamma', U_0\boxed{S} \rhd \Delta') \uplus (\Gamma \Rightarrow \Delta)]\!]^+
\end{aligned}$$

Neutral case Suppose $|U\square| > 0$ and $U\square = \Gamma \langle \mathcal{S}; U_0\square \rangle \Delta$. Then by IH we have $[\![U_0\boxed{T} \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \leq_{\mathcal{C}} [\![U_0\boxed{S} \uplus (\Gamma \Rightarrow \Delta)]\!]^+$, and thus

$$\begin{aligned}
[\![\Gamma \langle \mathcal{S}; U_0\boxed{T} \rangle \Delta]\!]^+ &= [\![\Gamma \langle \mathcal{S} \rangle \Delta]\!]^+ \wedge [\![U_0\boxed{T} \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \\
&\leq_{\mathcal{C}} [\![\Gamma \langle \mathcal{S} \rangle \Delta]\!]^+ \wedge [\![U_0\boxed{S} \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \\
&= [\![\Gamma \langle \mathcal{S}; U_0\boxed{S} \rangle \Delta]\!]^+
\end{aligned}$$

□

Theorem 8.6.16 (Soundness) *If* $S \rightarrow T$ *then* $[\![T]\!]^+ \leq_{\mathcal{C}} [\![S]\!]^+$.

Proof. By definition of $\rightarrow$ and Lemma 8.6.15 with $\Gamma = \Delta = \varnothing$. □

We also get for free soundness with respect to the negative interpretation, which we call *co-soundness*:

Theorem 8.6.17 (Co-soundness) *If* $S \rightarrow T$ *then* $[\![S]\!]^- \leq_{\mathcal{C}} [\![T]\!]^-$.

Proof. By Lemma 8.6.3 we have $S^\dagger \rightarrow T^\dagger$, and thus by soundness $[\![T^\dagger]\!]^+ \leq_{\mathcal{C}} [\![S^\dagger]\!]^+$. Then we can conclude by Lemma 8.6.5. □

As for shallow soundness (Corollary 8.6.11 and Corollary 8.6.13), we can easily generalize the proof of Lemma 8.6.15 to Heyting algebras and Heyting-Brouwer algebras, and thus extend our soundness result to intuitionistic and bi-intuitionistic logic:

Corollary 8.6.18

- $S \rightarrow_{\mathsf{B}_{\mathcal{H}}} T$ *implies* $[\![T]\!]^+ \leq [\![S]\!]^+$
- $S \rightarrow_{\mathsf{B}_{\mathcal{HB}}} T$ *implies* $[\![T]\!]^+ \leq_{\mathcal{HB}} [\![S]\!]^+$

Proof. Lemma 8.6.10 is the only lemma used in the proof of Lemma 8.6.15 that relies on Boolean algebras. Thus we can easily replace it by Corollary 8.6.11 to get soundness in Heyting-Brouwer algebras.

For soundness in Heyting algebras, we know that the negative case in the proof of Lemma 8.6.15 will never happen because formulas cannot contain exclusions. The other cases only depend on Lemma 8.6.10, thus we can again replace it by Corollary 8.6.11. □

Once again in order to extend contextual soundness to dual-intuitionistic logic, we need to dualize lemmas to the negative interpretation:

Lemma 8.6.19 (Co-functoriality) *Let* $\mathcal{X} \in \{\mathcal{B}, \mathcal{HB}, \mathcal{C}\}$.

- $[\![I]\!]^- \leq_{\mathcal{X}} [\![J]\!]^-$ *implies* $[\![(\Rightarrow I) \uplus S]\!]^- \leq_{\mathcal{X}} [\![(\Rightarrow J) \uplus S]\!]^-$
- $[\![J]\!]^+ \leq_{\mathcal{X}} [\![I]\!]^+$ *implies* $[\![(I \Rightarrow) \uplus S]\!]^- \leq_{\mathcal{X}} [\![(J \Rightarrow) \uplus S]\!]^-$

Lemma 8.6.20 (Contextual co-soundness) *If* $S \rightharpoonup T$ *then* $[\![U\boxed{S} \uplus (\Gamma \Rightarrow \Delta)]\!]^- \leq_{\mathcal{C}} [\![U\boxed{T} \uplus (\Gamma \Rightarrow \Delta)]\!]^-$.

From the assumed proof of Lemma 8.6.20, we finally get:

Corollary 8.6.21 $S \rightarrow_{\mathsf{B}_{\mathcal{B}}} T$ *implies* $[\![S]\!]^- \leq_{\mathcal{B}} [\![T]\!]^-$.

Combined with the completeness proof of the next section, this will give us our main result that $\mathsf{B}_{\mathcal{H}}$, $\mathsf{B}_{\mathcal{B}}$, $\mathsf{B}_{\mathcal{HB}}$ and B capture exactly provability in intuitionistic, dual-intuitionistic, bi-intuitionistic and classical logic.

8.7. Completeness

We are now going to prove the *completeness* of the bi-intuitionistic (and propositional) fragment $\mathsf{B}_{\mathcal{HB}}$ of system B, by simulating the nested sequent system DBiInt of Postniece. In [206] she shows that this calculus is sound and complete with respect to another calculus LBiInt, and in Chapter 4

[206]: Postniece (2009), 'Deep Inference in Bi-intuitionistic Logic'

of her thesis [207] she proves that LBiInt is sound and complete with respect to the Kripke semantics of bi-intuitionistic logic. Importantly, the cut rule is shown to be *admissible* in both systems, through a syntactic process of cut-elimination in LBiInt. We will rely on this result to obtain admissibility of the cut rule i↑ in $\mathsf{B}_{\mathcal{HB}}$, and by extension in B, $\mathsf{B}_{\mathcal{H}}$ and $B_{\mathcal{B}}$. It might be interesting to have our own internal cut-elimination procedure for system B, notably to unveil its computational content in the spirit of the Curry-Howard correspondence. But this would lead us astray from the purpose of this thesis, and thus we leave this task for future work.

[207]: Postniece (2010), 'Proof theory and proof search of bi-intuitionistic and tense logic'

Definition 8.7.1 (Structure) *The structures of* DBiInt *are generated by the following grammar:*

$$X, Y ::= \varnothing \mid A \mid (X, Y) \mid X \Rightarrow Y$$

The structural connective ',' (comma) is associative and commutative and $\varnothing$ is its unit. We always consider structures modulo these equivalences.

Definition 8.7.2 (Structure translation) *The* translation $X^\bullet$ *of a structure X as a multiset of items Γ is defined recursively as follows:*

$$\begin{aligned} \varnothing^\bullet &= \varnothing & \quad (X, Y)^\bullet &= X^\bullet, Y^\bullet \\ A^\bullet &= A & \quad (X \Rightarrow Y)^\bullet &= X^\bullet \Rightarrow Y^\bullet \end{aligned}$$

Note that the translation $(-)^\bullet$ is clearly *injective*: in fact structures are isomorphic to multisets of items that contain only *unsaturated* subsolutions. Thus from now on, we will always apply the translation implicitly, and rely on meta-variables X, Y to distinguish structures from arbitrary solutions when necessary.

The rules of DBiInt are given in Figure 8.13. Note that like bubble calculi, DBiInt is truly a *deep inference* system, in the sense that rules can be applied on sequents nested arbitrarily deep inside structures[22]. The main difference lies in the fact that proofs in DBiInt are *trees* built up by composing traditional inference rules with multiple premisses, while we use saturated solutions (neutral bubbles) to internalize the tree structure of proofs inside solutions. This gives a lot of expressive power since saturated solutions can themselves be nested in unsaturated solutions and thus *polarized*, a phenomemon which cannot be simulated in DBiInt. This is why we did not prove soundness in Section 8.6 by simulating directly system B inside DBiInt, and conversely this will explain the ease with which DBiInt can be simulated inside system B.

22: Our presentation of rules is slightly different from [206]: the contexts in which rules apply are left implicit, and thus we do not rely on their polarity. The counterpart is that rules always apply on sequents and never on formulas, which makes them more verbose. Also we do not rely on the notion of "top-level formulas" of a structure, making the propagation rules yet more verbose.

Definition 8.7.3 (Syntactic entailment) *We say that* Γ entails Δ *in a fragment* F *of rules of* system B, *written* $\Gamma \mid\!\!\frac{}{\mathsf{F}} \Delta$, *if and only if* $\Gamma \Rightarrow \Delta \rightarrow^*_{\mathsf{F}} \langle\rangle$. *Similarly, we say that X entails Y in a fragment* F *of rules of* DBiInt, *written* $X \mid\!\!\frac{}{\mathsf{F}} Y$, *if and only if $X \Rightarrow Y$ has a proof in* DBiInt *using only rules in* F.

Propagation

Identity

$$\dfrac{}{X, A \Rightarrow A, Y}\ \mathsf{id}$$

$$\dfrac{X, A, (X', A \Rightarrow Y') \Rightarrow Y}{X, (X', A \Rightarrow Y') \Rightarrow Y}\ {\Rightarrow_{\mathsf{L1}}} \qquad \dfrac{X \Rightarrow (X' \Rightarrow A, Y'), A, Y}{X \Rightarrow (X' \Rightarrow A, Y'), Y}\ {\Rightarrow_{\mathsf{R1}}}$$

$$\dfrac{X, A \Rightarrow (X', A \Rightarrow Y'), Y}{X, A \Rightarrow (X' \Rightarrow Y'), Y}\ {\Rightarrow_{\mathsf{L2}}} \qquad \dfrac{X, (X' \Rightarrow A, Y') \Rightarrow A, Y}{X, (X' \Rightarrow Y') \Rightarrow A, Y}\ {\Rightarrow_{\mathsf{R2}}}$$

Logic

$$\dfrac{}{X, \bot \Rightarrow Y}\ \bot_{\mathsf{L}} \qquad \dfrac{}{X \Rightarrow \top, Y}\ \top_{\mathsf{R}}$$

$$\dfrac{X, A \wedge B, A, B \Rightarrow Y}{X, A \wedge B \Rightarrow Y}\ \wedge_{\mathsf{L}} \qquad \dfrac{X \Rightarrow A, A \wedge B, Y \quad X \Rightarrow B, A \wedge B, Y}{X \Rightarrow A \wedge B, Y}\ \wedge_{\mathsf{R}}$$

$$\dfrac{X, A \vee B, A \Rightarrow Y \quad X, A \vee B, B \Rightarrow Y}{X, A \vee B \Rightarrow Y}\ \vee_{\mathsf{L}} \qquad \dfrac{X \Rightarrow A, B, A \vee B, Y}{X \Rightarrow A \vee B, Y}\ \vee_{\mathsf{R}}$$

$$\dfrac{X, A \supset B \Rightarrow A, Y \quad X, A \supset B, B \Rightarrow Y}{X, A \supset B \Rightarrow Y}\ \supset_{\mathsf{L}} \qquad \dfrac{X \Rightarrow (A \Rightarrow B), A \supset B, Y}{X \Rightarrow A \supset B, Y}\ \supset_{\mathsf{R}}$$

$$\dfrac{X, A \subset B, (A \Rightarrow B) \Rightarrow Y}{X, A \subset B \Rightarrow Y}\ \subset_{\mathsf{L}} \qquad \dfrac{X \Rightarrow A, A \subset B, Y \quad X, B \Rightarrow A \subset B, Y}{X \Rightarrow A \subset B, Y}\ \subset_{\mathsf{R}}$$

Figure 8.13.: Rules of the deep nested sequent system DBiInt

Lemma 8.7.1 (Simulation of DBiInt) *If* $X \vdash_{\mathsf{DBiInt}} Y$ *then* $X \vdash_{\mathsf{B}_{\mathcal{HB}} \setminus \{\mathsf{i}\uparrow\}} Y$.

Proof. By induction on the derivation of $X \vdash_{\mathsf{DBiInt}} Y$. The detailed proof is available in appendix (Section A.2). □

Assuming that the consequence relation of the Kripke semantics used by Postniece to prove the completeness of DBiInt coincides with the order relation of Heyting-Brouwer algebras, we have the following fact:

Fact 8.7.1 (Completeness of DBiInt) If $A \leq_{\mathcal{HB}} B$ then $A \vdash_{\mathsf{DBiInt}} B$.

Combined with the simulation of DBiInt from Lemma 8.7.1, this gives us the *cut-free* completeness of $\mathsf{B}_{\mathcal{HB}}$:

Theorem 8.7.2 (Cut-free completeness) *If* $A \leq_{\mathcal{HB}} B$ *then* $A \vdash_{\mathsf{B}_{\mathcal{HB}} \setminus \{\mathsf{i}\uparrow\}} B$.

In fact there are other rules of $\mathsf{B}_{\mathcal{HB}}$ that were not used in the simulation, namely the $\mathbb{F}$-rule $\mathsf{f}\uparrow$, and all $\mathbb{M}$-rules other than p. Combined with the soundness of $\mathsf{B}_{\mathcal{HB}}$ (Corollary 8.6.18), this gives us the following *admissibility* theorem:

Theorem 8.7.3 (Admissibility) *If* $\vdash_{\mathsf{B}_{\mathcal{HB}}} A$ *then* $\vdash_{\mathsf{B}_{\mathcal{HB}} \setminus \{\mathsf{i}\uparrow, \mathsf{f}\uparrow, \mathsf{p}-, \mathsf{p}+, \mathsf{a}, \mathsf{a}-, \mathsf{a}+\}} A$.

Although these rules are admissible, they do not seem to be derivable from other rules. We believe that they might help in making proofs more *compact* by improving *factorizability*, just like the cut rule does in standard proof formalisms.

As in sequent calculus, every rule of system B other than $\mathsf{i}\uparrow$ satisfies the *subformula property*:

Fact 8.7.2 (Subformula property) If $S \to_{\mathsf{B}\setminus\{\mathsf{i}\uparrow\}} T$ and $A \prec T$, then there is a formula B such that A is a subformula of B and $B \prec S$.

Thanks to Theorem 8.7.3, we thus get that $\mathsf{B}_{\mathcal{HB}}$ is *analytic*. This has many nice consequences, a well-known one being that when searching for a proof of a given solution S, one does not need to come up with or "invent" a formula that does not appear in S. This is crucial when designing *automated* decision procedures because it reduces drastically the search space, but is also desirable in the setting of *interactive* proof building. Indeed with our Proof-by-Action interpretation of bubble calculi (Section 7.4), this means that all logical reasoning can be performed by direct manipulation of *what is already there*. Then the cut rule $\mathsf{i}\uparrow$ is indispensable, but confined to a role of *theory building*: it allows the creation of *lemmas*, in order to make proofs shorter and more tractable by humans.

As noted in [206], one can simply ignore rules related to the exclusion connective $\subset$ to get a sound and complete system for intuitionistic logic. In DBiInt, these rules are the introduction rules $\subset_{\mathsf{R}}$ and $\subset_{\mathsf{L}}$, as well as the propagation rules $\Rightarrow_{\mathsf{L1}}$ and $\Rightarrow_{\mathsf{R2}}$. Indeed, propagation rules are only useful in combination with introduction rules, since $\subset_{\mathsf{L}}$ is the only rule of DBiInt that can introduce nested sequents in negative contexts. The situation is similar in system B, and in fact the proof of Lemma 8.7.1 shows that the intuitionistic fragment $\mathsf{B}_{\mathcal{H}}$ is sufficient to simulate DBiInt without the aforementioned rules. The dual argument can be made for dual-intuitionistic logic, and thus we obtain (cut-free) intuitionistic (resp. dual-intuitionistic) completeness of $\mathsf{B}_{\mathcal{H}}$ (resp. $\mathsf{B}_{\mathcal{B}}$):

[206]: Postniece (2009), 'Deep Inference in Bi-intuitionistic Logic'

Corollary 8.7.4 (Intuitionistic completeness)

- *If* $A \leq_{\mathcal{H}} B$ *then* $A \vdash_{\mathsf{B}_{\mathcal{H}}\setminus\{\mathsf{i}\uparrow\}} B$.
- *If* $A \leq_{\mathcal{B}} B$ *then* $A \vdash_{\mathsf{B}_{\mathcal{B}}\setminus\{\mathsf{i}\uparrow\}} B$.

Figure 8.14 shows a proof of the *double-negation elimination law* (DNE) $\neg\neg A \Rightarrow A$ in system B. Since $\mathsf{B}_{\mathcal{H}}$ is intuitionistically complete, the well-

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\langle\rangle}{\langle\langle\rangle\rangle}\,\mathsf{p}}{\langle \Rightarrow (\langle\rangle)\rangle}\,\mathsf{p}+}{\langle \Rightarrow (A \Rightarrow A)\rangle}\,\mathsf{i}\downarrow}{\langle \Rightarrow (A \Rightarrow), A\rangle}\,\mathsf{f}{+}{+}\downarrow}{\langle \Rightarrow (A \Rightarrow)\rangle\; A}\,\mathsf{f}{+}\downarrow}{\langle \Rightarrow (A \Rightarrow); \langle\rangle\rangle\; A}\,\mathsf{p}}{\langle \Rightarrow (A \Rightarrow); \bot \Rightarrow \rangle\; A}\,\bot-}{\langle \Rightarrow (A \Rightarrow \bot); \bot \Rightarrow \rangle\; A}\,\bot+}{\langle \Rightarrow \neg A; \bot \Rightarrow \rangle\; A}\,\supset+}{\neg\neg A \Rightarrow A}\,\supset-
$$

Figure 8.14.: Proof of DNE in system B

known double-negation embedding of classical logic into intuitionistic logic tells us that $\neg\neg A$ is provable in $\mathsf{B}_{\mathcal{H}}$ (and a fortiori in system B) if A is a theorem of classical logic. Combining the two previous facts, we obtain the classical completeness of system B. In fact the proof of DNE only relies on the use of the $\mathsf{f{+}{+}}\!\downarrow$ rule, so we can make the following stronger statement:

Corollary 8.7.5 (Classical completeness) *If A is a theorem of classical logic, then* $\vdash_{\mathsf{B}_{\mathcal{H}} \cup \{\mathsf{f{+}{+}}\downarrow\}} A$.

Proof. By the double-negation embedding, we have $\vdash_{\mathsf{B}_{\mathcal{H}}} \neg\neg A$. Then we can build the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\langle\rangle}{\langle\,\langle\rangle\,\rangle}\,\mathsf{p}}{\langle \neg\neg A \Rightarrow A\rangle}\,\mathrm{DNE}}{\langle \neg\neg A \Rightarrow \rangle\ A}\,\mathsf{f{+}}\!\downarrow}{\langle\,\langle\rangle; \neg\neg A \Rightarrow \rangle\ A}\,\mathsf{p}}{\dfrac{\langle \Rightarrow \neg\neg A; \neg\neg A \Rightarrow \rangle\ A}{\Rightarrow A}\,\mathsf{i}\!\uparrow}$$

□

Alas this argument makes use of the $\mathsf{i}\!\uparrow$ rule. Note however that the reason we chose to prove completeness of $B_{\mathcal{HB}}$ by simulating a rather exotic system like DBiInt, was that standard sequent calculi for bi-intuitionistic logic like the one of Rauszer [209] are not *cut-free* complete; and in our literature review, DBiInt was the cut-free system closest in its syntax and rules to system B. But for classical logic we do not have this limitation, and thus it is straightforward to simulate directly a cut-free sequent calculus such as G3cp inside system B [183]:

[209]: Rauszer (1974), 'A Formalization of the Propositional Calculus of H-B Logic'

[183]: Negri et al. (2001), *Structural Proof Theory*

Lemma 8.7.6 (Simulation of G3cp) *If* $\Gamma \vdash_{\mathsf{G3cp}} \Delta$*, then* $\Gamma \vdash_{\mathsf{B}_{\mathcal{H}} \cup \{\mathsf{f{+}{+}}\downarrow\} \setminus \{\mathsf{i}\uparrow\}} \Delta$.

Proof. By induction on the G3cp derivation. See Section A.2 for the detailed proof. □

Lastly, let us mention a recent result of Goré and Shillito [101], where they uncover a distinction between a *weak* and a *strong* consequence relation in the semantics of bi-intuitionistic logic. Although they define the same set of theorems, these two relations have different properties at the meta-level, and thus the authors argue that they define two distinct logics, called respectively wBIL and sBIL. At the end of the article, they conjecture that the various existing calculi in the literature are sound and complete for wBIL, including a calculus designed by Postniece. Since our completeness proof is by simulation of the system DBiInt also designed by Postniece, we follow this conjecture regarding the completeness of the bi-intuitionistic fragment $B_{\mathcal{HB}}$ of system B. For soundness, we would need to clarify the relationship between Heyting-Brouwer algebras and these consequence relations, which stem instead from an analysis of the Kripke semantics of bi-intuitionistic logic. Since system B offers a very expressive syntax, it would be interesting to investigate its ability to capture both wBIL and sBIL, maybe by using distinct sets of $\mathbb{F}$-rules. Goré and Shillito suggest that a framework that captures both *provability* and *refutability* "in one shot" would be needed, and we believe system B might just provide this: indeed a derivation $S \to^* \langle\rangle$ can be read both as a *proof* of $[\![S]\!]^+$ and a *refutation* of $[\![S]\!]^-$.

8.8. Invertible calculus

8.8.1. Modifying rules

An important thing to note, is that all the rules of DBiInt are *invertible*[23]. Thus it follows immediately from Lemma 8.7.1 that one can just take the translation of the rules of DBiInt in system B, and get a complete, fully invertible calculus. But this would be a waste of the expressive power and nice properties of system B, like linearity and locality.

23: Lemma 5.2.4 in Postniece's thesis [207].

Instead, we will target precisely the non-invertible rules of system B, and modify only those. From the proof of Lemma 8.6.10, we can identify which rules of system B are invertible, and which are probably not. Indeed if the soundness of a rule only relies on a chain of equivalences, then it is necessarily invertible. On the contrary if it relies on an inequality, then it is probably not invertible[24].

24: To ensure that it is not invertible, we would need additionally to find a countermodel that invalidates the converse inequality.

Fact 8.8.1 (Invertibility of system B) All rules in the fragment $\mathbb{I} \cup \{\mathsf{c}-, \mathsf{c}+\} \cup \mathbb{M} \cup \mathbb{H} \setminus \{\supset-, \subset+\}$ of system B are invertible.

Thus the only remaining rules of system B that are (most probably) not invertible are the weakening rules $\{\mathsf{w}-, \mathsf{w}+\}$, all the $\mathbb{F}$-rules, and the $\mathbb{H}$-rules $\{\supset-, \subset+\}$ that *apply* an implication/exclusion[25]. In Figure 8.15 we define the $\mathsf{B}_{\mathsf{inv}}$ calculus, which results from the following modifications to the previous rules:

25: For quantifier rules, we conjecture that as in sequent calculus, the rules $\{\forall+, \exists-\}$ are invertible, while the rules $\{\forall-, \exists+\}$ are not. And as in sequent calculus, this can be remedied by systematically duplicating the instantiated formula.

Weakening Here we follow a standard technique in sequent calculus, that merges the weakening rule in all *terminal* rules of the calculus (i.e. rules with no premisses). In bubble calculi, the notion of premiss is captured by neutral bubbles; thus we incorporate weakenings in all rules that solve subgoals by saturating solutions. Those are the rules $\{\mathsf{i}\downarrow, \mathsf{p}-, \mathsf{p}+\}$, which for the occasion have also been generalized to arbitrary solutions. Indeed in system B we restricted them to unsaturated solutions to make them *local*, but here the weakenings break locality anyway, and the general version improves *factorizability* by solving instantly all subgoals inside $\triangleright$.

Flow As shown by the simulation of Lemma 8.7.1, the *propagation* rules of DBiInt combine an instance of *contraction* followed by the application

Identity

$$\frac{\langle\rangle}{\Gamma, A \triangleright A, \Delta}\;\mathsf{i}{\downarrow}$$

Flow

$$\frac{\Gamma, I\,\langle \Gamma', I \blacktriangleright \Delta'\,;\,\mathscr{S}\rangle\;\Delta}{\Gamma, I\,\langle \Gamma' \blacktriangleright \Delta'\,;\,\mathscr{S}\rangle\;\Delta}\;\mathsf{f}{-}{\downarrow} \qquad \frac{\Gamma\,\langle \mathscr{S}; \Gamma' \blacktriangleright I, \Delta'\rangle\; I, \Delta}{\Gamma\,\langle \mathscr{S}; \Gamma' \blacktriangleright \Delta'\rangle\; I, \Delta}\;\mathsf{f}{+}{\downarrow}$$

$$\frac{\Gamma, I \triangleright (\Gamma', I \blacktriangleright \Delta'), \Delta}{\Gamma, I \triangleright (\Gamma' \blacktriangleright \Delta'), \Delta}\;\mathsf{f}{-}{+}{\downarrow} \qquad \frac{\Gamma, (\Gamma' \blacktriangleright I, \Delta') \triangleright I, \Delta}{\Gamma, (\Gamma' \blacktriangleright \Delta') \triangleright I, \Delta}\;\mathsf{f}{+}{-}{\downarrow}$$

$$\frac{\Gamma, I, (\Gamma', I \blacktriangleright \Delta') \triangleright \Delta}{\Gamma, (\Gamma', I \blacktriangleright \Delta') \triangleright \Delta}\;\mathsf{f}{-}{-}{\uparrow} \qquad \frac{\Gamma \triangleright (\Gamma' \blacktriangleright I, \Delta'), I, \Delta}{\Gamma \triangleright (\Gamma' \blacktriangleright I, \Delta'), \Delta}\;\mathsf{f}{+}{+}{\uparrow}$$

$$\frac{\Gamma, I \triangleright (\Gamma', I \blacktriangleright \Delta'), \Delta}{\Gamma \triangleright (\Gamma', I \blacktriangleright \Delta'), \Delta}\;\mathsf{f}{-}{+}{\uparrow} \qquad \frac{\Gamma, (\Gamma' \blacktriangleright I, \Delta') \triangleright I, \Delta}{\Gamma, (\Gamma' \blacktriangleright I, \Delta') \triangleright \Delta}\;\mathsf{f}{+}{-}{\uparrow}$$

$$\frac{\Gamma, I, (\Gamma', I \blacktriangleright \Delta') \triangleright \Delta}{\Gamma, I, (\Gamma' \blacktriangleright \Delta') \triangleright \Delta}\;\mathsf{f}{-}{-}{\downarrow} \qquad \frac{\Gamma \triangleright (\Gamma' \blacktriangleright I, \Delta'), I, \Delta}{\Gamma \triangleright (\Gamma' \blacktriangleright \Delta'), I, \Delta}\;\mathsf{f}{+}{+}{\downarrow}$$

Membrane

$$\frac{\Gamma\,\langle \mathscr{S}\rangle\;\Delta}{\Gamma\,\langle \mathscr{S}; \langle\rangle\rangle\;\Delta}\;\mathsf{p}$$

$$\frac{\langle\rangle}{\Gamma, (\langle\rangle) \triangleright \Delta}\;\mathsf{p}{-} \qquad \frac{\langle\rangle}{\Gamma \triangleright (\langle\rangle), \Delta}\;\mathsf{p}{+}$$

$$\frac{\Gamma\,\langle S\rangle\;\Delta}{\Gamma\,\langle\langle S\rangle\rangle\;\Delta}\;\mathsf{a}$$

$$\frac{\Gamma, S \triangleright \Delta}{\Gamma, (\langle S\rangle) \triangleright \Delta}\;\mathsf{a}{-} \qquad \frac{\Gamma \triangleright S, \Delta}{\Gamma \triangleright (\langle S\rangle), \Delta}\;\mathsf{a}{+}$$

Heating

$$\frac{\Gamma \triangleright \Delta}{\Gamma, \top \triangleright \Delta}\;\top{-} \qquad \frac{\Gamma \triangleright (\langle\rangle), \Delta}{\Gamma \triangleright \top, \Delta}\;\top{+}$$

$$\frac{\Gamma, (\langle\rangle) \triangleright \Delta}{\Gamma, \bot \triangleright \Delta}\;\bot{-} \qquad \frac{\Gamma \triangleright \Delta}{\Gamma \triangleright \bot, \Delta}\;\bot{+}$$

$$\frac{\Gamma, A, B \triangleright \Delta}{\Gamma, A \wedge B \triangleright \Delta}\;\wedge{-} \qquad \frac{\Gamma\,\langle \Rightarrow A; \Rightarrow B\,\rangle\;\Delta}{\Gamma \Rightarrow A \wedge B, \Delta}\;\wedge{+}$$

$$\frac{\Gamma\,\langle A \Rightarrow; B \Rightarrow\,\rangle\;\Delta}{\Gamma, A \vee B \Rightarrow \Delta}\;\vee{-} \qquad \frac{\Gamma \triangleright A, B, \Delta}{\Gamma \triangleright A \vee B, \Delta}\;\vee{+}$$

$$\frac{\Gamma, A \supset B\,\langle \Rightarrow A; B \Rightarrow\,\rangle\;\Delta}{\Gamma, A \supset B \Rightarrow \Delta}\;\supset{-} \qquad \frac{\Gamma \triangleright (A \Rightarrow B), \Delta}{\Gamma \triangleright A \supset B, \Delta}\;\supset{+}$$

$$\frac{\Gamma, (A \Rightarrow B) \triangleright \Delta}{\Gamma, A \subset B \triangleright \Delta}\;\subset{-} \qquad \frac{\Gamma\,\langle \Rightarrow A; B \Rightarrow\,\rangle\; A \subset B, \Delta}{\Gamma \Rightarrow A \subset B, \Delta}\;\subset{+}$$

$$\frac{\Gamma, \forall x.A, A\{t/x\} \triangleright \Delta}{\Gamma, \forall x.A \triangleright \Delta}\;\forall{-} \qquad \frac{\Gamma \triangleright A, \Delta}{\Gamma \triangleright \forall x.A, \Delta}\;\forall{+}$$

$$\frac{\Gamma, A \triangleright \Delta}{\Gamma, \exists x.A \triangleright \Delta}\;\exists{-} \qquad \frac{\Gamma \triangleright A\{t/x\}, \exists x.A, \Delta}{\Gamma \triangleright \exists x.A, \Delta}\;\exists{+}$$

In the $\forall{+}$ and $\exists{-}$ rules, x is not free in Γ, Δ and $\triangleright$.

Figure 8.15.: Rules for the invertible bubble calculus $\mathsf{B}_{\mathsf{inv}}$

of a $\mathbb{F}$-rule on the duplicated formula. Thus we can make all $\mathbb{F}$-rules of system B invertible by systematically duplicating the moved formula, although this breaks *linearity*.

A downside of propagation rules in the style of DBiInt, is that they create a lot of unnecessary copies of the moved formula A. Often, one will want to move A in a subgoal/supergoal at a distance n in the proof tree, with $n > 1$. Usually this would be performed by n applications of $\mathbb{F}$-rules, which by linearity indeed just move the formula. But with propagation rules, n copies of A will be created, with one copy in each subgoal met on the path to the destination.

To prevent this, one would need a way to copy formulas at an arbitrary distance. This can be done with inference rules that are *doubly* deep, by encoding the path to the destination as a second context inside the context where the rule is applied[26]. It turns out to be hard to express in bubbles, because this requires a syntactic way to describe contexts that correspond to valid flow paths of arbitrary length[27]. But in principle it should be feasible, and would enable a more comfortable use in a Proof-by-Action setting.

26: Such rules are sometimes called *super-switch* rules in the deep inference literature, see for instance [108, Chapter 8, Section 2.1].

27: This problem is solved trivially in the flower calculus (Chapter 10), by formulating a so-called *pollination* relation.

Note also that we removed the f↑ rule of Figure 8.8. Indeed even after turning it into a propagation rule, the moved copy of the duplicated subgoal S cannot be weakened because it lives in a neutral bubble. Thus the rule stays non-invertible, and cannot be included in $\mathsf{B}_{\mathsf{inv}}$. Fortunately, we showed that it is admissible in Theorem 8.7.3, so this is not problematic.

Implication/Exclusion The last source of non-invertibility is the $\mathbb{H}$-rules $\supset-$ and $\subset+$, that respectively allow to use an implication hypothesis, and prove an exclusion conclusion. Here we can just duplicate the implication/exclusion formula, as in the introduction rules of DBiInt. Also like in DBiInt, we removed the contraction rules c− and c+, which are now merged with these two rules as well as the $\mathbb{F}$-rules. Although contraction rules are invertible, they induce a lot of complexity in proof search, because it is hard to predict the (occurrences of) formulas that need to be duplicated, and one can duplicate *ad infinitum*. Thus it is preferable to design a calculus where they are admissible. But unlike what is done in DBiInt, we did not incorporate contraction in other $\mathbb{H}$-rules. Thus we cannot simulate exactly all the introduction rules of DBiInt in $\mathsf{B}_{\mathsf{inv}}$.

Remark 8.8.1 We also changed the $\bot-$ and $\top+$ rules, so that they create polarized, saturated empty solutions. This makes them both local, and generic with respect to the saturation status of the ambient solution. The previous version can then be simulated by combination with the popping rules p− and p+.

These modifications only change superficially the proof of soundness, and thus we do not redo it. As for completeness, we would need to prove that the contraction rules are admissible, in order to solve the aforementioned problem of simulating DBiInt's introduction rules:

Lemma 8.8.1 (Admissibility of contraction)

- *If* $\vdash_{\mathsf{B_{inv}}} S\boxed{\Gamma, A, A \rhd \Delta}$, *then* $\vdash_{\mathsf{B_{inv}}} S\boxed{\Gamma, A \rhd \Delta}$.
- *If* $\vdash_{\mathsf{B_{inv}}} S\boxed{\Gamma \rhd A, A, \Delta}$, *then* $\vdash_{\mathsf{B_{inv}}} S\boxed{\Gamma \rhd A, \Delta}$.

Note that it is sufficient to prove admissibility of contraction on formulas, rather than on arbitrary items. Indeed we only need it to simulate the introduction rules of DBiInt, which always duplicate formulas. For now we only conjecture completeness of $\mathsf{B_{inv}}$, since it not clear what method should be used to prove Lemma 8.8.1. In her thesis [207, Lemma 5.2.3], Postniece does a proof by induction on the depth of the derivation, relying on the fact that all introduction rules of DBiInt preserve the principal formula; but this is precisely what we are trying to avoid with our version of the rules. Of course, if we either give up on this constraint or include contraction rules in $\mathsf{B_{inv}}$, then we immediately get our desired result: $\mathsf{B_{inv}}$ is a fully invertible calculus, where the same fragments as system B capture intuitionistic, dual-intuitionistic, bi-intuitionistic and classical logic.

8.8.2. Semi-automated proof search

In the intuitionistic (propositional) fragment of $\mathsf{B_{inv}}$, a canonical way to search for a proof of a formula A consists in the following 5 *phases*, applied successively in a loop until the saturated empty solution is reached:

Decomposition Decompose A by applying recursively $\mathbb{H}$-rules, until either atoms, negative implications $\supset$, negative disjunctions $\vee$, or positive conjunctions $\wedge$ are reached.

Indeed since the $\supset-$ rule duplicates the implication, it cannot be used to decompose it. Regarding the $\vee-$ and $\wedge+$ rules, they can only be applied when the formula is in an *unsaturated* solution. A first option is to let the system automatically distribute them in all unsaturated subsolutions that are reachable, so that it can keep decomposing them. But this might create an explosion in the number of created subgoals. Another option is to let the user manually decompose them. We believe this second option is preferable, if one wants to keep control over the proof search process. Indeed, it is only natural that the user should be able to choose which *cases* to consider when building a proof.

Absorption Apply the absorption rules $\{\mathsf{a}, \mathsf{a}-, \mathsf{a}+\}$ wherever possible. This will prevent atoms from being unnecessarily stuck on neutral membranes in the next phase. This phase can also be trivially automated.

Linking Try to bring together every pair of dual atoms, so that they annihilate each other in an instance of the $\mathsf{i}\downarrow$ rule. This is reminiscent of our *drag-and-drop* actions of Chapter 2. In a touch-based GUI, rather than dragging a complex formula onto another complex formula, one could *pinch* together the two atoms: if there is no $\mathbb{F}$-law stucking one of the atoms on some membrane (orange arrows in Figure 8.10), then the pinch succeeds, and the system saturates the subsolution at the location where the pinch ends by applying the $\mathsf{i}{\downarrow}$ rule. Thus the user can choose the subgoal to solve by controlling the destination of the pinch, which can be seen as a more symmetric and powerful version of DnD actions. Generally though, one will want to apply the following *rule of thumb* (pun intended):

Fact 8.8.2 (Rule of thumb) When linking a pair of dual atoms, follow these steps:

1. put your *thumb* on the *outermost* atom, and your *index* on the *innermost* atom;
2. try to bring your index to your thumb;
3. if you get stuck on a membrane, try to bring your thumb to your index;
4. if you again get stuck, then give up on this pair.

The point of this heuristic, is that it should maximize the *factorization* of the proof: when it succeeds, it will solve the subgoal that is located closest to the root of the goal, maximizing the size of the pruned branch, and thus the number of subgoals solved in one go. It can also be used to completely automate this phase.

Popping Pop every saturated empty bubble in the goal with the rules $\{\mathsf{p}, \mathsf{p}-, \mathsf{p}+\}$. This phase can also be trivially automated, and corresponds to the unit elimination phase in subformula linking (Section 3.3).

Application When there are no more pairs of dual atoms, or all the remaining pairs have been given up (last step of the rule of thumb), let S be the current goal, and

$$\mathsf{imp}(S) \triangleq \left\{(S_0\square, A, B, \rhd, \Delta) \;\middle|\; S = S_0\boxed{\Gamma, A \supset B \rhd \Delta} \text{ for some } \Gamma\right\}$$

If $\mathsf{imp}(S) = \varnothing$, then S should not be provable, and we can stop the proof search procedure. Otherwise for each $(S_0\square, A, B, \rhd, \Delta) \in \mathsf{imp}(S)$, we might need to apply the $\supset-$ rule on $A \supset B$, either directly in $S_0\square$ if $\rhd = \Rightarrow$ and $\max_{I \in \Delta} |I| = 0$, otherwise in some subgoal $T\boxed{U} \in \rhd \cup \Delta$. This is where the proof needs *insight*, because it is not clear if the antecedant A will be provable with the context available in $S_0\square$ or in one of the $T\square$, or if the hypothesis B is even needed at all.

A first possibility is to let the user rely on her intuition, by choosing manually a specific subsolution in $\mathsf{imp}(S)$ to apply the $\supset-$ rule upon. Additionally, she might need to determine a subgoal $T\boxed{U}$ in which A is provable, and first duplicate $A \supset B$ in $T\square$ before applying $\supset-$. This will always be possible with the $\mathbb{F}$-rules $\mathsf{f}{-}{\downarrow}$ and $\mathsf{f}{-}{+}{\downarrow}$. Ideally, she would also pick the most general $T\square$ to factorize the proof, by minimizing its depth $|T\square|$ (Definition 8.6.11).

A second possibility is to duplicate eagerly every $A \supset B$ of $\mathsf{imp}(S)$ in every unsaturated subsolution of S where it can be so, and then apply $\supset-$ on all the newly created copies. To avoid an explosion of the size of S, the system should mark all the copies as *used*, so that during the next **Application** phases, all the already *used* copies are ignored, and only the original occurrence of $A \supset B$ is considered.

Then we can restart the procedure, by applying the **Decomposition** phase to every copy of A and B.

Remark 8.8.2 By adopting $\mathbb{H}$-rules in the style of DBiInt's introduction rules, we would make the **Decomposition** phase, and thus the whole procedure inoperable, since the **Linking** phase depends crucially on it. Allowing contraction rules would also jeopardize the potential completeness of the procedure, because contraction might be needed at unpredictable moments, and on unpredictable formulas.

A strength of our proof search procedure, compared to the state-of-the-art in other formalisms, is that most of its automation preserves the *size* (number of atoms) and the *structure* of the goal:

- In the **Decomposition** phase, if we opt out of the automatic distribution of negative $\vee$ and positive $\wedge$, then the system will only apply $\mathbb{H}$-rules that split logical connectives, and create a *partition* of the atoms of the goal by enclosing them in bubbles. Thus the size of the goal is kept intact, and the structure modified but in a controlled, local way.
- In the **Absorption** phase, we simply merge some membranes together, preserving both the size and the structure of the goal.
- In the **Linking** phase, the particular way in which we use $\mathbb{F}$-rules ensures that we only decrease the number of atoms. Indeed, if we assume as discussed earlier that we have "super-flow" rules that copy at a distance, then either:
  1. the link is successful, and the two created copies of atoms are immediately destroyed by the $\mathsf{i}{\downarrow}$ rule. Then the solved subsolution is entirely pruned out, decreasing the size of the goal; or
  2. the link fails, but then we can instantly "undo" it. Or rather, one should consider that rules are applied only when the link is successful.
- In the **Popping** phase, entire branches of the goal are pruned out, decreasing the size of the goal.

Then only the automation of the **Application** phase (and part of the **Decomposition** phase) is susceptible of both significantly increasing the size of the goal, and altering its global structure. But as is the case for every phase, the user can easily opt out of this automation, and do the reasoning manually when it is necessary to keep the goal understandable by humans. Typically in an educational setting, it should be quite instructive to have the ability to perform **Decomposition** and **Linking** by hand (literally).

8.8.3. Failure of full iconicity

Because of the implicit contraction in the rules ⊃− and ⊂+, one cannot fully decompose a formula into an equivalent solution by deterministically applying a sequence of $\mathbb{H}$-rules (and possibly $\mathbb{F}$-rules, to distribute positive conjunctions and negative disjunctions). Thus $\mathsf{B}_{\mathsf{inv}}$ fails to be *fully iconic*, because it relies on the *symbolic* connectives ⊃ and ⊂ to represent logical statements.

This can be understood as resulting from the inability of solutions to represent natively *negative implications* and *positive subtractions*, although they can represent natively all other polarizations of connectives. This is illustrated by the mapping of Figure 8.16 from polarized formulas to equivalent solutions, which is really just the $\mathbb{H}$-rules of system B (Figure 8.9) where the right-hand solution is enclosed in a bubble of the corresponding polarity. The reader can easily check that if A is mapped to S, then $[\![A]\!]^+ \simeq [\![S]\!]^+$.

This seems to be a fundamental limitation of system B, caused by its symmetric treatment of implication and subtraction. For instance in the nested sequent calculus JN of Guenot for implicative logic [108, Chapter 3], which is fully decomposable, nested sequents that appear in negative contexts are interpreted as implications, as illustrated by the left introduction rule e (Figure 8.17). But we cannot do this in system B, because this would conflict with the subtractive reading of negative solutions, i.e. $[\![A \Rightarrow B]\!]^- = [\![A]\!]^- \subset [\![B]\!]^-$. In Chapter 10, the problem will also be solved through an asymmetric treatment of nested sequents, capturing only intuitionistic logic instead of bi-intuitionistic logic. But this is a small price to pay, since bi-intuitionistic logic does not (currently) have any applications in the realm of interactive theorem proving.

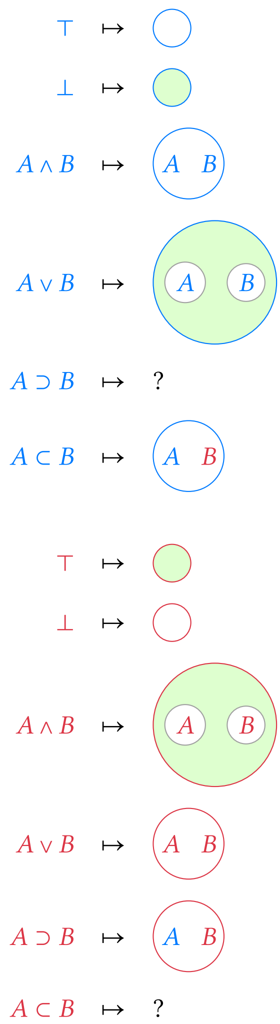


Figure 8.16.: Mapping of formulas to equivalent solutions

$$\frac{\Gamma, (\Delta, A \Rightarrow B) \Rightarrow C}{\Gamma, (\Delta \Rightarrow A \supset B) \Rightarrow C}\,\mathsf{e}$$

Figure 8.17.: Left introduction rule for ⊃ in JN

Existential Graphs | 9.

9.1 Alpha graphs 156
9.1.1 Icons 156
9.1.2 Relationship with formulas 157
9.2 Illative transformations 158
9.3 Graphs as multisets . 161
9.3.1 Syntax 161
9.3.2 Rules 161
9.4 Illative atomicity . . . 163
9.4.1 Insertions and omissions 163
9.4.2 Computational aspects . 164
9.4.3 The more scientific way . 165
9.5 Soundness 167
9.6 Completeness 169
9.6.1 Calculus of structures . . 170
9.6.2 Simulation 172
9.6.3 Analyticity 175
9.7 Beta graphs 176
9.7.1 Syntax 177
9.7.2 Deconstructing identity . 178
9.7.3 Rules 180
9.8 Gardens 183

> The System of Existential Graphs which I have now sufficiently described — or, at any rate, have described as well as I know how, leaving the further perfection of it to others — greatly facilitates the solution of problems of Logic, as will be seen in the sequel, not by any mysterious properties, but simply by substituting for the symbols in which such problems present themselves, concrete visual figures concerning which we have merely to say whether or not they admit certain describable relations of their parts. Diagrammatic reasoning is the only really fertile reasoning. If logicians would only embrace this method, we should no longer see attempts to base their science on the fragile foundations of metaphysics or a psychology not based on logical theory; and there would soon be such an advance in logic that every science would feel the benefit of it.
>
> **Charles S. Peirce**, *Prolegomena to an Apology for Pragmaticism*, 1906

C. S. Peirce is famous for his contributions to symbolic logic, including among others his eponymous law for classical logic, and his pioneering work on the algebra of relations and quantification [199]. But far less widespread are his achievements in the realm of diagrammatic logic, or *iconic logic* as Shin calls it [221]. He dedicated a large chunk of his life to the investigation of graphical systems, starting in 1882 with the *entitative graphs* and culminating with the *existential graphs* (EGs), which he developed from 1896 until his death in 1914 [214]. Interestingly, Peirce perceived existential graphs as his *"chef d'oeuvre"*, and that they *"ought to be the logic of the future"*[1].

[214]: Roberts (1973), *The Existential Graphs of Charles S. Peirce*

1: Both citations are sourced in [214, p. 11].

Recent works have started to realize this vision: for example Sowa based his conceptual graphs for computerized knowledge representation on EGs [225]; Brady, Trimble [25, 26], Gianluca, Rocco [35] and Haydon, Sobociński [115] proposed various reconstructions of EGs through the lens of *topology* and *category theory*; lastly, Melliès, Zeilberger [170] and Bonchi et al. [23] refined respectively the interpretations of [26] and [115] by making further connections with *linear logic* [92] and *linear bicategories*. The full story has yet to be told, but we hope that our work will constitute one more step towards the vision Peirce had in mind.

[225]: Sowa (1976), 'Conceptual Graphs for a Data Base Interface'

[25]: Brady et al. (2000), 'A categorical interpretation of C.S. Peirce's propositional logic Alpha'

[26]: Brady et al. (2000), *A String Diagram Calculus for Predicate Logic and C. S. Peirce's System Beta*

[35]: Caterina et al. (2020), 'A New Syntax for Diagrammatic Logic'

[115]: Haydon et al. (2020), 'Compositional Diagrammatic First-Order Logic'

[170]: Melliès et al. (2016), 'A bifibrational reconstruction of Lawvere's presheaf hyperdoctrine'

[23]: Bonchi et al. (2024), *Diagrammatic Algebra of First Order Logic*

[92]: Girard (1987), 'Linear logic'

In this chapter, we propose a self-contained exposition of EGs, that tries at the same time to be faithful to the original presentation of the systems by Peirce, and more modern in some aspects of their formalization. The goal will be to familiarize the reader with the unique approach to proofs inherent to EGs, which can be difficult to relate to more standard frameworks like Hilbert and Gentzen proof systems, and even deep inference proof systems like the calculus of structures. This shall prove useful to get a good understanding of the historical and technical foundations behind our *flower calculus*, to be introduced in Chapter 10.

The chapter is organized as follows: we start in Section 9.1 by presenting the diagrammatic syntax of the system Alpha of EGs for classical propositional logic. In Section 9.2, we introduce the inference rules of Alpha for manipulating EGs, called *illative transformations* by Peirce. In Section 9.3, we give an equivalent formulation of the syntax and rules of Alpha as a *multiset* rewriting system. In Section 9.4, we formalize a variant of the (De)iteration principle described by Peirce in [202] that eliminates the need for the Double−cut principle, and discuss how it was motivated by Peirce's quest for *illative atomicity*. In Section 9.5, we take advantage of our reformulation to give a simple proof of soundness for Alpha, based on a direct truth-evaluation of graphs. In Section 9.6 we give a syntactic proof of completeness for Alpha, by simulating the calculus of structures SKS of Brünnler and Tiu [31]. In this way, Alpha is shown to have subsystems that inherit the *locality* property of SKS, and where the Deletion and Insertion rules are respectively admissible for *provability* and *refutability*, making Alpha *analytic*. In Section 9.7, we illustrate the original mechanism of *lines of identity* used by Peirce to handle quantifiers and equality in his Beta system. We end in Section 9.8 by showing how to recast lines of identity in a more traditional binder-based syntax.

[202]: Peirce (1906), 'Prolegomena to an Apology for Pragmaticism'

[31]: Brünnler et al. (2001), 'A Local System for Classical Logic'

9.1. Alpha graphs

Peirce designed in total three systems of EGs, which he called respectively Alpha, Beta and Gamma. They were invented chronologically in that order, which also captures their relationship in terms of complexity: Alpha is the foundation on which the other systems are built, and can today be understood as a diagrammatic calculus for classical *propositional* logic. As we will see in Section 9.7, Beta corresponds to a variable-free representation of *predicate* logic without function symbols, and with primitive support for *equality*. The last system Gamma is more experimental, with various unfinished features that have been interpreted as attempts to capture *modal* [261] and *higher-order* logics.

[261]: Zeman (1964), 'The Graphical Logic of C. S. Peirce'

Digression

At the end of his life, Peirce pushed his experimentations beyond the scope of *logic* in the contemporary sense of the word, with so-called *tinctured* existential graphs [214, Chapter 6]. Roughly, the idea was to represent a variety of *modes of expression* with different background shades on the sheet of assertion, not unlike our graphical depiction of *saturated* solutions in Figure 8.9, or the background colors used for the various kinds of text boxes in this document. In addition to the usual act of asserting the truth of a proposition, one could for instance express a *subjective* or *objective* possibility, or signify an interrogative or imperative mood, all by using different colors. For print in publications, he would in fact use *heraldic tinctures* instead of colors, hence the "tinctured" qualificative. The precise rules, meaning and purpose of tinctured EGs remain elusive to this day, and might constitute the most esoteric part of Peirce's work.

9.1.1. Icons

Sheet of Assertion The most fundamental concept of Alpha is the *sheet of assertion*, denoted by SA thereafter. It is the space where statements are scribed by the reasoner, typically a sheet of paper, a blackboard, or a computer display. In a proof assistant, this would either be the buffer of a text editor where the user writes her theories, or the proof view displaying goals to be proved, depending on who the reasoner is (the user or the computer, respectively). This last analogy suggests an important property of SA: it must offer a *virtually infinite* amount of space, so that one can perform as much reasoning as needed. Just like a Turing machine has an infinite tape, so that one can perform as much computation as needed. In symbolic logic, this is captured by the fact that formulas, although usually finite, can have an unbounded size.

As its name indicates, scribing a statement on SA amounts to *asserting its truth*. Thus very naturally, the empty SA where nothing is scribed will denote vacuous truth, traditionally symbolized by the formula $\top$.

Juxtaposition As we know from natural deduction, asserting the truth of the conjunction $a \wedge b$ of two propositions a and b, amounts to asserting *both* the truth of a and the truth of b. In Alpha, there is no need to introduce the symbolic connective $\wedge$, since one can just write both a and b at distinct locations on SA:

$$a \quad b$$

More generally, one might consider any two portions G and H of SA, and interpret their *juxtaposition* $G\,H$ as signifying that we assert the truth of their conjunction.

Cuts Asserting the truth of the negation $\neg a$ of a proposition a, amounts to *denying* the truth of a. Using the original notation of Peirce, this is done in Alpha by *enclosing* a in a closed curve like so:

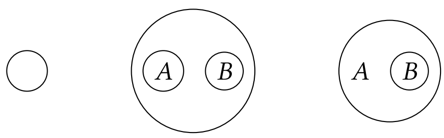
Peirce called such curves *cuts*[2], because they ought to be seen as literal cuts in the paper sheet that embodies SA. Note that they do not need to be circles: all that matters is that a is in a separate area from the rest of SA. This is precisely the content of the *Jordan curve theorem* in topology, and thus we can take cuts to be arbitrary Jordan curves. This entails in particular that cuts cannot intersect each other, but can be freely nested inside each other. Then as for juxtaposition, one can replace a by any *graph* G — i.e. any portion of SA — as long as the cut does not intersect other cuts in G.

2: Not to be confused with the name given to instances of the cut rule in sequent calculus.

9.1.2. Relationship with formulas

With just these two icons, juxtaposition and cuts, one can therefore assert the truth of any proposition made up of conjunctions and negations and built from atomic propositions. Importantly, the only symbols needed for doing so are letters $a, b, c \dots$ denoting atomic propositions, that is "pure" symbols that do not have any logical meaning associated to them.

Now, it is well-known that $\{\wedge, \neg\}$ is *functionally complete*, meaning that any boolean truth function can be expressed as the composition of boolean conjunctions and negations. In particular, the symbolic definitions of absurdity $\bot \triangleq \neg\top$, classical disjunction $A \vee B \triangleq \neg(\neg A \wedge \neg B)$ and classical implication $A \supset B \triangleq \neg(A \wedge \neg B)$ can be expressed by the following three graphs[3]:



3: Note the resemblance with the translation of formulas as solutions in Figure 8.16, in particular for negative disjunctions.

Thus one can easily encode any propositional formula into a classically equivalent graph. Conversely, one can translate any graph into a classically equivalent formula, as has been shown for instance in [221]. In fact,

[221]: Shin (2002), *The Iconic Logic of Peirce's Graphs*

there are usually many possible formula readings of a given graph. One reason is that juxtaposition of graphs is a *variadic* operation, as opposed to conjunction of formulas which is *dyadic*: thus formulas that only differ up to *associativity* are associated to the same graph. Also, thanks to the topological nature of SA, juxtaposition is naturally *commutative*: the locations of two juxtaposed graphs do not matter, as long as they live in the same area delimited by a cut. The combination of these properties is called the *isotropy* of SA in [159], and is captured in traditional proof theory through the use of *(multi)sets* for modelling contexts in sequents.

[159]: Ma et al. (2019), 'A graphical deep inference system for intuitionistic logic'

Remark 9.1.1 In a first version of EGs called *entitative graphs*, Peirce used juxtaposition to denote *disjunction* instead of conjunction. Although $\{\vee, \neg\}$ is also functionally complete, Peirce quickly grew unsatisfied with these entitative graphs, stating that EGs formed "a far preferable system on the whole" (Ms 280, pp. 21–22). I find it interesting that more contemporary works in logic have also made the choice to take conjunction and negation as their primitive operations, like the tensorial logic of Melliès [168], or the realizability constructions for linear logic in Girard's transcendental syntax [77].

9.2. Illative transformations

Deep inference In order to have a proof system, one needs a collection of *inference rules* for deducing true statements from other true statements. In Alpha, inference rules are implemented by what Peirce called *illative transformations* on graphs. In modern terminology, they correspond to rewriting rules that can be applied to any subgraph. By measuring the depth of a subgraph as the number of cuts in which it is enclosed, we thus have that the rules of Alpha are applicable on subgraphs of arbitrary depth. This makes Alpha deserving of the title of deep inference system.

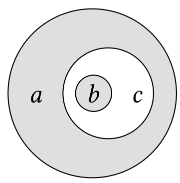


Figure 9.1.: Peirce's notation for emphasizing negative areas

Polarity Before introducing the rules, let us make a small change in the way we depict the graphs. The idea is that we want to visualize more clearly the polarity of any subgraph G, understood as the *parity* of the number of cuts (negations) enclosing G. In one of his unpublished manuscripts (Ms 514), Peirce did this by *shading* negative areas — those enclosed in an odd number of cuts — in gray, as illustrated in Figure 9.1 [226]. Unconstrained by hand-drawing, one could adopt an even more iconic notation, where negative areas are *literally* drawn like a negative in photography, by inverting white and black. The example of Figure 9.1 would then be drawn as in Figure 9.2. However in this thesis, we will stick to Peirce's notation, which is both less straining for the eyes by being less contrasted, and more economical in ink for print.

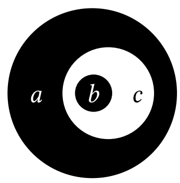


Figure 9.2.: Drawing negative areas literally in negative

[226]: Sowa (2011), 'Peirce's Tutorial on Existential Graphs'

A nice advantage of these notations is that they remove the need to count manually the number of cuts starting from the top-level of SA: the information is immediately apparent in the subgraph, and thus completely *local*[4].

4: A similar device is used in the deep inference system ISp of Tiu [241], where the polarities of substructures are attached to them as explicit labels.

> **Remark 9.2.1** Whereas in bubble calculi the concept of polarity was understood as a property of *objects* — i.e. utterances of propositions — by assigning them opposite colors (blue and red), the previous notations for graphs suggest that it is instead a property of the *space* in which objects reside. This is more natural from the point of view of *game semantics*: for instance in a game of chess, the two players can easily exchange their roles by switching places or rotating the board by 180°, rather than by repainting laboriously each piece in the opposite color.

Inference rules Quite surprisingly, Peirce showed that one only needs five inference rules to get a *strongly complete* system, in the sense that if the truth of a graph G entails the truth of another graph H, then G can always be rewritten into H by applying exclusively instances of these five rules[5]. A nice way to understand the rules of Alpha is as *edition principles*, like the most basic actions one executes pervasively when editing text on a computer[6]. The first two rules are the most powerful and mysterious in all systems of EGs, and can be applied in areas of any polarity:

5: Of course Peirce did not show completeness formally in the sense of modern model theory, although Sowa argues in [226, Section 4] that he had started to develop his own model theory equivalent to Tarski's (but closer to the *game-theoretical semantics* of Hintikka [121]).

6: Even though computers did not exist yet in Peirce's time! In fact, Martin Irvine argues in [130] that Peirce anticipated many developments in computer science and information technologies, such as the use of electrical switches to compute boolean functions, whose invention is usually attributed to Claude Shannon.

Iteration ***(Copy & Paste)*** A graph G may be duplicated at any depth inside of a juxtaposed graph H. Using our notation for holed contexts from previous chapters, this can be represented schematically like so:

$$G \;\; H\boxed{} \;\rightarrow\; G \;\; H\boxed{G} \qquad\qquad G \;\; H\boxed{} \;\rightarrow\; G \;\; H\boxed{G}$$

It can be seen as a deep generalization of the axiom rule of sequent calculus, where the top-level occurrence of G justifies the occurrence of G located inside $H\Box$[7]. Note that while in the axiom rule, the justifying (resp. justified) occurrence must be a negative hypothesis (resp. a positive conclusion), the Iteration rule also allows the opposite relationship of a conclusion justifying a hypothesis, thus exhibiting one aspect of the cut rule of sequent calculus.

7: This might also be related to the notion of *justified move* in game semantics, where the nesting of cuts in the context $H\Box$ corresponds to a sequence of alternating moves between Player and Opponent.

Deiteration ***(Factorization)*** Formally, this is the converse of Iteration:

$$G \;\; H\boxed{G} \;\rightarrow\; G \;\; H\boxed{} \qquad\qquad G \;\; H\boxed{G} \;\rightarrow\; G \;\; H\boxed{}$$

Its interpretation as an edition principle is a bit trickier, but it can be understood as a form of *sharing* of information. Indeed, it roughly says that a subgraph G can be erased if it already occurs "higher" on SA. Also this does precisely the opposite of copy-pasting, which is known in software engineering as *factorization*[8].

Compared to sequent calculus, it can be seen as a deep generalization of the *contraction* rule, the base case where $H\Box = \Box$ giving $G\, G \rightarrow G$.

8: This is closely related to the kind of factorization at work in bubble calculi. In particular, the fact that the factorizing occurence is higher and usually outside of a cut is very reminiscent of the *outward* flow rules of system B (those whose name ends with ↑ in Figure 8.8); and the deduplicating effect makes Deiteration even closer to the variant of the same rules in $\mathsf{B}_{\mathsf{inv}}$ (Figure 8.15).

The applicability of the next two rules depends on the polarity of the subgraph's area:

Insertion Any graph G may be inserted in a negative area:

$$\rightarrow \;\; G$$

This is akin to a *weakening* rule, stating that one might add (useless) hypotheses at will. The closest equivalent we found in the deep inference literature is indeed the weakening rule w$\mathsf{l}\!\downarrow$ of ISp in [241].

[241]: Tiu (2006), 'A Local System for Intuitionistic Logic'

Deletion Any graph G occurring in a positive area may be erased:

$$G \quad \rightarrow$$

This is exactly the dual of Insertion, stating that if a proposition is known to be true, then one might as well refrain from asserting it. It is the only *non-analytic* rule of the system when reading rules from conclusion to premiss, since G does not already appear in the right-hand side. It is thus strongly related to the cut rule of sequent calculus, which it can simulate together with the Deiteration rule.

The last rule is more of a *space management* principle that works as an *isotopy*, i.e. a bidirectional topological deformation:

Double–cut A *double-cut* may be inserted or erased around any graph G:

The bidirectional arrow $\leftrightarrow$ expresses that the rule can be applied in both directions. Logically, this corresponds to the classical equivalence $\neg\neg A \simeq A$, where in particular the deletion direction $\neg\neg A \supset A$ is not true intuitionistically. Topollogically, the double-cut forms a *ring*, that separates G from the rest of SA while preserving its polarity. Then the two directions of the rules can be understood as the following dual *homotopies*:

Contraction The ring is created by cutting SA around G, and then *contracting* the inner area where G resides on itself. This effectively "pulls apart" G from the rest of the sheet, leaving apparent in the empty space of the ring whatever lies behind SA. Peirce thought of positive and negative areas as being the *recto* and *verso* of SA, respectively. Thus in the positive version of the rule (on the left), the ring would represent negative empty space on the verso of SA.

Expansion The ring is erased by *expanding* the inner area where G resides towards the outer border of the ring. Unfolding the metaphor to its conclusion, the inner area is then "glued back" to the rest of SA[9].

9: This is reminiscent of the *absorption* rules $\{\mathsf{a}, \mathsf{a}-, \mathsf{a}+\}$ of system B, as is very clear in their graphical presentation (Figure 8.9).

Figure 9.3 shows a derivation of Peirce's law with the rules of Alpha. Note that the direction of arrows has been reversed compared to the above presentation: as usual, we prefer to read rules from conclusion to premiss, starting from the goal to prove — here the graph associated to the formula $((a \supset b) \supset a) \supset a$ — that we reduce to the empty goal, represented by the empty SA. Also, the reader unfamiliar with EGs might find it hard to convince herself that all the steps followed in the derivation are sound logically. We suggest her to either build a *syntactic* intuition for the rules by practicing them on various tautologies of propositional logic, or to wait until we give a formal *semantic* proof of soundness in Section 9.5.

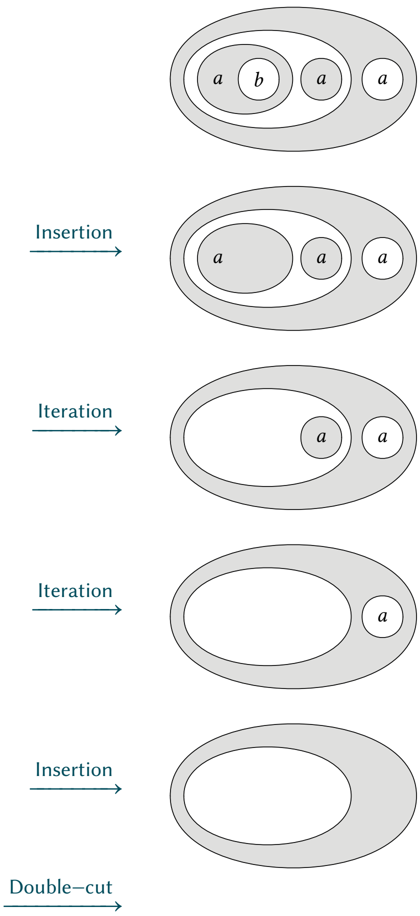


Figure 9.3.: A derivation of Peirce's law in Alpha

9.3. Graphs as multisets

9.3.1. Syntax

As noted by various authors[10], the nesting of cuts on SA induces a *tree* structure on graphs: each cut constitutes a node, whose children are either leaves corresponding to atomic propositions residing in the area of the cut, or nodes corresponding to nested cuts. Empty cuts have no children, and thus also form leaves of the tree. Then SA may be seen either as a forest of atoms and cuts, or as a rooted tree whose root represents SA, and is distinguished from cut nodes. This can be captured by the following grammar:

10: See for instance the Tree Existential Graphs of Roberts and Pronovost [215], or [25, Section 2.2].

$$\mathsf{SA} ::= G \qquad G, H, K ::= g_1, \dots, g_n \qquad g, h, k ::= a \mid [G]$$

Example 9.3.1 The graph of Figure 9.1 may be written as either one of the following expressions:

$$[a, [[b], c]] \qquad [a, [c, [b]]] \qquad [[[b], c], a] \qquad [[c, [b]], a]$$

To abstract from the specific order in which nodes are sequenced in this notation, and thus represent faithfully the isotropy of SA, we formally define the graphs of Alpha as (recursive) *finite multisets*:

Definition 9.3.1 (Graph) *Given a denumerable set of atomic propositions $\mathscr{A}$, the sets of nodes $\mathbf{N}_\alpha$ and graphs $\mathbf{G}_\alpha$ are defined mutually inductively as follows:*

(Spot) *If $a \in \mathscr{A}$, then $a \in \mathbf{N}_\alpha$;*

(Area) *If $G \subset \mathbf{N}_\alpha$ is a finite multiset, then $G \in \mathbf{G}_\alpha$;*

(Enclosure) *If $G \in \mathbf{G}_\alpha$, then $[G] \in \mathbf{N}_\alpha$.*

The terminology written in parentheses is the one used by Peirce to denote the same concepts in [202]. Note the similarity with the definitions of bubbles (enclosures) and solutions (areas). The main difference between graphs and solutions, is that in the former formulas (ions) are restricted to atoms (spots), and they are not *polarized* (see Remark 9.2.1).

[202]: Peirce (1906), 'Prolegomena to an Apology for Pragmaticism'

9.3.2. Rules

The five rules of Alpha can now be formalized as multiset rewriting rules on graphs. But first, we need a notion of *context* in which rules apply:

Definition 9.3.2 (Context) *A context $G\square$ is a graph which contains*

ALPHA

$$\frac{K\boxed{G, H\boxed{}}}{K\boxed{G, H\boxed{G}}}\ \textsf{Iter} \qquad \frac{K\boxed{G, H\boxed{G}}}{K\boxed{G, H\boxed{}}}\ \textsf{Deit}$$

$$\frac{K^{-}\boxed{}}{K^{-}\boxed{G}}\ \textsf{Ins} \qquad \frac{K^{+}\boxed{G}}{K^{+}\boxed{}}\ \textsf{Del}$$

$$\frac{K\boxed{G}}{K\boxed{[[G]]}}\ \textsf{Dcut}\!\downarrow \qquad \frac{K\boxed{[[G]]}}{K\boxed{G}}\ \textsf{Dcut}\!\uparrow$$

Figure 9.4.: Inductive presentation of the rules of Alpha

exactly one occurrence of a special node written $\square$, *called its* hole. *The hole can always be filled (substituted) with any other graph* H *or context* $K\square$, *producing a new graph* $G\boxed{H}$ *or context* $G\boxed{K\square}$. *In particular, filling with the empty graph* $\varnothing$ *will yield a graph* $G\boxed{}$, *which is just* $G\square$ *with its hole removed.*

Then to reason by induction on contexts, we need to define formally how to measure their *depth*. It turns out the only way to increase the depth of an graph is to insert *cuts*, and thus the depth of a context coincides with its number of *inversions*, i.e. the number of cuts enclosing its hole:

Definition 9.3.3 (Depth) *The* depth $|-|$ *of contexts is defined recursively by:*

$$|H, \square| = 0$$
$$|H, [G\square]| = |G\square| + 1$$

Definition 9.3.4 (Inversions) *The number of* inversions *of a context* $G\square$ *is defined by* $\mathsf{inv}(G\square) = |G\square|$.

Definition 9.3.5 (Polarity) *We say that a context* $G\square$ *is* positive *if* $\mathsf{inv}(G\square)$ *is even, and* negative *otherwise. We denote positive and negative contexts respectively by* $G^{+}\square$ *and* $G^{-}\square$.

The inductive version of the rules of Alpha is given in Figure 9.4, as a set of unary inference rules on graphs: when read *top-down*, they correspond to usual inferences from premiss to conclusion, as we first introduced them in Section 9.2. But as already mentioned there, we will rather emphasize their *bottom-up* reading: then they express the different ways in which one may choose to simplify a goal.

Definition 9.3.6 (Derivation) *We write $G \to H$ to indicate a rewrite* step *in* Alpha*, that is an instance of some rule from Figure 9.4 with H as premiss and G as conclusion. A* derivation $G \to^n H$ *is a sequence of rewrite steps $G_0 \to G_1 \ldots \to G_n$ with $G_0 = G$, $G_n = H$ and $n \geq 0$. Generally the length n of the derivation does not matter, and we just write $G \to^* H$.*

Definition 9.3.7 (Proof) *A* proof *of a* graph *G is a derivation $G \to^* \varnothing$.*

9.4. Illative atomicity

9.4.1. Insertions and omissions

A remarkable feat of Peirce's rules, on which he insisted very much, is that they are only expressed in terms of *insertions* and *omissions* of graphs on SA. He thought that those were the *smallest* steps in which reasoning could be dissected, making his system extremely appropriate for *analytical* purposes. This is summarized in the following excerpt [202, p. 533]:

[202]: Peirce (1906), 'Prolegomena to an Apology for Pragmaticism'

> In the first place, the most perfectly analytical system of representing propositions must enable us to separate illative transformations into indecomposable parts. Hence, an illative transformation from any proposition, A, to any other, B, must in such a system consist in first transforming A into AB, followed by the transformation of AB into B. For an omission and an insertion appear to be indecomposable transformations and the only indecomposable transformations.

We already considered this question of decomposing logical inferences into their most elementary operations, when reflecting on the graphical presentation of BJ at the end of Section 7.3.4. In this setting, the most basic insertions and omissions we could find were not logically *sound*, whereas in Alpha they are. This is quite promising, and prompts us to reevalute our conception of *illative atomicity*, understood precisely as the definition of what it means for an inference step to be (the most) *elementary*.

Note that this should be distinguished from the notion of *analyticity*, as popularized by Gentzen with the *subformula property* in sequent calculus: the latter is concerned with the analysis of *propositions* into their constituents through inference rules, while here we are interested in the analysis of the inference rules themselves[11]. However there is a conceptual bottleneck, because inference rules are usually conceived *by definition* as the smallest constituents of proofs in a given proof system; and it is very hard to formulate objective criteria for comparing rules in different proof systems.

11: We will give a positive answer to the question of analyticity in Alpha at the end of Section 9.6.

Apart from Peirce, the only other logician we know of who attempted to give a non-trivial account of illative atomicity is J.-Y. Girard. In fact it can be argued that it is the main motivation behind most of his works starting from linear logic, which became explicit in ludics with his slogan "From the rules of logic to the logic of rules" [93].

[93]: Girard (2001), 'Locus Solum'

9.4.2. Computational aspects

Linearity There is an intriguing remark by Peirce in [202, pp. 536–537] about the atomicity of the rules presented thus far, that seems to have gone unnoticed in the literature on EGs. Indeed, Peirce argues that the principle of Double–cut "cannot be assumed as an undeduced Permission" — i.e. a primitive rule of the system — because when the double-cut is removed, the area inside the inner cut becomes identified with the area outside the outer cut, a transformation that "is not strictly an Insertion or a Deletion".

Another way to interpret this, is that Double–cut is the only *linear* rule of the system, in the sense that the premiss and conclusion contain exactly the same atomic graphs. Contrast this with linear logic, which instead takes linearity as the criterion for illative atomicity, as exemplified by the linear decomposition of implication $A \supset B \triangleq\ !A \multimap B$. This might be the consequence of an opposite treatment given to *negation*: while in EGs it is the only primitive constructor of the system — remember that the only way to increase the depth of a graph is with a cut, in LL negation is the only defined notion through De Morgan dualities. Thus EGs are closer (at least syntactically) to *type theories*, which also take a negative operation (the arrow or dependent product type) as their sole primitive construct.

Interactivity and Locativity Peirce then suggests a "more scientific way", where the principle of Double–cut is subsumed by a restricted variant of the principles of Iteration and Deiteration. His description of this "more scientific" (De)iteration principle is based on a relation of *local justification* (our terminology) between two areas of a graph, that captures the fact that the deeper occurrence of G in the Iter and Deit rules (Figure 9.4) is justified by the other occurrence by virtue of their respective *locations*. Later in the text, Peirce emphasizes the importance of this locative aspect of argumentation [202, pp. 544–545]:

> [...] when an Argument is brought before us, there is brought to our notice aprocess whereby the Premisses bring forth the Conclusion, not informing theInterpreter of its Truth, but appealing to him to assent thereto. This Processof Transformation, which is evidently the kernel of the matter, is no more builtout of Propositions than a motion is built out of position.

Once again, game-theoretical ideas and the concept of space (Remark 9.2.1) are prominent in this excerpt: Truth is not primitive, but rather a side effect of the interaction between an Interpreter (Opponent) being lead to agree with the Graphist (Player), whenever the latter performs a transformation on the graph under discussion. The soundness of such a transformation guarantees that this will work for *any* Interpreter/Opponent, leading to what is known as a *winning strategy* in game semantics. Since illative transformations only consist of insertions and omissions, whose validity depends solely on the positions where they occur in the graph, it ensues that the components of an argumentation can be reduced to "motions" (moves) that relate pure locations.

It is then interesting to notice that the quest for illative atomicity, who led Peirce to discover these interactive and locative aspects of logic, also led Girard to identify these properties as fundamental, in his recent works on ludics [93] and transcendental syntax [76]. We tend to share the vision put forth by Boris Eng in his thesis [76, §24.4], that logic is mostly about *space* and the *shape* of objects, while *time* and *dynamics* pertain more to the realm of computation. In this view, Peirce's systems of EGs are a logico-computational complex where each aspect can clearly be identified: the Process of Illative Transformation is an interactive computation among the Graphist and Interpreter, whose logical nature is determined by the spatial constraints of the Permissions, that are expressible thanks to the topology induced by cuts on SA[12].

12: Both Peirce and Girard also shared the ambition to develop a comprehensive philosophical foundation for logic, as part of a more general theory of *meaning*: for Peirce it was his *semeiotic*, stemming from his overarching doctrine of *pragmaticism* [33]; for Girard it was the theory of programming languages and their semantics.

13: Although we limit ourselves to propositional logic in Alpha, while in [202] Peirce also accounts for the lines of identity of Beta handling predicate logic, to be introduced in Section 9.7.

9.4.3. The more scientific way

Let us now go back to the "more scientific" (De)iteration principle proposed by Peirce. With our formalization of graphs as multisets, we can give a more rigorous formulation than the original natural language description given by Peirce[13]. In this setting, a location in a graph is represented by the hole of a context, thus the relation of local justification between two areas is defined on contexts:

Definition 9.4.1 (Local justification) *Given two contexts $G\square$ and $H\square$, we say that $H\square$ is* locally justified *by $G\square$, written $G\square \succeq_0 H\square$, if and only if one of the following conditions holds for some $K\square$, $G_0\square$, $H_0\square$ such that $G\square = K\boxed{G_0\square}$ and $H\square = K\boxed{H_0\square}$:*

1. $G_0\square = H_0\square = K_0, \square$ *for some* K_0*;*
2. $G_0\square = K_0, [K_1], \square$ *and* $H_0\square = K_0, [K_1, \square]$ *for some* K_0, K_1*;*
3. $G_0\square = K_0, [K_1, [K_2]], \square$ *and* $H_0\square = K_0, [K_1, [K_2, \square]]$ *for some* K_0, K_1, K_2*;*
4. $G_0\square = K_0, [[K_1, \square]]$ *and* $H_0\square = K_0, \square, [[K_1]]$ *for some* K_0, K_1*.*

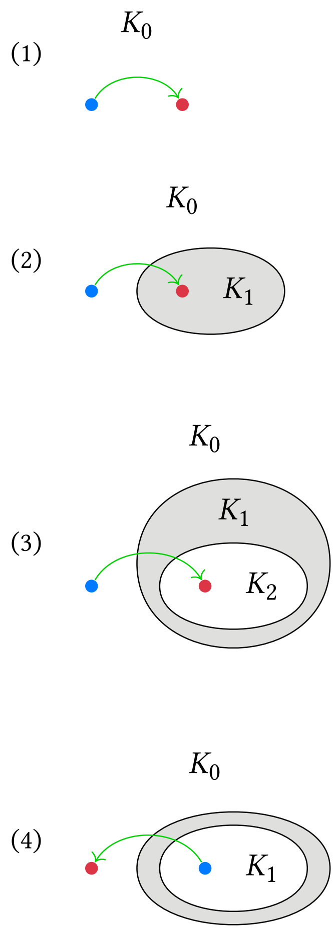


Figure 9.5.: Graphical representation of the four conditions of local justification

These four conditions are exactly the formal counterpart of those given by Peirce in [202]. They might be more easily understood by looking at their graphical representation in Figure 9.5: the red and blue dots denote respectively the locations of the holes in the justified context $H\square$ and justifying context $G\square$, as suggested by the arrow between them. In particular, it becomes clear that it is Condition 4 that will account for the principle of Double−cut.

Then the new rules of iteration Iter+, Iter− and deiteration Deit+, Deit− are given in a so-called *atomic* variant of Alpha, that we name Alpha[a] in Figure 9.6. As promised, Alpha[a] only comprises rules that truly are insertions and omissions of *arbitrary* graphs[14]. The atomic (de)iteration rules are a restriction of the original ones in two respects:

ALPHA$^{\mathsf{a}}$

$$\frac{K^{-}\boxed{H_S\boxed{G}, H_T^{-}\boxed{}}}{K^{-}\boxed{H_S\boxed{G}, H_T^{-}\boxed{G}}}\ \mathsf{Iter-} \qquad \frac{K^{+}\boxed{H_S\boxed{G}, H_T^{+}\boxed{}}}{K^{+}\boxed{H_S\boxed{G}, H_T^{+}\boxed{G}}}\ \mathsf{Iter+}$$

$$\frac{K^{-}\boxed{H_S\boxed{G}, H_T^{+}\boxed{G}}}{K^{-}\boxed{H_S\boxed{G}, H_T^{+}\boxed{}}}\ \mathsf{Deit-} \qquad \frac{K^{+}\boxed{H_S\boxed{G}, H_T^{-}\boxed{G}}}{K^{+}\boxed{H_S\boxed{G}, H_T^{-}\boxed{}}}\ \mathsf{Deit+}$$

$$\frac{K^{-}\boxed{}}{K^{-}\boxed{G}}\ \mathsf{Ins} \qquad \frac{K^{+}\boxed{G}}{K^{+}\boxed{}}\ \mathsf{Del}$$

In the rules {Iter+, Iter−, Deit+, Deit−}, we require that $H_S\square \succeq_0 H_T\square$.

Figure 9.6.: The illatively atomic system Alpha$^{\mathsf{a}}$

Locality Following Definition 9.4.1, the depth of the justified context $H_T\square$ can be at most 2 in the atomic rules, while it is unbounded in the original rules. This does not hinder their expressivity however: global (de)iterations can be simulated by successive applications of local ones, by erasing intermediate copies with the Ins and Del rules. This is because the *global justification* relation $\succeq$ associated with the original rules coincides with the transitive closure of the local relation $\succeq_0$, modulo the 4$^{\text{th}}$ condition for double-cuts[15].

Polarity In the atomic iteration (resp. deiteration) rules, the justified context must be positive (resp. negative), while it can have an arbitrary polarity in the original rules. This is expressed by splitting each of the latter into two rules, one where the outer context $K\square$ is positive (Iter+, Deit+), and one where it is negative (Iter−, Deit−). Again, this does not alter their deductive power: every iteration (resp. deiteration) in a negative (resp. positive) context can be trivially performed by an instance of the Ins (resp. Del) rule. Thus atomic rules eliminate a redundancy of Alpha, where many insertions/omissions could be interpreted as instances of either Iter/Deit or Ins/Del. They also eliminate the possibility for a conclusion to justify a hypothesis, as remarked in Section 9.2 when commenting on the principle of Iteration, making them closer to the rules of sequent calculus[16].

In the rest of this chapter, we settle on the more standard system Alpha, and leave a more detailed and rigorous study of Alpha$^{\mathsf{a}}$ for future work. But the above informal arguments should convince the reader that there is little doubt that Alpha$^{\mathsf{a}}$ is both sound and complete if and only if Alpha is, which is the object of the following two sections.

14: The terminology "atomic" might be a bit confusing: here we think of *illative* atomicity in Peirce's sense, not the fact that the graphs manipulated by rules are atomic, which might be termed *structural* atomicity. There seems to be a symmetric tradeoff when comparing EGs to the calculus of structures: in the former, one maximizes illative atomicity by minimizing linearity and structural atomicity; while in the latter, one maximizes structural atomicity and linearity by minimizing illative atomicity. This will become explicit in Section 9.6, when simulating the calculus of structures SKS in Alpha.

15: Our notation for justification relations actually comes from [159], where the authors define the same notion informally for an intuitionistic variant of EGs.

16: We suspect however that the more general (de)iteration rules are still relevant from a computational point of view.

9.5. Soundness

We are now going to prove formally the *soundness* of each rule of Alpha, by showing that if $G \to H$ and H is *true*, then so is G. In classical propositional logic, one can easily evaluate the truth of any formula A, given a *truth-valuation* $v : \mathscr{A} \to \{0, 1\}$ for the atoms of A. The same applies to graphs:

Definition 9.5.1 (Evaluation) *Given a valuation $v : \mathscr{A} \to \{0, 1\}$ and a graph G, the* evaluation $v(G)$ *of G is defined by mutual recursion as follows:*

$$v(G) = \begin{cases} 1 & \text{if } G = \varnothing \\ \min_{g \in G} v(g) & \text{otherwise} \end{cases}$$

$$v([G]) = 1 - v(G)$$

This follows the standard way to evaluate conjunctions and negations.

To factorize the proof of soundness, we first prove a few lemmas for the *invertible* rules of Alpha, that is those who satisfy $v(G) = v(H)$ for every valuation v if $G \to H$.

Lemma 9.5.1 (Iteration) *For every graph G, context $H\square$ and valuation v, we have*

$$v\big(G, H\boxed{}\big) = v\big(G, H\boxed{G}\big)$$

Proof. By induction on $|H\square|$.

Base case Suppose $H\square = H', \square$. Then we have

$$\begin{aligned} v(G, H') &= \min_{g \in G \cup H'} v(g) \\ &= \min_{g \in G \cup H' \cup G} v(g) \\ &= v(G, H', G) \end{aligned}$$

Recursive case Suppose $H = H', [H_0\square]$. Then we have

$$\begin{aligned} v\big(G, H', [H_0\boxed{}]\big) &= \min\Big(v(H'), v\big(G, [H_0\boxed{}]\big)\Big) \\ &= \min\Big(v(H'), v\big(G, [H_0\boxed{G}]\big)\Big) \quad \text{(IH)} \\ &= v\big(G, H', [H_0\boxed{G}]\big) \end{aligned}$$

□

Lemma 9.5.2 (Double-cut) *For every graph G and valuation v, we have*

$$v([[G]]) = v(G)$$

Proof.

$$v([[G]]) = 1 - (1 - v(G)) = \begin{cases} 1 - (1 - 0) = 0 & \text{if } v(G) = 0 \\ 1 - (1 - 1) = 1 & \text{if } v(G) = 1 \end{cases}$$

In both cases we have $v([[G]]) = v(G)$. ☐

Since all rules apply in an arbitrary deep context $K\square$, we will benefit from the following *functoriality* lemmas:

Lemma 9.5.3 (Variance) *For every context $K\square$, graphs G, H and valuation v such that $v(G) \leq v(H)$, we have:*

1. $v\left(K\boxed{G}\right) \leq v\left(K\boxed{H}\right)$ *if $K\square$ is positive;*
2. $v\left(K\boxed{H}\right) \leq v\left(K\boxed{G}\right)$ *if $K\square$ is negative.*

Proof. By induction on $|K\square|$.

Base case ($|K\square| = 0$)

1. Suppose $K\square = K', \square$. Then we have

$$\begin{aligned} v(K', G) &= \min(v(K'), v(G)) \\ &\leq \min(v(K'), v(H)) \quad \text{(Hypothesis)} \\ &= v(K', H) \end{aligned}$$

2. We have $|K\square| > 0$ since $K\square$ is negative. Contradiction.

Recursive case ($|K\square| > 0$)

1. Suppose $K\square = K', [K_0^- \square]$. Then by IH we have $v\left(K_0^-\boxed{H}\right) \leq v\left(K_0^-\boxed{G}\right)$, and thus $1 - v\left(K_0^-\boxed{G}\right) \leq 1 - v\left(K_0^-\boxed{H}\right)$. Therefore

$$\begin{aligned} v\left(K', [K_0^-\boxed{G}]\right) &= \min\left(v(K'), 1 - v\left(K_0^-\boxed{G}\right)\right) \\ &\leq \min\left(v(K'), 1 - v\left(K_0^-\boxed{H}\right)\right) \\ &= v\left(K', [K_0^-\boxed{H}]\right) \end{aligned}$$

2. Suppose $K\square = K', [K_0^+ \square]$. Then by IH we have $v\left(K_0^+\boxed{G}\right) \leq v\left(K_0^+\boxed{H}\right)$, and thus $1 - v\left(K_0^+\boxed{H}\right) \leq 1 - v\left(K_0^+\boxed{G}\right)$. Therefore

$$\begin{aligned} v\left(K', [K_0^+\boxed{H}]\right) &= \min\left(v(K'), 1 - v\left(K_0^+\boxed{H}\right)\right) \\ &\leq \min\left(v(K'), 1 - v\left(K_0^+\boxed{G}\right)\right) \\ &= v\left(K', [K_0^+\boxed{G}]\right) \end{aligned}$$

☐

Corollary 9.5.4 (Functoriality) *If* $v(G) = v(H)$ *then* $v\left(K\boxed{G}\right) = v\left(K\boxed{H}\right)$.

Theorem 9.5.5 (Soundness) *If* $G \to H$*, then* $v(H) \leq v(G)$ *for every valuation* v*.*

Proof. By inspection of each rule.

Iter, Deit We have $v\left(K\boxed{G, H\boxed{}}\right) = v\left(K\boxed{G, H\boxed{G}}\right)$ by Lemma 9.5.1 and functoriality.

Dcut↓, Dcut↑ We have $v\left(K\boxed{G}\right) = v\left(K\boxed{[[G]]}\right)$ by Lemma 9.5.2 and functoriality.

Ins, Del This follows from the fact that $v(G) \leq 1 = v(\varnothing)$ and Lemma 9.5.3.

□

9.6. Completeness

To show the completeness of Alpha, it is standard in the literature on EGs to simulate an existing proof system for classical propositional logic, that is itself known to be complete. For instance in [214], completeness is shown by simulating the Hilbert-style system P of Church, which only comprises 3 axioms for the (functionally complete) fragment $\{\supset, \neg\}$. We propose in this section a proof by simulation of a *deep inference* system, more specifically the calculus of structures SKS first introduced in [31]. This should provide a good overview of the similarities and differences between the two systems, and in particular of how they exemplify two distinct approaches to *symmetry* in a deep inference setting.

[31]: Brünnler et al. (2001), 'A Local System for Classical Logic'

Note

This section was written without being aware of the work of Ma and Pietarinen in [158], where they give a simulation of the calculus of structures SKSg in Alpha, and also conversely a simulation of Alpha in SKSg. While they do a similar comparison of features between the two systems, in particular concerning their treatment of symmetry and polarity, our work differs mainly in two respects:

Locality SKS is the *local* version of SKSg, thus we briefly comment on locality in SKS and Alpha and what our simulation says about it;

Analyticity crucially, our objective is to show at the end of this section that Alpha is *analytic*. Ma and Pietarinen discuss this question very quickly in their paper, by affirming that Alpha is analytic both in the sense of Gentzen because it can simulate the cut rule, and in the sense of *illative atomicity* discussed in Section 9.4. We disagree with the first claim, and use our simulation to show analyticity in the proper sense of satisfying a form of subformula property.

[158]: Ma et al. (2017), 'Proof Analysis of Peirce's Alpha System of Graphs'

9.6.1. Calculus of structures

As the name indicates, the objects manipulated by inference rules in calculi of structures are so-called *structures*. In the case of SKS, they correspond to formulas in *negation normal form* built from atoms and units $\{\top, \bot\}$, i.e. where connectives are restricted to the fragment $\{\wedge, \vee\}$, and negation is pushed to atoms by relying on De Morgan's laws.

Definition 9.6.1 (Structure) *The structures of SKS are generated by the following grammar:*

$$S, T, U, V, W ::= a \mid \bar{a} \mid \top \mid \bot \mid S \wedge T \mid S \vee T$$

Definition 9.6.2 (De Morgan dual) *The* De Morgan dual *of a structure S is defined recursively as follows:*

$$\begin{aligned} \overline{a} &= \bar{a} & \qquad \overline{\bar{a}} &= a \\ \overline{\top} &= \bot & \qquad \overline{\bot} &= \top \\ \overline{S \wedge T} &= \overline{S} \vee \overline{T} & \qquad \overline{S \vee T} &= \overline{S} \wedge \overline{T} \end{aligned}$$

It is customary in CoS to further quotient the set of structures with additional equations between them, which account for various algebraic properties of connectives such as associativity, commutativity and unitality. Here we will rely instead on the formulation of SKS given by Tubella and Straßburger in [242], where all equations are incorporated in the system as explicit rules.

The full set of rules of SKS is given in Figure 9.7. All rules are implicitly applicable in any *context* $W\square$ of arbitrary depth, with the usual notion of *context* as a structure with a hole.

Definition 9.6.3 (Depth) *The* depth $|S\square|$ *of a context $S\square$ is defined recursively as follows:*

$$|\square| = 0$$
$$|S\square \wedge T| = |T \wedge S\square| = |S\square \vee T| = |T \vee S\square| = |S\square| + 1$$

Remark 9.6.1 Contexts for structures are always *positive*, since negation is pushed down to atoms. This is the opposite situation from that of Alpha, where negation is the *only* construct that can increase the depth of a graph. This explains why some rules in Alpha need explicit indications for the polarity of the context in which they apply.

The rules of SKS satisfy two notable properties:

Symmetry Every rule r has a *dual* rule $\bar{\mathsf{r}}$, that is $U\boxed{S} \xrightarrow{\mathsf{r}} U\boxed{T}$ if and only if $U\boxed{\overline{T}} \xrightarrow{\bar{\mathsf{r}}} U\boxed{\overline{S}}$. For rules whose name ends with $\downarrow$, the dual is the rule with the same name ending with $\uparrow$. This corresponds to all rules except

Symmetric Klassisch Structures

$$\frac{\top}{a \vee \bar{a}}\ \mathsf{ai}{\downarrow} \qquad \frac{a \wedge \bar{a}}{\bot}\ \mathsf{ai}{\uparrow}$$

$$\frac{S \wedge (T \vee U)}{(S \wedge T) \vee U}\ \mathsf{s}$$

$$\frac{\bot}{a}\ \mathsf{aw}{\downarrow} \qquad \frac{a \vee a}{a}\ \mathsf{ac}{\downarrow} \qquad \frac{a}{a \wedge a}\ \mathsf{ac}{\uparrow} \qquad \frac{a}{\top}\ \mathsf{aw}{\uparrow}$$

$$\frac{\bot}{\bot \wedge \bot}\ \mathsf{nm}{\downarrow} \qquad \frac{(S \wedge T) \vee (U \wedge V)}{(S \vee U) \wedge (T \vee V)}\ \mathsf{m} \qquad \frac{\top \vee \top}{\top}\ \mathsf{nm}{\uparrow}$$

$$\frac{S \vee (T \vee U)}{(S \vee T) \vee U}\ {\downarrow} \qquad \frac{S \vee T}{T \vee S}\ {\downarrow} \qquad \frac{S \wedge T}{T \wedge S}\ {\uparrow} \qquad \frac{S \wedge (T \wedge U)}{(S \wedge T) \wedge U}\ {\uparrow}$$

$$\frac{S}{S \vee \bot}\ \mathsf{f}{\downarrow} \qquad \frac{S}{S \wedge \top}\ \mathsf{t}{\downarrow} \qquad \frac{\bot \vee S}{S}\ \mathsf{t}{\uparrow} \qquad \frac{\top \wedge S}{S}\ \mathsf{f}{\uparrow}$$

All rules apply in an arbitrary context $W\square$.

Figure 9.7.: Inference rules of SKS

the *switch* rule s and the *medial* rule m which are self-dual, i.e. $\bar{\mathsf{s}} = \mathsf{s}$ and $\bar{\mathsf{m}} = \mathsf{m}$. In Alpha, duality is captured in the *polarity* of contexts rather than through De Morgan's laws:

Fact 9.6.1 (Duality) $K^+\boxed{G} \xrightarrow{\mathsf{r}} K^+\boxed{H}$ if and only if $K^-\boxed{H} \xrightarrow{\bar{\mathsf{r}}} K^-\boxed{G}$, where

$$\overline{\mathsf{Iter}} = \mathsf{Deit} \qquad \overline{\mathsf{Deit}} = \mathsf{Iter}$$
$$\overline{\mathsf{Ins}} = \mathsf{Del} \qquad \overline{\mathsf{Del}} = \mathsf{Ins}$$
$$\overline{\mathsf{Dcut}{\downarrow}} = \mathsf{Dcut}{\uparrow} \qquad \overline{\mathsf{Dcut}{\uparrow}} = \mathsf{Dcut}{\downarrow}$$

Remark 9.6.2 Contrary to De Morgan duality, the notion of polarity of a context also exists in intuitionistic logic, and is in fact used in the same way to obtain dual rules in the intuitionistic calculus of structures SISa of Tiu [241]. This constitutes one more argument in favor of the view defended by Ma and Pietarinen in [159], that Peirce had a *pre-intuitionistic* conception of negation. It also echoes our observation in Remark 9.1.1, that the choice of negation and conjunction as primitives is to be connected with the eminently constructive works of Girard and Melliès, in particular the non-involutive tensorial negation of the latter. The issue of finding seeds of intuitionism in Peirce's work will be discussed more in depth in Section 10.2.

[159]: Ma et al. (2019), 'A graphical deep inference system for intuitionistic logic'

Locality Every rule is *local*, in the sense that it does not require the inspection of expressions of arbitrary size (Definition 2.1.1 in [242]). This is almost the opposite in Alpha: to the exception of the rules Dcut↓ and Dcut↑ (which are best seen as a structural equivalence like those of CoS), all rules depend heavily on the creation, duplication or deletion of subgraphs of arbitrary size. This is exemplified by the derivation of Peirce's law in Figure 9.3, where not a single rule is instantiated on atoms. In fact it is quite hard to see how to build a derivation of Peirce's law that performs only local transformations.

In light of this, it becomes surprising that we *will* be able to simulate SKS in Alpha. Indeed, it means that there is a set $\mathsf{Alpha}_{\mathsf{SKS}}$ of perfectly local rules on graphs, corresponding to the translation of the rules of SKS, and which is entirely derivable in Alpha. Thus by restricting oneself to the rules of $\mathsf{Alpha}_{\mathsf{SKS}}$ (and forgetting that they are derived with non-local rules), one gets a fully local subsystem of Alpha!

9.6.2. Simulation

To formulate the simulation, we need to translate the structures of SKS into equivalent graphs. This is easily done by exploiting the functional completeness of $\{\wedge, \neg\}$ (see Section 9.1):

Definition 9.6.4 (Structure translation) *The* translation $S^\bullet$ *of a structure S as a graph is defined recursively as follows:*

$$a^\bullet = a \qquad \bar{a}^\bullet = [a]$$
$$\top^\bullet = \varnothing \qquad \bot^\bullet = []$$
$$(S \wedge T)^\bullet = S^\bullet, T^\bullet \qquad (S \vee T)^\bullet = [[S^\bullet], [T^\bullet]]$$

As per Remark 9.6.1, the translation of a structure context (where the hole is translated as itself) will always be positive:

Fact 9.6.2 For every context $S\square$, $S\square^\bullet$ is positive.

Proof. By a straightforward induction on $|S\square|$. □

It is easy to show that a structure and its translation as a graph are semantically equivalent, i.e. $v(S) = v(S^\bullet)$ for any valuation v. Thus to get the completeness of Alpha, it is sufficient to simulate the translation of each rule of SKS. But first, we need to ensure that Alpha satisfies a property of *contextual closure*: this will allow us to ignore the implicit context $W\square$ in the rules of Figure 9.7.

Lemma 9.6.1 (Positive closure) *If $G \to H$, then $K^+\boxed{G} \to K^+\boxed{H}$.*

Proof. Since all rules of Alpha apply in a context of unbounded depth, we know that there are some graphs G_0, H_0 and context $K'\square$ such that $G = K'\boxed{G_0}$ and $H = K'\boxed{H_0}$. Then either $K'\square$ is positive, and $\mathsf{inv}(K^+\boxed{K'\square}) = \mathsf{inv}(K^+\square) + \mathsf{inv}(K'\square)$ is even since it is the sum of two even numbers; or $K'\square$ is negative, and $\mathsf{inv}(K^+\boxed{K'\square})$ is odd since it is the sum of an even and an odd number. In both cases $K^+\boxed{K'\square}$ has the same polarity as $K'\square$, and thus the same rule can be applied. □

Theorem 9.6.2 (Completeness) *If* $U\boxed{S} \to U\boxed{T}$, *then* $U^\bullet\boxed{S^\bullet} \to^* U^\bullet\boxed{T^\bullet}$.

Proof. We show that $S^\bullet \to^* T^\bullet$ by simulating each rule of Figure 9.7. The closure with $U\square^\bullet$ follows from Fact 9.6.2 and Lemma 9.6.1.

To make the notation lighter, we implicitly apply the translation $(-)^\bullet$ on substructures. We also add some coloring to put clearly in evidence the subgraphs manipulated by rules. Assuming that the rules are read from bottom to top:

(De)iteration In the rules Iter and Deit, the justifying occurrence is squared in blue. In the Iter (resp. Deit) rule, the erased (resp. space for the inserted) copy is highlighted in blue.

Insertion/Deletion In the rule Ins (resp. Del), the erased (resp. space for the inserted) subgraph is highlighted in red.

Double-cut In the rule Dcut↓ (resp. Dcut↑), the space around which the double-cut is erased (resp. inserted) is highlighted in gray.

We start with the identity rules $\{\mathsf{i}{\downarrow}, \mathsf{i}{\uparrow}\}$:

$$\frac{\top}{a \vee \bar{a}}\,\mathsf{ai}{\downarrow} \quad \mapsto \quad \cfrac{\cfrac{\cfrac{\cfrac{}{[[\;]]}\,\mathsf{Dcut}{\downarrow}}{[[],a]}\,\mathsf{Ins}}{[[a],\boxed{a}]}\,\mathsf{Iter}}{[[a],[[a]]]}\,\mathsf{Dcut}{\downarrow} \qquad\qquad \frac{a \wedge \bar{a}}{\bot}\,\mathsf{ai}{\uparrow} \quad \mapsto \quad \cfrac{\cfrac{a,[a]}{\boxed{a},[\;]}\,\mathsf{Deit}}{[]}\,\mathsf{Del}$$

Then onto weakening {aw↓, aw↑} and contraction {ac↓, ac↑, nm↓, nm↑}:

$$\frac{\bot}{a}\,\mathsf{aw}\!\downarrow \quad\mapsto\quad \cfrac{\cfrac{[\,]}{[\,[a]\,]}\,\mathsf{Ins}}{a}\,\mathsf{Dcut}\!\uparrow \qquad\qquad \frac{a}{\top}\,\mathsf{aw}\!\uparrow \quad\mapsto\quad \frac{a}{\quad}\,\mathsf{Del}$$

$$\frac{a \vee a}{a}\,\mathsf{ac}\!\downarrow \quad\mapsto\quad \cfrac{\cfrac{[[a],[a]]}{[\,\boxed{[a]}\quad\,]}\,\mathsf{Deit}}{a}\,\mathsf{Dcut}\!\uparrow \qquad\qquad \frac{a}{a \wedge a}\,\mathsf{ac}\!\uparrow \quad\mapsto\quad \frac{a}{\boxed{a},\ a}\,\mathsf{Iter}$$

$$\frac{\bot}{\bot \wedge \bot}\,\mathsf{nm}\!\downarrow \quad\mapsto\quad \frac{[\,]}{\boxed{[\,]},\ [\,]}\,\mathsf{Iter} \qquad\qquad \frac{\top \vee \top}{\top}\,\mathsf{nm}\!\uparrow \quad\mapsto\quad \cfrac{\cfrac{[[\,],[\,]]}{[\,\boxed{[\,]}\quad\,]}\,\mathsf{Deit}}{\quad}\,\mathsf{Dcut}\!\uparrow$$

For the switch rule s, we give two dual derivations: the first uses the rules Deit and Ins to move U *into* the cuts enclosing T, while the second uses the rules Del and Iter to move S *out of* the cuts of T.

$$\cfrac{\cfrac{\cfrac{\cfrac{S,[[T],[U]]}{[[\ S,[[T],[U]]\ ]]}\,\mathsf{Dcut}\!\downarrow}{[[S,[[T],[U]]],\ [U]\ ]}\,\mathsf{Ins}}{[[S,[[T\quad]],\boxed{[U]}]]}\,\mathsf{Deit}}{[[S,\ T\ ],[U]]}\,\mathsf{Dcut}\!\uparrow \qquad\qquad \cfrac{\cfrac{S,[[T],[U]]}{\boxed{S},[[\ S\ ,T],[U]]}\,\mathsf{Iter}}{\quad[[S,T],[U]]}\,\mathsf{Del}$$

Similarly for the medial rule m, which is the other self-dual rule of SKS, we have two dual derivations:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{[[S,T],[U,V]]}{[[S,T],[U,[[\ V\ ]]]]}\,\mathsf{Dcut}\!\downarrow}{[[S,[[\ T\ ]]],[U,[[V]]]]}\,\mathsf{Dcut}\!\downarrow}{[[S,[[T]]],[U,[\ [T]\ ,[V]]]]}\,\mathsf{Ins}}{[[S,[[T],\ [V]\ ]],[U,[[T],[V]]]]}\,\mathsf{Ins}}{[[S,[[T],[V]]],[U,[[T],[V]]]],\ [[T],[V]]}\,\mathsf{Ins}}{[[S,[[T],[V]]],[U\qquad]],\boxed{[[T],[V]]}}\,\mathsf{Deit}}{[[S\qquad],[U]],\boxed{[[T],[V]]}}\,\mathsf{Deit}$$

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{[[S,T],[U,V]]}{\boxed{[[S,T],[U,V]]},\ [[S,T],[U,V]]}\,\mathsf{Iter}}{[[S,T],[U,V]],[[S,T],[\quad V]]}\,\mathsf{Del}}{[[S,T],[U,V]],[[\quad T],[V]]}\,\mathsf{Del}}{[[S,T],[U\quad]],[[T],[V]]}\,\mathsf{Del}}{[[S\quad],[U]],[[T],[V]]}\,\mathsf{Del}$$

The other rules correspond to equations on structures: α for *associativity*, σ for *commutativity*, and f and t for unitality. Note that all rules involving $\wedge$ and $\top$ are trivially simulated by the isotropy of SA. Simulating the other rules only requires the double-cut rules, substantiating our claim (based on Peirce's own view, see Section 9.4) that the latter should be seen as expressing a structural equivalence, rather than as *bona fide* inference rules.

$$\dfrac{S \vee (T \vee U)}{(S \vee T) \vee U}\,\alpha{\downarrow} \quad\mapsto\quad \dfrac{\dfrac{[[S],[[[T],[U]]]]}{[[S],\ [T],[U]\ ]}\,\mathsf{Dcut}{\uparrow}}{[[[\ [S],[T]\ ]],[U]]}\,\mathsf{Dcut}{\downarrow} \qquad\qquad \dfrac{S \wedge (T \wedge U)}{(S \wedge T) \wedge U}\,\alpha{\uparrow} \quad\mapsto\quad S,T,U$$

$$\dfrac{S \vee T}{T \vee S}\,\sigma{\downarrow} \quad\mapsto\quad [[S],[T]] \qquad\qquad \dfrac{S \wedge T}{T \wedge S}\,\sigma{\uparrow} \quad\mapsto\quad S,T$$

$$\dfrac{S}{S \vee \bot}\,\mathsf{f}{\downarrow} \quad\mapsto\quad \dfrac{\dfrac{S}{[[\ S\ ]]}\,\mathsf{Dcut}{\downarrow}}{[[S],[[\ \ ]]]}\,\mathsf{Dcut}{\downarrow} \qquad\qquad \dfrac{\top \wedge S}{S}\,\mathsf{f}{\uparrow} \quad\mapsto\quad S$$

$$\dfrac{S}{S \wedge \top}\,\mathsf{t}{\downarrow} \quad\mapsto\quad S \qquad\qquad \dfrac{\bot \vee S}{S}\,\mathsf{t}{\uparrow} \quad\mapsto\quad \dfrac{\dfrac{[[[]],[S]]}{[[\ \ S]]}\,\mathsf{Dcut}{\uparrow}}{S}\,\mathsf{Dcut}{\uparrow}$$

□

9.6.3. Analyticity

A powerful result of Brünnler and Tiu [31], is that the whole up-fragment of SKS (all rules whose name ends with ↑) is *admissible*: if a structure S has a *proof* $S \to^* \top$, then it also has a proof $S \xrightarrow{\mathsf{KS}}{}^* \top$, with KS defined as SKS without the up-fragment[17]. Dually, the whole down-fragment (all rules whose name ends with ↓) is *"co-admissible"*: if a structure S has a *refutation* $\bot \to^* S$, then it also has a refutation $\bot \xrightarrow{\overline{\mathsf{KS}}}{}^* S$, with $\overline{\mathsf{KS}}$ defined as SKS without the down-fragment.

17: The name KS comes from SKS, with the first 'S' standing for "symmetric" dropped.

This duality reflects nicely in our simulation: we were careful to always give derivations for up-rules that mirror closely those for down-rules, modulo the use of the double-cut principle. Roughly, if the simulation of $S \xrightarrow{r} T$ has the shape $S^\bullet \xrightarrow{r_1} \dots \xrightarrow{r_n} T^\bullet$, then the simulation of $\overline{T} \xrightarrow{\bar{r}} \overline{S}$ has the shape $\overline{T}^\bullet \xrightarrow{\overline{r_n}} \dots \xrightarrow{\overline{r_1}} \overline{S}^\bullet$ (see Fact 9.6.1). An important consequence is that the deletion rule Del is never used in the simulation of KS, if one chooses the appropriate derivation among the two provided for the switch and medial rules s and m. Thus deletion is admissible in Alpha, a result that seems to be novel in the literature on EGs.

Corollary 9.6.3 (Admissibility of Deletion)

$$\text{If } G \to^* \varnothing, \text{ then } G \xrightarrow{\mathsf{Alpha}\setminus\{\mathsf{Del}\}}{}^* \varnothing.$$

Dually, the insertion rule Ins is never used in the simulation of $\overline{\mathsf{KS}}$, implying the co-admissibility of insertion. In fact there is a curious dissymmetry, in that the rule Dcut↓ of *double-cut* insertion also never appears in the simulation of $\overline{\mathsf{KS}}$, while Dcut↑ is used multiple times in the simulation of KS:

Corollary 9.6.4 (Co-admissibility of Insertion)

$$\text{If } [] \to^* G, \text{ then } [] \xrightarrow{\mathsf{Alpha}\setminus\{\mathsf{Ins},\mathsf{Dcut}\downarrow\}}{}^* G.$$

Corollary 9.6.3 is what allows us to conclude that Alpha is an *analytic* system, in a sense very close to that of Gentzen. Because we do not have a notion of logical connective nor tree-shaped derivations, we must reduce the subformula property to *atomic* graphs. Then it is easy to see that all rules of Alpha except Del satisfy this property:

Definition 9.6.5 (Subgraph) *A graph G is a* subgraph *of a graph H, written $G \prec H$, if there exists a context $K\square$ such that $H = K\boxed{G}$.*

Fact 9.6.3 For every $a \in \mathscr{A}$, if $G \xrightarrow{\mathsf{Alpha}\setminus\{\mathsf{Del}\}} H$ and $a \prec H$ then $a \prec G$.

Corollary 9.6.5 (Analyticity) *If G is provable in Alpha, then it has a proof $G \to G_1 \to \ldots \to G_n \to \varnothing$ where $a \prec G$ for all i and $a \in \mathscr{A}$ such that $a \prec G_i$.*

9.7. Beta graphs

Before working on EGs, Peirce had already developed a deep understanding of the logic of relations of arbitrary arity, inventing the notions of variables and quantifiers 30 years before the standard Russell-Whitehead syntax for predicate logic appeared in 1910 [33]. This all stemmed from his extensive study of *relation algebras*, first investigated by De Morgan in 1860. However in his system Beta of EGs, Peirce gives a very different account of the logic of relations, both in the graphical representation of relational statements, and the illative transformations that govern them. In the following, we illustrate informally the principles of Beta, and how they are able to capture what is identified nowadays in symbolic logic as *purely relational* first-order theories (that is, without constant nor function symbols) equipped with a primitive equality predicate.

[33]: Burch (2022), 'Charles Sanders Peirce'

9.7.1. Syntax

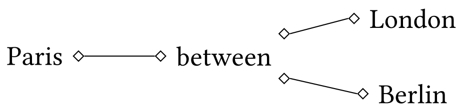
Spots In the propositional system Alpha, atomic graphs represent sentences that can be asserted or denied (or equivalently, assigned a truth value), but they do not exhibit any internal structure syntactically: they might as well be depicted just as (distinguished) points on SA. As most logicians of his time, Peirce was directly influenced by the *term logic* of Aristotle, where assertions are decomposed into a *subject* to which applies some *predicate*. However, while Aristotle's notion of predicate has a metaphysical flavor, Peirce's notion is purely grammatical. For instance, Aristotle rejected the sentence "The person sitting down is Socrates" as a genuine predication, because Socrates is an *individual*, and in his view predicates could only be so-called *universals* like "humans" or "mortals" [224]. In Beta there is no such restriction, and the previous sentence can be represented by the following graph:

[224]: Smith (2022), 'Aristotle's Logic'

$$\text{The person sitting down} \;\diamond\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\diamond\; \text{Socrates} \tag{9.1}$$

Here both "The person sitting down" and "Socrates" are modelled as unary predicates. The little diamonds, called *hooks* by Peirce, represent placeholders for the arguments (subjects) of each predicate, and the data of a predicate together with its hooks is called a *spot*[18]. Any predicate of arity n can then be represented by a spot with n hooks disposed freely around its periphery. For instance, the sentence "Paris is between London and Berlin" can be expressed by the graph

18: The use of little diamonds to depict hooks is our own addition, Peirce never drew them explicitly.



where "between" is modelled as a ternary predicate.

Lines of Identity To assert that there exists an individual who is both the person sitting down and Socrates in graph (9.1), we connect the two hooks with a so-called *line of identity* (LoI). In Peirce's view, each point in a LoI denotes an individual of the universe represented by SA. Since scribing anything on SA means asserting its truth in the universe, then it suffices to reduce the truth of an individual to its existence, in order to interpret the marking of LoIs as having existential force. This is actually the origin of the "existential" qualificative in the denomination "existential graph". Note that no information is given but the individual's existence: in particular, two distinct points on SA may or may not denote two distinct individuals, just as two distinct variables x and y in predicate calculus may or may not refer to the same object. It is the *continuity* of a LoI that signifies, in an iconic way, the identity of every point/individual constituting, and connected by the line. Hence the following graph

$$\text{The person sitting down} \;\diamond\!\!-\qquad -\!\!\diamond\; \text{Socrates}$$

expresses that both the person sitting down and Socrates exist, but we do not know whether they are the same individual.

First-order logic "The person sitting down is Socrates" might be equivalently expressed in a first-order language as the formula

$$\exists x.\mathrm{PersonSittingDown}(x) \wedge \mathrm{Socrates}(x) \tag{9.2}$$

LoIs can then be seen as encoding the concept of *existential quantification* over a single *variable* x, where the occurrences of x correspond to the extremities of the line connected to the hooks. As in Alpha, the conjunctive aspect of the sentence is accounted for by the fact that the two spots are juxtaposed in the same area on SA.

Now if one considers a first-order theory with a predicate symbol $=$ satisfying the usual axioms for equality, then our sentence can alternatively be expressed by the following formula:

$$\exists x.\exists y.\mathrm{PersonSittingDown}(x) \wedge \mathrm{Socrates}(y) \wedge x = y \tag{9.3}$$

9.7.2. Deconstructing identity

In Peirce's original notation, there is no way to distinguish between the two formulations (9.2) and (9.3), as they would both be represented by graph (9.1). However in [115], in order to have a rigorous interpretation of the syntax of Beta in category theory, the authors propose to analyze LoIs into essentially two distinct icons, *binders* and *teridentities*[19], from which every LoI can be reconstructed:

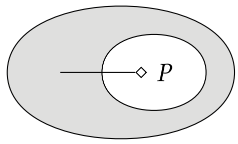
[115]: Haydon et al. (2020), 'Compositional Diagrammatic First-Order Logic'

19: This is our own terminology, chosen mainly for historical reasons. In [115], binders and teridentities correspond to the generators of the monoid-comonoid pair of a Frobenius algebra.

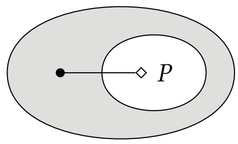
Binder As we have just seen, one function of LoIs is to *quantify* over individuals. For now we have only considered LoIs located at the top-level of SA, i.e. in a *positive* area, where they are given *existential* force. If we were to negate the graph (9.1) by enclosing it in a cut, we would get a graph expressing the negation of formula (9.2), which by De Morgan duality is equivalent to

$$\forall x.\neg\mathrm{PersonSittingDown}(x) \vee \neg\mathrm{Socrates}(x)$$

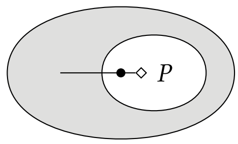
More generally, it is well-known that in classical logic, *universal* quantification can be defined symbolically by $\forall x.P(x) \triangleq \neg\exists x.\neg P(x)$, and hence that any formula in the usual language of FOL has a classically equivalent formula in the fragment $\{\neg, \wedge, \exists\}$. This is precisely how Peirce expresses universal quantification in Beta, as illustrated for a unary predicate P by the graph (9.4) of Figure 9.8. But in order to interpret correctly this graph, one needs to adopt what Peirce calls an *endoporeutic*[20] reading: one should start inspecting the graph from the *top-level* of SA, and then descend (recursively) into its various cuts. In particular, the location of a LoI should be identified with its *outermost* end, as illustrated in graph 9.5; if we were to associate it instead with its innermost end as in graph (9.6), then we would swap the positions of $\exists$ and $\neg$ in the associated statement, giving the non-equivalent formula $\neg\neg\exists x.P(x)$.



(9.4)



(9.5)



(9.6)

Figure 9.8.: Universal quantification $\forall x.P(x)$ in Beta

From these observations, we get that the type of quantification prformed by a LoI is fully captured by its location: existential in a positive area, universal in a negative area. This leads us to analyze the syntax of

20: From the greek *endon* ('within') and *poros* ('passage, pore') [203], literally "that lets through within". This physically-flavored terminology is reminiscent of the intuition we developed within our own bubble calculus and illustrated in Figure 7.1, even though we were not aware of the existence of EGs at the time!

LoIs into two components: so-called *binders* that encode *quantifiers* as distinguished heavy dots on SA like those of Figure 9.8[21]; and two-ended *wires* that connect binders to the hooks of predicates, encoding the *identity* between a bound variable x and an occurrence of x.

21: Peirce liked to insist that LoIs should be drawn as *heavy* lines, to distinguish them from the normal lines used to depict cuts.

Remark 9.7.1 The graph of universal quantification (9.5) bears a striking similarity to that of implication, as illustrated in Figure 9.9. While the similarity also exists for the symbolic encoding of these connectives in the fragment $\{\neg, \wedge, \exists\}$ captured by Beta, the graphical representation makes this fact more apparent. In constructive type theories, both universal quantification $\forall x.B$ and implication $A \supset B$ are seen as instances of a more general construct, the *dependent product type* $\Pi x : A.B$. Thus retrospectively, one might interpret the above observation as another manifestation of Peirce's pre-intuitionistic conception of logic.

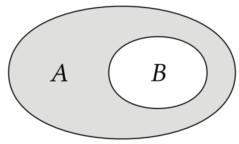
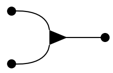
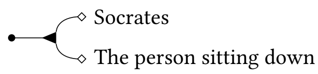


Figure 9.9.: Implication $A \supset B$ in Alpha

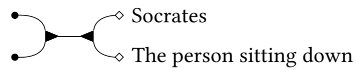
Teridentity Binders and wires are sufficient to express unary predicates applied to distinct bound variables, but they cannot identify multiple occurrences of the *same* variable, and thus cannot account for graph (9.1). To palliate this, Peirce used a construct that he called *teridentity*, that we propose to represent as a black triangle ►. The three vertices of the triangle should be seen as three *plugs* on which one can connect wires, so that the graph



can be interpreted as the formula $\exists x.\exists y.\exists z.x = y \wedge y = z$. Rather than a way to express equality between variables, we think it is more useful however to have an *operational* understanding of teridentities: their real purpose is to *duplicate* wires (e.g. by splitting them), or dually to *merge* two wires into a single one. This gives a constructive way to explain the notion of *occurrence* of a variable[22]. Then formulas (9.2) and (9.3) can be represented faithfully and respectively by the graphs (9.7) and (9.8) of Figure 9.10.



(9.7)



(9.8)

Figure 9.10.: Decomposing lines of identity

22: A very similar metaphor is brought up by Girard in [94], where the fact that a variable can have infinitely many occurrences is seen precisely as a consequence of the possibility to split or "debit" indefinitely a wire into smaller wires.

LoIs (or their decomposition into the above constructs) constitute the only icons introduced in Beta compared to Alpha.

Iconic atomicity The fact that Peirce took LoIs to be a primitive, unanalyzed icon can be seen as a consequence (or a cause?) of his view that only *closed* sentences should be considered. Indeed in an early exposition of EGs (Ms 493), he proposed a way to show, if necessary, that "a complete assertion is not intended" [214, p. 49]. That is, he devised a syntax equivalent in purpose to that of *free variables* in predicate calculus. Our analysis into wires that connect hooks, binders and teridentities allows this, and more generally makes the syntax of graphs closer to predicate calculus. But in later expositions, Peirce always required every hook to be filled with a LoI, that is every variable to be quantified.

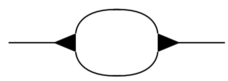


Figure 9.11.: Building a two-ended wire from two teridentities

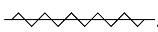
Now, it is unclear which of those presentations is the more "analytical" or primitive, when restricting oneself to closed (or in Peirce's terminology, complete) assertions. Indeed, consider that we give the force of quantification to wires, as Peirce does with LoIs: then one can replace every binder by a dangling wire, rendering binders useless in the syntax. Also since empty hooks and teridentities are forbidden, they will always be filled with wires. Then couldn't we just reduce the full syntax of LoIs to wires?

A remarkable insight of Peirce, is that one cannot build the concept of teridentity from two-ended wires, but that the converse *is* possible, as illustrated in Figure 9.11. This is most clearly (and speculatively) understood by seeing LoIs as made out of *pipes* rather than wires. Indeed, just gluing one extremity of a two-ended pipe `P` to the exterior of another pipe `Q` will not make `P` and `Q` communicate; and there is no reason to interpret the joining, or (infinitesimally) close juxtaposition of two lines on SA at one point, as doing more than the gluing of two pipes.

One solution would be to first drill a hole in `Q`, before glueing `P` on it. Another is to have *branching* pipes as the basic building blocks for our plumbing, i.e. teridentities. This is how we interpret and justify metaphorically the following enigmatic quote from Peirce [214, p. 116]:

> Teridentity is not mere identity. It is identity and identity, but this 'and' is a distinct concept [from that denoted by the juxtaposition of graphs on SA], and is precisely that of teridentity.

Then every LoI can be built out of teridentities, which Peirce expressed like so [214, p. 117]:

> Every line of identity ought to be considered as bristling with microscopic points of teridentity; so that when magnified shall be seen to be .

Digression

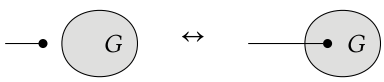
If one wants to keep the wire metaphor, joining two LoIs can be interpreted as just putting in contact two wires. Then electrical current can flow from one wire to the other, but the problem resides elsewhere, at the *illative* level: indeed nothing prevents us from separating back the two wires, or connecting two initially disjoint wires. But in a negative (resp. positive) context, the former (resp. latter) action corresponds to forgetting a possibly necessary equality hypothesis between two variables (resp. identifying two possibly distinct individuals), which is not valid logically speaking. To prevent this, one needs to do more than just put the two wires in contact with eachother (juxtaposition), i.e. solder them together (teridentity).

9.7.3. Rules

A remarkable fact about Beta is that it does not need any new illative principle compared to Alpha. Rather, it simply generalizes those of Alpha to account for LoIs[23].

23: Thus in a sense, the first-order individuals denoted by LoIs behave in the exact same way as the propositions denoted by the graphs of Alpha. This can be interpreted as a manifestation of Peirce's *psycho-physical monism* [33], that blurs the distinction between the psychological level of propositions (concepts) and the physical level of individuals (objects) enforced in the language of predicate logic, and inherited from Aristotle's metaphysical conception of predication. Independently, Girard has recently been pushing the idea further in his transcendental syntax programme, by proposing to see individuals as particular kinds of *linear* propositions (see [76, §84.3]). And what is more linear than a line?

Iteration/Deiteration When a spot is iterated, every hook of the new copy must be connected to the same hook of the original copy with a LoI. Conversely, when a spot is deiterated, every LoI of the deleted copy must be retracted to the corresponding hook of the original copy. This applies in particular to any binder seen as a unary spot, which allows to extend (resp. retract) any LoI inside (resp. outside) a cut:



Insertion/Deletion Every pair of binders residing in the same negative area can be connected and replaced by a wire (Insertion):

Dually, every wire in a positive area can be severed in two, capping off the newly created ends with two binders (Deletion):

By reading the rules from right to left — i.e. in *proof search* mode, Insertion and Deletion on LoIs can be understood as capturing respectively the operations of *anti-unification* and *unification* on two variables. That is:

Unification adding a wire between two disconnected binders x and y is equivalent in purpose to substituting x for y (resp. y for x) in every spot/predicate connected by a LoI to y (resp. x);

Anti-unification while severing a wire connected to a binder z in two parts capped by binders x and y amounts to partitioning the set of spots connected to z in two sets $\{P_i\}$ and $\{Q_j\}$, and substituting x for z in every P_i, and y for z in every Q_j.

Unification and anti-unification are the heart of many (semi-)decision procedures implemented in automated and interactive theorem provers, including the unification of subformulas in our own approach to SFL (Subsection 3.2.2); thus it is remarkable that they constitute a core illative principle of Beta.

Remark 9.7.2 In the original Beta system, Peirce enforces the usual model-theoretic assumption that the universe of discourse must be non-empty — i.e. contain at least one individual, through an *axiom* permitting to "scribe a heavy dot or unattached line on SA" [214, p. 47]. In fact together with the axiom allowing to assert the blank SA, which was implicit in our notion of proof for Alpha (Definition 9.3.7), these are the only axioms in all systems of EGs. This is yet another striking similarity with Girard's philosophy, who attempted to get rid of axioms in logic starting with ludics — although in many of his writings, he actively criticizes the non-empty model assumption.

Figure 9.12 gives a proof of the famous syllogism from Aristotle in Beta, by reducing the graph associated to the formula

$$\begin{aligned}&\forall x.\mathsf{Socrates}(x) \wedge \mathsf{Human}(x) \wedge (\forall y.\mathsf{Human}(y) \supset \mathsf{Mortal}(y)) \supset \\ &\exists z.\mathsf{Socrates}(z) \wedge \mathsf{Mortal}(z)\end{aligned} \tag{9.9}$$

to the empty SA. Again, we invert the direction of arrows in inference steps, to follow the proof search reading of rules. Note that in many steps, we add or remove some teridentities and binders without further justification: these correspond to splits, merges and rewirings of LoIs, and a more rigorous set of equations describing these operations can be found in [115, Section 3: "The algebra of lines of identity"].

The essence of the syllogism lies in the instantiation of the universally quantified variable y by x in formula (9.9), captured by the Deletion step in Figure 9.12. Thus contrary to Alpha, it seems that Deletion is not admissible in Beta anymore. Because of the *subterm property* of first-order logic [60], this should not break analyticity: indeed we should only need

[60]: Degtyarev et al. (2001), 'The Inverse Method'

Figure 9.12.: A proof of a famous syllogism in Beta

Deletion on LoIs, which connects already existing binders. However one still needs to find out the binders that must be connected, which we conjecture to be a major factor in the undecidability of first-order logic.

9.8. Gardens

Ergonomy of LoIs Overall, the example of Figure 9.12 demonstrates how Beta is particularly well-suited to analyze the fine structure of relational reasoning: be it at the level of *statements*, with the complex circuits resulting from the composition of LoIs; or at the level of *proofs*, with a decomposition of such a simple syllogism into 8 distinct inferential steps. While this is satisfying from the standpoint of meta-logical investigation originally pursued by Peirce, this syntax seems to be too cumbersome to form the basis for a practical theorem proving interface, where the user would perform illative transformations through direct manipulation of graphs. In the words of Peirce himself [202, p. 544]:

> There are a number of deduced liberties of transformation, by which even much more complicated inferences than a syllogism can be performed at a stroke. For that sort of problem, however, which consists in drawing a conclusion or assuring oneself of its correctness, this System is not particularly adapted.

Variables Still, the complexity of Beta from a UX perspective stems mostly from the tedious management of LoIs. Our analysis into binders and teridentities gives us a hint towards the solution: since we now have binders, why not just replace the complex circuits of teridentities by *variables*? The process is simple:

1. Take any complex LoI connected to n binders and m hooks;
2. Among the n binders, choose one that occurs in the outermost area of SA, and give it a fresh name x;
3. Replace the m wires connected to the hooks by m occurrences of x;
4. Erase all remaining wires and teridentities.

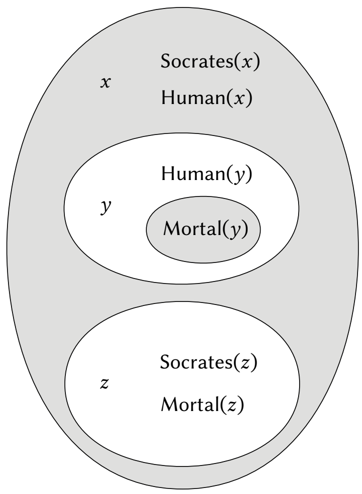


Figure 9.13.: Using variables in EGs

Thus for instance, both the first and second graphs in Figure 9.12 would be represented by the graph of Figure 9.13. In fact, Peirce already had the idea to use a name-based syntactic device similar to, and arguably more primitive than variables, which he called *selectives*, in order to avoid the ambiguity of LoIs crossing cuts [202, p. 531]:

> A Ligature crossing a Cut is to be interpreted as unchanged in meaning by erasing the part that crosses to the Cut and attaching to the two Loose Ends so produced two Instances of a Proper Name nowhere else used; such a Proper name (for which a capital letter will serve) being termed a *Selective*.

The idea of connecting two locations by marking them with the same symbol is quite natural, and is implemented for instance in *footnotes* — or in this thesis, sidenotes — with the help of *numbers* rather than capital letters. Footnotes are a good example, because contrary to variables, they share with selectives a *linearity* property, that exactly *two* occurrences of the symbol must be present. This is necessary to simulate accurately a two-ended wire, and it is not surprising that the same device has been used recently (under the name of *ports*) to give an algebraic syntax to *interaction nets* [70], a model of computation inspired by *linear* logic and its graphical, string-diagram like syntax of *proof nets* [92].

[70]: Dorman et al. (2013), 'A Hierarchy of Expressiveness in Concurrent Interaction Nets'

Bridges After introducing selectives, Peirce further remarks on the next page:

> In order to avoid the intersection of Lines of Identity, either a Selective may be employed, or a *Bridge*, which is imagined to be a bit of paper ribbon.

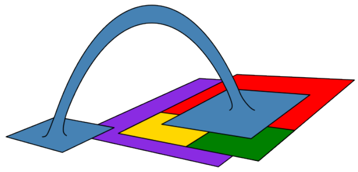


Figure 9.14.: A depiction of Peirce's Bridge for lines of identity

Source: `https://commons.wikimedia.org/wiki/File:4CT_Inadequacy_Explanation.svg`

Thus he already identified the problem of readability stemming from having too many wires crossing eachother, a well-known concern in the design of graphical programming languages[24]. The proposed alternative solution of having so-called *bridges* is quite interesting, in that it makes the syntax of EGs *three-dimensional*, in order to preserve the continuity of lines. A nice illustration of the bridge is given in Figure 9.14. We found this picture in the Wikipedia article of the *four color theorem* [252], which is no coincidence according to Burch [33]:

24: This is to be opposed to critics of the syntax of EGs such as Quine, who devised a notation similar to LoIs, and deemed it "too cumbersome for practical use" [214, p. 125].

> Peirce began to research the four-color map conjecture, to work on the graphical mathematics of de Morgan's associate A. B. Kempe, and to develop extensive connections between logic, algebra, and topology, especially topological graph theory. Ultimately these researches bore fruit in his existential graphs [...]

Multisets The Wikipedia article on Alfred Kempe also mentions the following interesting fact [251]:

> Kempe (1886) revealed a rather marked philosophical bent, and much influenced Charles Sanders Peirce. Kempe also discovered what are now called multisets, although this fact was not noted until long after his death [104, 141].

[104]: Grattan-Guinness (2000), *The Search for Mathematical Roots, 1870-1940: Logics, Set Theories and the Foundations of Mathematics from Cantor through Russell to Godel*

[141]: Kempe (1886), 'A memoir on the theory of mathematical form'

As it turns out, we can also give a multiset formalization of the syntax of graphs in Beta extending that of Section 9.3, and based on the previous idea of replacing teridentities by variables. Every area will now be equipped with a set of binders, in addition to the multiset of nodes (i.e. atoms and cuts). Anticipating our flower metaphor of Chapter 10, we call sets of binders *sprinklers*, which can be imagined as irrigating spots on their hooks by sending water through the (now invisible) LoIs, seen as hoses. Naturally, the pair formed by a sprinkler and a multiset of nodes is called a *garden*.

BETA

$$\dfrac{K\boxed{G, H\boxed{}}}{K\boxed{G, H\boxed{G}}}\;\textsf{Iter}\qquad\qquad \dfrac{K\boxed{G, H\boxed{G}}}{K\boxed{G, H\boxed{}}}\;\textsf{Deit}$$

$$\dfrac{K^{-}\boxed{}}{K^{-}\boxed{G}}\;\textsf{Ins}\qquad\qquad \dfrac{K^{+}\boxed{G}}{K^{+}\boxed{}}\;\textsf{Del}$$

$$\dfrac{K^{+}\boxed{[\mathbf{x}, y\cdot G]}}{K^{+}\boxed{[\mathbf{x}\cdot G\{z/y\}]}}\;\textsf{Unif}\!\downarrow\qquad\qquad \dfrac{K^{-}\boxed{[\mathbf{x}\cdot G\{z/y\}]}}{K^{-}\boxed{[\mathbf{x}, y\cdot G]}}\;\textsf{Unif}\!\uparrow$$

$$\dfrac{K\boxed{G}}{K\boxed{[\cdot[\cdot G]]}}\;\textsf{Dcut}\!\downarrow\qquad\qquad \dfrac{K\boxed{[\cdot[\cdot G]]}}{K\boxed{G}}\;\textsf{Dcut}\!\uparrow$$

Figure 9.15.: Inductive presentation of the rules of Beta

Definition 9.8.1 (Graph) *Given a denumerable set of variables $\mathscr{V}$ and a denumerable set of predicate symbols $\mathscr{P}$ together with their arities $\mathsf{ar} : \mathscr{P} \to \mathbb{N}$, the sets of nodes $\mathbf{N}_\beta$, gardens $\mathbf{\Gamma}_\beta$ and graphs $\mathbf{G}_\beta$ are defined mutually inductively as follows:*

- ***(Spot)*** *If $p \in \mathscr{P}$ with $\mathsf{ar}(p) = n$, then $p(x_1, \ldots, x_n) \in \mathbf{N}_\beta$;*
- ***(Graph)*** *If $G \subset \mathbf{N}_\beta$ is a finite multiset, then $G \in \mathbf{G}_\beta$.*
- ***(Garden)*** *If $\mathbf{x} \subset \mathscr{V}$ is a finite set and $G \subset \mathbf{N}_\beta$ a finite multiset, then $\mathbf{x}\cdot G \in \mathbf{\Gamma}_\beta$;*
- ***(Enclosure)*** *If $\gamma \in \mathbf{\Gamma}_\beta$, then $[\gamma] \in \mathbf{N}_\beta$.*

Example 9.8.1 The graph of Figure 9.13 can be written in textual notation as the following expression:

$$\begin{array}{l}[x\cdot \mathrm{Socrates}(x), \mathrm{Human}(x),\\ \;[y\cdot \mathrm{Human}(y), [\cdot \mathrm{Mortal}(y)]],\\ \;[z\cdot \mathrm{Socrates}(z), \mathrm{Mortal}(z)]]\end{array}$$

Note that the '$\cdot$' operator for constructing gardens has lower precedence than the ',' operator for juxtaposition of graphs. Thus the expression $x\cdot \mathrm{Socrates}(x), \mathrm{Human}(x)$ is to be read as $x\cdot(\mathrm{Socrates}(x), \mathrm{Human}(x))$, and not $(x\cdot \mathrm{Socrates}(x)), \mathrm{Human}(x)$ (which would be ill-typed anyway).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[\cdot[\cdot]]}\;\textsf{Dcut}\!\downarrow}{[x\cdot[\cdot]]}\;\textsf{Unif}\!\uparrow}{\begin{array}{l}[x\cdot \mathrm{Socrates}(x), \mathrm{Mortal}(x),\\ \;[\cdot]]\end{array}}\;\textsf{Ins}}{\begin{array}{l}[x\cdot \mathrm{Socrates}(x), \mathrm{Mortal}(x),\\ \;[\cdot \mathrm{Socrates}(x), \mathrm{Mortal}(x)]]\end{array}}\;\textsf{Iter}}{\begin{array}{l}[x\cdot \mathrm{Socrates}(x), \mathrm{Mortal}(x),\\ \;[z\cdot \mathrm{Socrates}(z), \mathrm{Mortal}(z)]]\end{array}}\;\textsf{Unif}\!\uparrow}{\begin{array}{l}[x\cdot \mathrm{Socrates}(x),\\ \;[\cdot[\cdot \mathrm{Mortal}(x)]],\\ \;[z\cdot \mathrm{Socrates}(z), \mathrm{Mortal}(z)]]\end{array}}\;\textsf{Dcut}\!\downarrow}{\begin{array}{l}[x\cdot \mathrm{Socrates}(x), \mathrm{Human}(x),\\ \;[\cdot[\cdot \mathrm{Mortal}(x)]],\\ \;[z\cdot \mathrm{Socrates}(z), \mathrm{Mortal}(z)]]\end{array}}\;\textsf{Ins}}{\begin{array}{l}[x\cdot \mathrm{Socrates}(x), \mathrm{Human}(x),\\ \;[\cdot \mathrm{Human}(x), [\cdot \mathrm{Mortal}(x)]],\\ \;[z\cdot \mathrm{Socrates}(z), \mathrm{Mortal}(z)]]\end{array}}\;\textsf{Iter}}{\begin{array}{l}[x\cdot \mathrm{Socrates}(x), \mathrm{Human}(x),\\ \;[y\cdot \mathrm{Human}(y), [\cdot \mathrm{Mortal}(y)]],\\ \;[z\cdot \mathrm{Socrates}(z), \mathrm{Mortal}(z)]]\end{array}}\;\textsf{Unif}\!\uparrow$$

Figure 9.16.: A proof in the inductive syntax of Beta

Inference rules As already suggested earlier, the garden syntax for graphs quotients the LoI syntax: the first Deiteration step in Figure 9.12 cannot be performed in the garden syntax, because its premiss and conclusion are represented by the same graph. In fact, both Iteration and

Deiteration are automatically handled by the notion of *scope* for binders, that results from the endoporeutic reading of graphs. Then it only remains to account for Insertion and Deletion on LoIs, which is done by capturing our intuition relating these principles to *unification* in the rules Unif↓ and Unif↑ of Figure 9.15, respectively. Thus the inductive version of Beta is obtained by simply adding these two rules to those of Alpha introduced in Figure 9.4.

Figure 9.16 gives a derivation of Aristotle's syllogism in this system. Even though we do not apply the Deit rule anymore compared to the graphical proof of Figure 9.12, we need two additional instances of Unif↑ on z and x, that would correspond to two instances of Del on the associated LoI.

Remark 9.8.1 As in Alpha, graphs G, H, K and their one-holed contexts $G\square, H\square, K\square$ are multisets of nodes. But now they do not correspond anymore to *areas* in the graphical notation, which are instead captured by gardens γ, δ, χ. In particular, this entails a subtle difference from Peirce's formulation of Beta: because rules apply to graphs and not gardens, one cannot have binders at the top-level of SA, they must be enclosed in at least one cut. Thus to be able to reason on (the garden version of) the graph (9.1), we must first enclose it in a double-cut, giving the graph

$$[\cdot [x \cdot \mathsf{TheSittingPerson}(x), \mathsf{Socrates}(x)]]$$

While this choice might seem confusing, it makes the formulation of rules more uniform with that of Alpha, and will ease the transition to the flower calculus in Chapter 10.

Note that the rules Unif↓ and Unif↑ rely on the usual notion of capture-avoiding substitution:

Definition 9.8.2 (Substitution) *The* capture-avoiding substitution *of a variable y for a variable x in a graph G, written $G\{y/x\}$, is defined by mutual recursion as follows:*

$$p(x_1, \dots, x_n)\{y/x\} = p(z_1, \dots, z_n) \text{ with } z_i = \begin{cases} y & \text{if } x_i = x \\ x_i & \text{otherwise} \end{cases}$$

$$g_1, \dots, g_n\{y/x\} = g_1\{y/x\}, \dots, g_n\{y/x\}$$

$$(\mathbf{z} \cdot G)\{y/x\} = \begin{cases} \mathbf{z} \cdot G\{y/x\} & \text{if } x \notin \mathbf{z} \\ \mathbf{z} \cdot G & \text{otherwise} \end{cases}$$

$$[\gamma]\{y/x\} = [\gamma\{y/x\}]$$

Also, this system supports free variables, and there is no need to forbid them like Peirce did in his original LoI-based syntax.

Flower Calculus | 10.

10.1 Intuitionistic existential graphs 188
10.1.1 The scroll 188
10.1.2 Seeds of intuitionism . . 189
10.1.3 Parallel conclusions . . . 191
10.1.4 Quantifiers 192
10.1.5 Coherent formulas . . . 193
10.2 Flowers 194
10.2.1 Blooming 194
10.2.2 Multisets 194
10.2.3 Substitutions 196
10.3 Calculus 198
10.3.1 Preliminary definitions 198
10.3.2 Rules 199
10.3.3 Proofs 201
10.4 Kripke semantics . . 209
10.5 Soundness 211
10.6 Completeness 217
10.6.1 Theories 218
10.6.2 Completion 219
10.6.3 Adequacy 219
10.7 Automated proof search 223
10.7.1 Preliminary definitions 223
10.7.2 The algorithm 225
10.7.3 Empirical assessments . 228
10.8 The Flower Prover . 231
10.8.1 Interaction principles . . 232
10.8.2 Towards a unified workflow 236
10.9 Conclusion 240
10.9.1 Related works 240
10.9.2 Future works 243
10.9.3 Theory vs. Practice . . . 245

> In a certain flower garden, each flower was either red, yellow, or blue, and all three colors were represented. A statistician once visited the garden and made the observation that whatever three flowers you picked, at least one of them was bound to be red. A second statistician visited the garden and made the observation that whatever three flowers you picked, at least one was bound to be yellow. Two logic students heard about this and got into an argument. The first student said: "It therefore follows that whatever three flowers you pick, at least one is bound to be blue, doesn't it?" The second student said: "Of course not!". Which student was right, and why?
>
> **Raymond Smullyan**, *The Flower Garden*, 1985

We introduce the *flower calculus*, a novel proof system for intuitionistic predicate logic based on syntactic objects called *flowers*. We start by explaining how flowers stem from considerations in graphical logic, and more specifically from an intuitionistic variant of the existential graphs of C. S. Peirce proposed by A. Oostra. Then we present our inductive syntax for flowers, reminiscent at the same time of the nested sequents of deep inference proof theory, and the geometric/coherent formulas of categorical logic.

A salient feature of our calculus inherited from EGs, is that it is *fully iconic*: it dispenses completely with the traditional notion of symbolic formula, operating instead as a rewriting system on flowers containing only atomic predicates. We also propose a notion of proof geared towards analyticity results à la Gentzen, suggesting new rules absent from other works on intuitionistic EGs. This allows us to prove admissibility theorems for many rules, including Peirce's deletion rule which is a variant of Gentzen's cut rule. These results are obtained as a consequence of our soundness and completeness proofs with respect to Kripke semantics, in the spirit of the *normalization-by-evaluation* technique.

Furthermore, the kernel of rules targetted by completeness is fully invertible, a desirable property in both automated and interactive proof search. This is illustrated by our implementation of the `Flower Prover`, an early prototype of GUI for ITPs that uses the rules of the flower calculus both for direct manipulation of flowers in its frontend, and automated simplification of goals in its backend.

The chapter is organized as follows: in Section 10.1, we retrace the origin of Oostra's syntax for *intuitionistic existential graphs* (IEGs) as a natural generalization of the *scroll*, an icon for implication introduced by Peirce that inspired the very creation of EGs. In Section 10.2, we explain how flowers are really just a fun and metaphorical way to draw IEGs, and proceed to give them an inductive, multiset-based syntax as in Section 9.3. In Section 10.3, we introduce the full set of inference rules of the flower calculus as well as our notion of proof, and prove a few syntactic properties, including two deduction theorems. In Section 10.4, we give a direct Kripke semantics to flowers, avoiding the need for translations to and from formulas. In Section 10.5, we show that the rules of the flower calculus are valid with respect to our Kripke semantics, and in Section 10.6 we identify a complete fragment of the system where all rules are both *analytic* and *invertible*. This entails the admissibility of all rules outside of this fragment, and as a consequence the analyticity of the system. We exploit these properties in Section 10.7 by describing an algorithm for fully automated proof search in the propositional fragment; unfortunately, the current version of the algorithm is neither terminating nor complete. Then in Section 10.8 we give an overview of the `Flower Prover`, a prototype of GUI in the Proof-by-Action paradigm whose actions map directly to the rules of the flower calculus, and which integrates nicely with (a restricted version of) our search procedure. We conclude in Section 10.9 by a comparison with some related works, and a discussion of future works and applications that we envision.

10.1. Intuitionistic existential graphs

10.1.1. The scroll

In Section 9.1, we presented the syntax of existential graphs (EGs) as stemming from two fundamental icons: the sheet of assertion (SA), with its ability to represent the conjunction of assertions through *juxtaposition*, and *cuts* in SA that signify the denial or negation of assertions. However as noted in Remark 9.1.1, the first interpretation of juxtaposition proposed by Peirce was that of *disjunction*, in his system of entitative graphs. According to him, the illative transformations of EGs are a necessary consequence of the *conjunctive* interpretation of juxtaposition, as witnessed by the following excerpt [202, p. 533]:

[202]: Peirce (1906), 'Prolegomena to an Apology for Pragmaticism'

> If you carefully examine the above conventions, you will find that they are simply the development, and excepting in their insignificant details, the inevitable result of the development of the one convention that if any Graph, A, asserts one state of things to be real and if another graph, B, asserts the same of another state of things, then AB, which results from setting both A and B upon the sheet, shall assert that both states of things are real.

He goes on to notice:

> This was not the case with my first system of Graphs, described in Vol. VII of The Monist, which I now call Entitative Graphs. But I was forced to this principle by a series of considerations which ultimately arrayed themselves into an exact logical deduction of all the features of Existential Graphs.

Thus the conjunctive reading of juxtaposition itself stemmed from "a series of considerations" that "forced" Peirce to adopt it. While in this article he does not give the full "exact logical deduction of all the features of Existential Graphs", he exposes in some details the initial and determining insight that kickstarted the whole development: the discovery of the icon called the *scroll*. Again, I will let Peirce speak for himself [202, pp. 533–534]:

> Accordingly, since logic has primarily in view argument, and since the conclusiveness of an argument can never be weakened by adding to the premisses or by subtracting from the conclusion, I thought I ought to take the general form of argument as the basal form of composition of signs in my diagrammatization; and this necessarily took the form of a "scroll", that is [...] a curved line without contrary flexure and returning into itself after once crossing itself, and thus forming an outer and an inner "close".

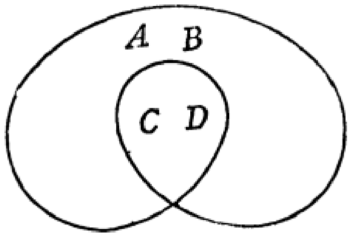


Figure 10.1.: Peirce's scroll

Figure 10.1 shows Peirce's drawing of the scroll as it appears in [202, Fig. 5]. He defines its intended meaning like so [202, p. 534–535]:

> I shall call the outer boundary the Wall; and the inner, the Fence. In the outer I scribed the Antecedent, in the inner the Consequent, of a Conditional Proposition de inesse. [...][Thus the meaning of Figure 10.1 is] that if both A and B are true, then both C and D are true. [...] a Conditional de inesse (unlike other conditionals) only asserts that either the antecedent is false or the consequent is true.

[151]: Lewis (1920), 'A Survey of Symbolic Logic'

This shows the classical view of Peirce on EGs, who interprets the scroll as signifying the *conditional de inesse* — also called nowadays *material implication*, and defined here in its disjunctive form, expressed symbolically by $A \supset B \triangleq \neg A \vee B$. This is no coincidence that Peirce based his most fundamental icon on implication: according to Lewis [151, p. 79], he was the one who introduced the "illative relation" of implication into symbolic logic in the first place, by giving it a distinguished symbol, and studying extensively the algebraic laws that govern it (including Peirce's law).

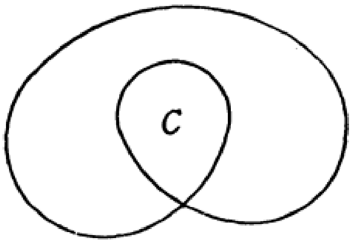


Figure 10.2.: Peirce's scroll with a blank antecedant

10.1.2. Seeds of intuitionism

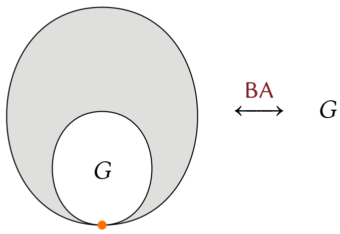


Figure 10.3.: The rule of Blank Antecedant

Blank Antecedant A first principle that Peirce derives from the scroll is the following [202, p. 534]:

> [...] any insertion [is] permitted in the outer close, and any omission from the inner close. By applying the former clause of this rule to [Figure 10.2], we see that this scroll with the outer close void, justifies the assertion that if no matter what be true, C is in any case true; so that the two walls of the scroll, when nothing is between them, fall together, collapse, disappear, and leave only the contents of the inner close standing, asserted, in the open field.

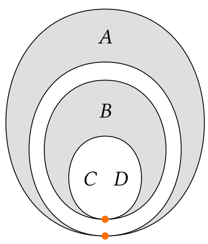


Figure 10.4.: Currying as scroll nesting

This first form of "collapsing of walls" is called the rule of Blank Antecedant in [159], and corresponds symbolically to the equivalence $\top \supset A \simeq A$. The reader might be tempted to see the "former clause" that permits any insertion in the outer close as a special case of the Insertion principle of Alpha (Section 9.1). However, we stress again that Peirce first identified this clause as a feature of the scroll, seen as the diagrammatic embodiment of the "general form of argument" mentioned in a previous excerpt. The principle of Insertion only followed as a subsequent generalization, stemming from the analysis of the scroll into two nested cuts [202, p. 535]:

> [...] and you will further see that a scroll is really nothing but one oval within another.

To emphasize this point, we will from now on depict scrolls as two nested cuts joined at a single point highlighted in orange, as illustrated in Figure 10.3.

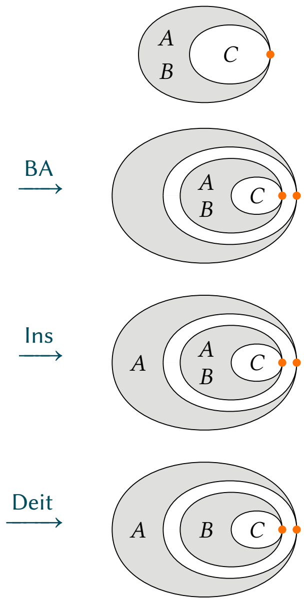


Figure 10.5.: Intuitionistic proof of currying

> **Remark 10.1.1** It is interesting to note that the rule of Blank Antecedant is not seen as primitive by Peirce, but as a consequence of a *dynamic potential* of the scroll: namely, the ability to insert anything in the outer close, at will. This is another manifestation of Peirce's concern for the question of *illative atomicity*, and is to be related to the elimination of the Double−cut rule discussed in Section 9.4.

Currying Peirce was aware of the phenomenon of *currying*, expressed symbolically by the equivalence $A \supset B \supset C \simeq A \wedge B \supset C$, as witnessed by the following passage [202, p. 535]:

> Now, Reader, if you will just take pencil and paper and scribe the scroll expressing that if A be true, then it is true that if B be true C and D are true [Figure 10.4], and compare this with [Figure 10.1], which amounts to the same thing in meaning, you will see that scroll walls with a void between them collapse even when they belong to different scrolls.

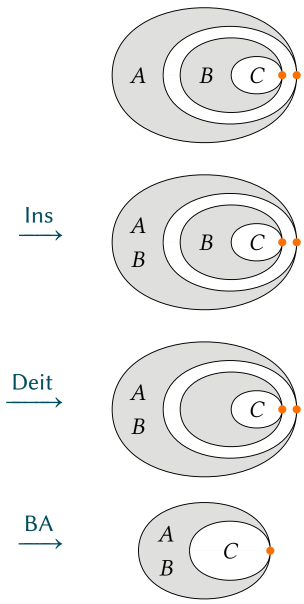


Figure 10.6.: Intuitionistic proof of uncurrying

It is remarkable that he comes to this conclusion by a topological argument, noting that this second form of "collapsing of walls", now involving two different scrolls, follows from the scroll beeing composed of two nested cuts. If we reject this interpretation by requiring that the Fence (the inner oval) stays glued to the Wall (the outer oval), then one cannot derive currying through the rule of Double−cut, precisely because the system only permits to collapse a Wall and a Fence continuously joined in the same scroll, by the weaker rule of Blank Antecedant. Fear not however, as one can still derive the currying and uncurrying laws in this intuitionistic setting, but through the additional use of the insertion and deiteration rules, as depicted in Figure 10.5 and Figure 10.6. Yet we find that Peirce's insight on the topological explanation of currying in the classical setting remains noteworthy.

> **Remark 10.1.2** Note that in Figure 10.5 and Figure 10.6, we give *forward* proofs that rewrite the premiss of the argument into its conclusion,

rather than *backward* proofs that rewrite a goal into the empty SA, as we usually did in previous chapters. Forward proofs correspond to Peirce's usage of the illative transformations — and thus to what can be found in most of the literature on EGs, and have the advantage of being more economical in space by leaving the goal implicit. One can easily go from a forward proof to a backward one as shown by the *deduction theorem* of Sowa [226, Section 6], which also applies in the intuitionistic setting by substituting the rule of Double–cut with the rule of Blank Antecedant.

10.1.3. Parallel conclusions

The *n*-ary scroll In [190], A. Oostra proposes to take the above remark seriously, by reifying the scroll as a primitive icon of EGs ("*rizo*" in Spanish), that exists alongside the cut ("*corte*"), and is distinguished from it. In fact he goes further than this, and proposes to generalize both the cut and the scroll into an n-ary construction called the *curl* ("*bucle*"), where n is the number of inner closes, called *loops* ("*lazos*"). Figure 10.7 shows an example of curl with five loops. In [159], the curl is simply called *n-ary scroll*, and is analyzed into the outer area (that enclosed by the Wall) called the *outloop*, and the inner areas (those enclosed by the n Fences, i.e. the loops of Oostra) called the *inloops*. Then cuts and scrolls are indeed special cases of n-ary scrolls, respectively with $n = 0$ and $n = 1$.

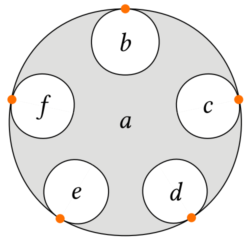


Figure 10.7.: A curl with five loops

Like the unary scroll, the n-ary scroll is to be read as an implication whose antecedant is the content of the outloop, and consequent the content of the inloops. The generalization then consists in taking the *disjunction* of the contents of all inloops: this reflects nicely the etymological meaning of the word "disjunction", since the inloops enclose *disjoint* areas of the outloop to which they are attached. Then the 5-ary scroll of Figure 10.7 is read as the formula $a \supset b \lor c \lor d \lor e \lor f$; and the 0-ary scroll obtained by removing all inloops from the latter as $a \supset \bot$, since a 0-ary disjunction is naturally evaluated to its neutral element $\bot$. This coincides with the intuitionistic reading of negation $\neg A \triangleq A \supset \bot$, and is thus consistent with the interpretation of cuts as negations.

Continuity With this interpretation of the n-ary scroll, the Alpha encodings of disjunction and implication as nested cuts are no longer valid, because they are not intuitionistically equivalent to the associated binary and unary scrolls. This is illustrated in Figure 10.8, where the closeness in meaning is reflected iconically (but not symbolically) in the fact that the graphs only differ in the *continuity* (or lack thereof) between inloops and their outloop. Indeed, contrary to nested cuts, any n-ary scroll can be drawn by a *continuous* movement of the pen, producing a self-intersecting curve as described by Peirce in [202].

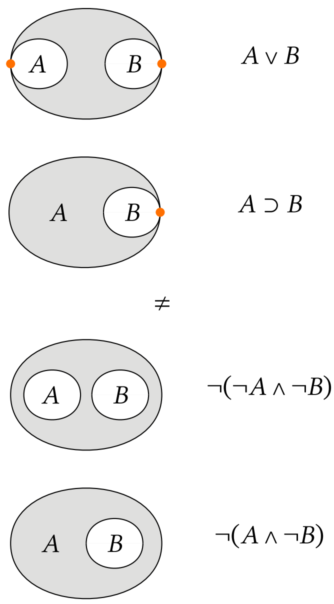


Figure 10.8.: Continuity, disjunction and implication in IEGs

This might be related to other manifestations of the notion of continuity in the semantics of intuitionistic logic, such as the well-known Stone-Tarski interpretation of formulas as topological spaces [232], and the interpretation of proofs as continuous maps in the *denotational semantics* of Dana Scott [2]. Before the advent of Oostra's IEGs, Zalamea gave a detailed analysis of Peirce's philosophy of the *continuum*, how it re-

[2]: Abramsky et al. (1995), *Domain Theory*

lates to modern developments in mathematics, and how it is embodied in existential graphs [259]. Actually according to Oostra [193, p. 162], "the possibility of developing intuitionistic existential graphs was first suggested by Zalamea in the 1990s [257, 258]".

[193]: Oostra (2022), 'Advances in Peircean Mathematics: The Colombian School'

10.1.4. Quantifiers

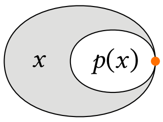
More generally, a *n*-ary scroll with atoms $G := a_1, \ldots, a_m$ in its outloop and $\Delta := \begin{pmatrix} a_{1,1} & \ldots & a_{1,p_1} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \ldots & a_{1,p_n} \end{pmatrix}$ in its inloops, where each row H_j in Δ encodes an inloop, can be interpreted as the formula

$$\bigwedge_{i=1}^{m} a_i \supset \bigvee_{j=1}^{n} \bigwedge_{k=1}^{p_n} a_{j,k}$$

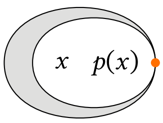
If one adds *binders* to the mix (see Section 9.8) by having $\gamma := \mathbf{x} \cdot G$ as outloop, and $\Xi := \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} \cdot \Delta$ as inloops, then the interpretation is extended into the formula

$$\forall \mathbf{x}. \left(\bigwedge_{i=1}^{m} a_i \supset \bigvee_{j=1}^{n} \exists \mathbf{x}_j. \bigwedge_{k=1}^{p_n} a_{j,k} \right)$$

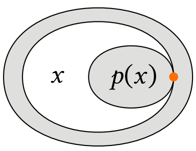
as depicted in Figure 10.9. Typically, the particular case where $\gamma := x \cdot \varnothing$ and $\Xi := (\varnothing) \cdot (p(x))$ encodes the graph



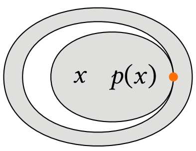
expressing the universal quantification $\forall x.p(x)$, and the case where $\gamma := \varnothing \cdot \varnothing$ and $\Xi := (x) \cdot (p(x))$ the graph



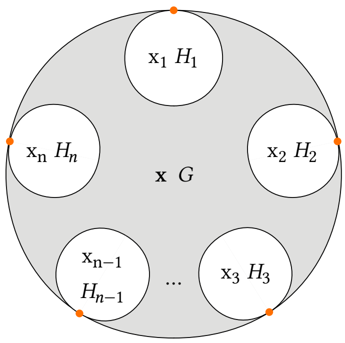
expressing the existential quantification $\exists x.p(x)$. The interpretation is invariant under polarity, meaning that for instance the graphs



and



obtained by enclosing the previous graphs in a cut are interpreted with the same quantifiers, as the formulas $\neg\forall x.p(x)$ and $\neg\exists x.p(x)$. In Beta, we would have exploited the classical equivalences $\neg\forall x.A \simeq \exists x.\neg A$ and



$$\forall \mathbf{x}. \left(\bigwedge G \supset \bigvee_{j=1}^{n} \exists \mathbf{x}_j. \bigwedge H_j \right)$$

Figure 10.9.: Formula interpretation of the *n*-ary scroll

$\neg\exists x.A \simeq \forall x.\neg A$ (justified by the Double−cut principle) in order to interpret them as $\exists x.\neg p(x)$ and $\forall x.\neg p(x)$, emphasizing the idea that positive and negative binders encode respectively $\exists$ and $\forall$. But this is not possible anymore with the intuitionistic interpretation of n-ary scrolls, where the $\exists/\forall$ duality is replaced by the inloop/outloop distinction. In fact, we are tempted to further qualify this distinction of *adjunction*, following a classical result of Lawvere in the context of categorical logic [148].

[148]: Lawvere (1970), 'Quantifiers and Sheaves'

10.1.5. Coherent formulas

Lawvere is also known for some contributions to the study of *geometric logic* [147], a subset of the formulas of FOL first discovered by Skolem [223] that is capable of expressing many mathematical theories, and has close connections to *topos theory*. Quite remarkably, the interpretation of n-ary scrolls coincides exactly with the class of *coherent formulas*, which are the formulas of geometric logic where infinitary disjunctions are restricted to finitary ones. There is a difference however: the full syntax of IEGs allows for arbitrary *nestings* of n-ary scrolls inside eachother, i.e. the multisets G and H_j in Figure 10.9 can contain n-ary scrolls in addition to atoms; while coherent formulas are restricted to atoms.

[147]: Lawvere (1975), 'Continuously Variable Sets; Algebraic Geometry = Geometric Logic'

[223]: Skolem (1920), 'Logisch-Kombinatorische Untersuchungen Über Die Erfüllbarkeit Oder Bewiesbarkeit Mathematischer Sätze Nebst Einem Theorem Über Dichte Mengen'

Coherent formulas have some nice properties, which might also apply to IEGs to some extent. We only mention two important ones:

Completeness Every first-order theory has a coherent conservative extension, making coherent formulas (and thus non-nested n-ary scrolls) in principle as expressive as arbitrary first-order formulas [74].

[74]: Dyckhoff et al. (2015), 'Geometrization of First-Order Logic'

Automation Coherent formulas benefit from faster proof-search procedures compared to arbitrary formulas, making automation more tractable computationally. They also allow the direct encoding of many reasoning problems, thanks to their use of the full set of connectives and quantifiers of FOL; and avoiding complex encodings (as can be found e.g. in SMT solvers) is crucial in *interactive* theorem proving, where the user and the computer manipulate the same formulas in goals [19]. This has been exploited already in some domain-specific theorem provers, like the `Larus` prover that automatically generates illustrated proofs in geometry[1] [131].

[19]: Bezem et al. (2005), 'Automating Coherent Logic'

1: Incidentally, projective geometry was one of the motivating applications that led Skolem to identify the class of coherent formulas [19].

[131]: Janičić et al. (2023), 'Automated generation of illustrated proofs in geometry and beyond'

Remark 10.1.3 Thus with IEGs, one becomes able to reason *geometrically* on geometric formulas that speak about geometry: another beautiful incarnation of the reflexivity at work in Peirce's iconic logic.

10.2. Flowers

10.2.1. Blooming

As we have seen, the (n-ary) scroll is a powerful icon, because it captures the distinction between classical and intuitionistic logic as being a matter of *continuity* between the space of inputs/hypotheses (outloop) and the spaces of outputs/conclusions (inloops), reflecting an intuition discovered much later in the denotational semantics of the λ-calculus. However as a diagrammatic component to be operated upon through direct manipulation, it has one notable flaw, also shared with the classical cut-based syntax: it quickly induces heavy nestings of curves in the plane, making even a simple graph like that of Figure 10.4 hard to read for an untrained eye.

Before devising an alternative syntax, one should ask: what are the essential features of the scroll that we want to preserve? Following the previous observations, we identified two of them:

Continuity the scroll is a self-intersecting continuous curve, which can be drawn in one stroke of the pen;

Polarity this curve delineates two kinds of areas: inloops that have the same polarity as the area on which the scroll is scribed, and the outloop which has the opposite polarity.

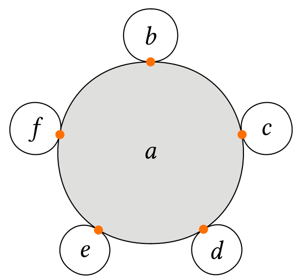


Figure 10.10.: Turning a 5-ary scroll inside-out

Fortunately, these two properties are preserved when turning inloops *inside-out*, as illustrated in Figure 10.10. This might be because the very process of turning inside-out can be seen as a continuous movement in three-dimensional space, where the inloops are rotated around their intersection points with the outloop. In this way, we have effectively divided the amount of curve-nesting in scrolls by two. And as an added bonus, the new icon is reminiscent of a *flower*, as if it had bloomed from its curled bud; or as if the pistol cylinder from Figure 10.7 had transformed into a *pistil*, and its bullet chambers into *petals*[2].

2: As the saying goes: make love, not war.

From that point onwards, we decided to fully embrace the flower metaphor: first in our drawing style as witnessed in Figure 10.11, but also in our syntactic terminology, to be introduced in the next pages. Negative outloops are now drawn as *yellow* pistils for a more colorful experience, and inloops as transparent petals, i.e. of the same color as the area on which they are scribed. We also drop the requirement that petals should intersect their pistil at a single point, for purely aesthetic reasons.

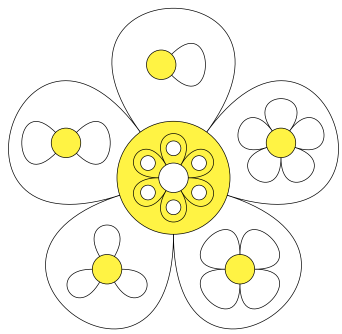


Figure 10.11.: Nested flowers

10.2.2. Multisets

As we did for classical EGs in Section 9.3 and Section 9.8, we are now going to distill the syntactic essence of flowers into an inductive, (multi)set-based data structure. This will allow for a more compact textual notation, that is better suited to proof-theoretical study.

In Section 9.7, we explained how the graphs of Beta allow to represent purely relational statements, without function symbols. Since functions are just deterministic relations, one can in principle formalize any first-order theory in this syntax[3]. However it is much more convenient to have a dedicated syntax for functions, and we will thus introduce them as is usually done in predicate calculus.

3: Conversely, every relation can be faithfully encoded as its characteristic function, which is the basis for the formalization of mathematics in *type theories*.

Definition 10.2.1 (First-order signature) *A first-order signature is a triplet $\Sigma = (\mathcal{F}, \mathcal{P}, \mathrm{ar})$, where $\mathcal{F}$ and $\mathcal{P}$ are respectively the countable sets of* function *and* predicate *symbols of Σ, and $\mathrm{ar} : \mathcal{F} \cup \mathcal{P} \to \mathbb{N}$ gives an* arity *to each symbol.*

In the following, we assume given a denumerable set of variables $\mathcal{V}$ and a first-order signature Σ.

Definition 10.2.2 (Terms) *The set of terms $\mathbb{T}$ is defined inductively as follows:*

(Variable) *If $x \in \mathcal{V}$ then $x \in \mathbb{T}$;*

(Application) *If $f \in \mathcal{F}$ and $\vec{\mathbf{t}} \in \mathbb{T}^{\mathrm{ar}(f)}$, then $f(\vec{\mathbf{t}}) \in \mathbb{T}$.*

Definition 10.2.3 (Flowers) *The sets of* flowers *$\mathbb{F}$ and* gardens *$\mathbb{G}$ are defined mutually inductively as follows:*

(Atom) *If $p \in \mathcal{P}$ and $\vec{\mathbf{t}} \in \mathbb{T}^{\mathrm{ar}(p)}$, then $p(\vec{\mathbf{t}}) \in \mathbb{F}$;*

(Garden) *If $\mathbf{x} \subset \mathcal{V}$ is a finite set and $\Phi \subset \mathbb{F}$ a finite multiset, then $\mathbf{x} \cdot \Phi \in \mathbb{G}$;*

(Flower) *If $\gamma \in \mathbb{G}$ and $\Delta \subset \mathbb{G}$ is a finite multiset, then $\gamma \supset \Delta \in \mathbb{F}$.*

Digression

According to the Merriam-Webster dictionary [171], the word "corollary" has *botanical* etymological roots:

> […] the seed of corollary was planted initially by the Latin noun corōlla meaning "small wreath of flowers", which later bloomed into another Latin noun, corōllārium, referring to a garland given as a reward as well as to a gratuity or an unsolicited payment. […] The formality of corollary is thanks to its formal roots […]

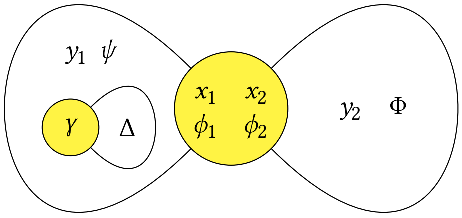
Then our flower metaphor is a meaningful tribute to these origins: the petals (corolla) of a flower can literally be seen as the *corollaries* of its pistil, when the latter happens to be true.

Any finite set $\mathbf{x} \subset \mathcal{V}$ of variables is called a *sprinkler*, finite multiset $\Phi \subset \mathbb{F}$ of flowers a *bouquet*, and finite multiset $\Gamma \subset \mathbb{G}$ of gardens a *corolla*. We will often write gardens as $x_1, \ldots, x_n \cdot \phi_1, \ldots, \phi_m$, where the x_i are called *binders*; and non-atomic flowers as $\gamma \supset \delta_1 ; \ldots ; \delta_n$, where γ is the *pistil*, and the δ_i are called *petals*. We write $\{E_i\}_i^n$ to denote a finite (multi)set of size n with elements E_i indexed by $1 \leq i \leq n$. We also omit writing the empty (multi)set, accounting for it with blank space as is done in sequent notation or in EGs; in particular, $\cdot$ stands for the empty garden $\varnothing \cdot \varnothing$, $\gamma \supset$ for the flower with no petals $\gamma \supset \varnothing$, and $\gamma \supset \ \cdot$ for the flower with one empty petal.

Note that the order of precedence of operators is $, < \cdot < ; < \supset$ so that for instance, the string

$$x_1, x_2 \cdot \phi_1, \phi_2 \supset y_1 \cdot \psi, (\gamma \supset \Delta); y_2 \cdot \Phi$$

is parsed as the flower



Also to improve readability, we will most of the time omit the garden dot '$\cdot$' when the sprinkler is empty, writing Φ instead of $\cdot\,\Phi$.

Kind	Letters
Variables ($\mathscr{V}$)	x, y, z
Terms ($\mathbb{T}$)	t, u, v
Flowers ($\mathbb{F}$)	ϕ, ψ, ξ
Gardens ($\mathbb{G}$)	γ, δ
Sprinklers	$\mathbf{x}, \mathbf{y}, \mathbf{z}$
Term vectors	$\vec{\mathbf{t}}, \vec{\mathbf{u}}, \vec{\mathbf{v}}$
Substitutions	σ, τ
Bouquets	Φ, Ψ, Ξ
Corollas	Γ, Δ
Contexts	$\Phi\square, \Psi\square, \Xi\square$
Theories	$\mathcal{T}, \mathcal{U}$

Table 10.1.: Notational conventions for meta-variables

Remark 10.2.1 In some places the choice of letter for meta-variables will be important to disambiguate the kind of syntactic object we denote. Table 10.1 summarizes our chosen notational conventions in this respect.

As usual, we introduce a *depth* measure that will allow us to reason inductively on the structure of flowers:

Definition 10.2.4 (Depth) *The depth $|-|$ of a flower or garden is defined mutually recursively as follows:*

$$
\begin{aligned}
|p(\vec{\mathbf{t}})| &= 0 \\
|\mathbf{x} \cdot \Phi| &= \max_{\phi \in \Phi} |\phi| \\
|\gamma \supset \Delta| &= 1 + \max(|\gamma|, \max_{\delta \in \Delta} |\delta|)
\end{aligned}
$$

10.2.3. Substitutions

We now proceed with routine definitions for handling variables and substitutions of terms in flowers.

Definition 10.2.5 (Free variables) *The sets of free variables $\mathsf{fv}(-)$ of a term, flower, bouquet or garden are defined mutually recursively as follows:*

$$
\begin{aligned}
\mathsf{fv}(x) &= \{x\} & \mathsf{fv}(\Phi) &= \bigcup_{\phi \in \Phi} \mathsf{fv}(\phi) \\
\mathsf{fv}(f(\vec{\mathbf{t}})) &= \bigcup_{t \in \vec{\mathbf{t}}} \mathsf{fv}(t) & \mathsf{fv}(\mathbf{x} \cdot \Phi) &= \mathsf{fv}(\Phi) \setminus \mathbf{x} \\
\mathsf{fv}(p(\vec{\mathbf{t}})) &= \bigcup_{t \in \vec{\mathbf{t}}} \mathsf{fv}(t) & \mathsf{fv}(\mathbf{x} \cdot \Phi \supset \Delta) &= \mathsf{fv}(\mathbf{x} \cdot \Phi) \cup \bigcup_{\mathbf{y} \cdot \Psi \in \Delta} \mathsf{fv}(\mathbf{x}, \mathbf{y} \cdot \Psi)
\end{aligned}
$$

We say that a term, flower, bouquet or garden is closed *when its set of free variables is empty.*

Remark 10.2.2 Note that the scope of a binder located in a pistil extends both to the pistil *and* to all its attached petals, whereas for a binder located in a petal it is limited to said petal. This is reflected in the above definition of free variables for (non-atomic) flowers, and is visually explained by the nesting of curves in an n-ary scroll.

Definition 10.2.6 (Bound variables) *The sets of bound variables* $\mathsf{bv}(-)$ *of a flower, bouquet or garden are defined mutually recursively as follows:*

$$\mathsf{bv}(p(\vec{\mathbf{t}})) = \varnothing \qquad \mathsf{bv}(\mathbf{x} \cdot \Phi) = \mathbf{x} \cup \mathsf{bv}(\Phi)$$

$$\mathsf{bv}(\Phi) = \bigcup_{\phi \in \Phi} \mathsf{bv}(\phi) \qquad \mathsf{bv}(\gamma \supset \Delta) = \mathsf{bv}(\gamma) \cup \bigcup_{\delta \in \Delta} \mathsf{bv}(\delta)$$

To avoid reasoning about α-equivalence, we adopt in this work the so-called *Barendregt convention* that all variable binders are distinct, both among themselves and from eventual free variables. Formally, we assume that for any bouquet Φ, the two following conditions hold:

1. computing $\mathsf{bv}(\Phi)$ as a multiset gives the same result as computing it as a set;
2. $\mathsf{bv}(\Phi) \cap \mathsf{fv}(\Phi) = \varnothing$.

To define substitutions, we introduce a general notion of *function update*, which will be useful for the semantic evaluation of flowers in Section 10.4.

Definition 10.2.7 (Function update) *Let A, B be two sets, $f, g : A \to B$ two functions and $R \subseteq A$ some subset of their domain. The* update *of f on R with g is the function defined by:*

$$(f|_R\, g)(x) = \begin{cases} g(x) & \text{if } x \in R \\ f(x) & \text{otherwise} \end{cases}$$

$-|_{\_}-$ is left-associative, that is $f|_R\, g|_S\, h = (f|_R\, g)|_S\, h$. Also if f or g is the identity function 1 we omit writing it, i.e. $f|_R\; = f|_R\, 1$ and $|_R\, g = 1\,|_R\, g$.

Definition 10.2.8 (Substitution) *A* substitution *is a function $\sigma : \mathscr{V} \to \mathbb{T}$ with a finite* support $\mathsf{supp}(\sigma) = \{x \mid \sigma(x) \neq x\}$*. By abuse of notation, we will write $\sigma : \mathbf{x} \to \mathbb{T}$ to denote a substitution σ whose support is $\mathbf{x}$. The domain of substitutions is extended to terms, flowers, bouquets and gardens mutually recursively as follows:*

$$\sigma(f(t_1, \dots, t_n)) = f(\sigma(t_1), \dots, \sigma(t_n))$$
$$\sigma(p(t_1, \dots, t_n)) = p(\sigma(t_1), \dots, \sigma(t_n))$$
$$\sigma(\phi_1, \dots, \phi_n) = \sigma(\phi_1), \dots, \sigma(\phi_n)$$

$$\sigma(\mathrm{x} \cdot \Phi) = \mathrm{x} \cdot \sigma|_{\mathrm{x}}(\Phi)$$
$$\sigma(\mathrm{x} \cdot \Phi \mathrel{\mathsf{D}} \delta_1 ; \ldots ; \delta_n) = \sigma(\mathrm{x} \cdot \Phi) \mathrel{\mathsf{D}} \sigma|_{\mathrm{x}}(\delta_1) ; \ldots ; \sigma|_{\mathrm{x}}(\delta_n)$$

Definition 10.2.9 (Capture-avoiding substitution) *We say that a substitution $\sigma : \mathbf{x} \to \mathbb{T}$ is* capture-avoiding *in a bouquet Φ if $\mathsf{fv}(\sigma(x)) \cap \mathsf{bv}(\Phi) = \varnothing$ for every $x \in \mathbf{x}$.*

10.3. Calculus

10.3.1. Preliminary definitions

Contexts Equipped with an inductive syntax, we can now express formally the inference rules of our flower calculus, just as we did for Alpha (Section 9.3) and Beta (Section 9.8). There, graphs and their contexts were defined as multisets of nodes, which have now turned into bouquets of flowers:

Definition 10.3.1 (Context) *A* context $\Phi\square$ *is a bouquet which contains exactly one occurrence of a special flower written $\square$, called its* hole*. The hole can always be* filled *(substituted) with any other bouquet Ψ or context $\Xi\square$, producing a new bouquet $\Phi\boxed{\Psi}$ or context $\Phi\boxed{\Xi\square}$. In particular, filling with the empty bouquet will yield a bouquet $\Phi\boxed{}$, which is just $\Phi\square$ with its hole removed. A* flower context $\phi\square$ *is a context with exactly one flower.*

Definition 10.3.2 (Depth) *The* depth $|\Phi\square|$ *of a context $\Phi\square$ is defined recursively as follows:*

$$|\Psi, \square| = 0$$
$$|\Psi, (\mathbf{x} \cdot \Phi\square \mathrel{\mathsf{D}} \Delta)| = 1 + |\Phi\square|$$
$$|\Psi, (\gamma \mathrel{\mathsf{D}} \mathbf{x} \cdot \Phi\square ; \Delta)| = 1 + |\Phi\square|$$

Contrarily to EGs (Definition 9.3.4), the number of inversions of a context does not coincide with its depth, since petals increase depth but preserve polarity:

Definition 10.3.3 (Inversions) *The number of* inversions $\mathsf{inv}(\Phi\square)$ *of a context $\Phi\square$ is defined recursively by:*

$$\mathsf{inv}(\Psi, \square) = 0$$
$$\mathsf{inv}(\Psi, (\mathbf{x} \cdot \Phi\square \mathrel{\mathsf{D}} \Delta)) = \mathsf{inv}(\Phi\square) + 1$$
$$\mathsf{inv}(\Psi, (\gamma \mathrel{\mathsf{D}} \mathbf{x} \cdot \Phi\square ; \Delta)) = \mathsf{inv}(\Phi\square)$$

Definition 10.3.4 (Polarity) *We say that a context $\Phi\square$ is* positive *if*

$\mathsf{inv}(\Phi\square)$ *is even, and* negative *otherwise. We denote* positive *and* negative *contexts respectively by* $\Phi^{+}\square$ *and* $\Phi^{-}\square$.

Pollination In order to formulate the equivalent of the (de)iteration rules of EGs for flowers, we introduce a *pollination* relation that captures the availability of a flower in a given context, akin to the *justification* relation of Subsection 9.4.3:

Definition 10.3.5 (Pollination) *We say that a flower* ϕ *can be* pollinated *in a context* $\Phi\square$, *written* $\phi \succ \Phi\square$, *when there exists a bouquet* Ψ *with* $\phi \in \Psi$ *and contexts* $\Xi\square$ *and* Ξ_0 *such that either:*

(*Cross-pollination*) $\Phi\square = \Xi\boxed{\Psi, \Xi_0}$;

(*Self-pollination*) $\Phi\square = \Xi\boxed{\mathbf{x}\cdot\Psi \mathbin{\mathsf{D}} \mathbf{y}\cdot\Xi_0 \,;\, \Delta}$ *for some* $\mathbf{x}, \mathbf{y}, \Delta$.

A bouquet Ψ *can be pollinated in* $\Phi\square$, *written* $\Phi \succ \Phi\square$, *if* $\phi \succ \Phi\square$ *for all* $\phi \in \Phi$.

We now employ the metaphor of *pollination* to speak about (de)iteration in flowers. This is illustrated in Figure 10.12, where the blue dot marks the location of the justifying/pollinating occurrence of ϕ, and the red dots all the areas that it (locally) justifies/pollinates, and thus where ϕ can be (de)iterated[4]. We distinguish two cases of *cross-pollination* and *self-pollination*, as botanists do when describing the reproduction of flowers. This distinction does not exist in classical EGs, because pistils and petals are both identified as instances of cuts[5]. If we were to replace binders by LoIs (lines of identity, see Section 9.7), then the pollination relation would also prescribe in which areas LoIs can be extended/iterated, providing an explanation for the scope of binders (Remark 10.2.2).

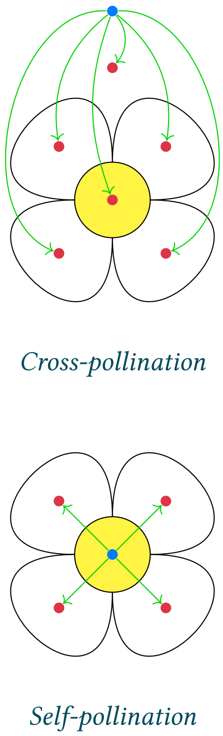


Figure 10.12.: Pollination in flowers

4: Figure 10.12 summarizes visually the flow of information in flowers, just like Figure 9.5 summarized the flow of information in EGs, and Figure 8.10 in bubbles. From a UI point of view, all these figures can be understood as kind of "cheat-sheets", that indicate with arrows the allowed drag-and-drop moves for importing a statement (the source of the DnD) into a new context (the destination of the DnD).

5: The same phenomenon is at work in *subformula linking* (Chapter 3): self-pollination and cross-pollination correspond respectively to the *backward* $\obslash$ and *forward* $\circledast$ interaction operators, which are collapsed into a single interaction operator $*$ in the original formulation of subformula linking for classical linear logic [37].

10.3.2. Rules

As has become standard in this thesis, we define the flower calculus as a *rewriting system* on bouquets, presented in Figure 10.14 as a set of unary deep inference rules: when read *top-down*, they correspond to usual inferences from premiss to conclusion, and will be justified by the soundness theorem of Section 10.5. But the more interesting direction, and the one around which the calculus has been designed, is when you read the rules *bottom-up*: then they are indeed rewriting rules, telling you the different ways in which you can choose to simplify a goal. This is how the *graphical* version of the rules is presented in Figure 10.16 and Figure 10.17.

Let us now describe the rules in more detail, starting with the fragment that is a direct adaptation of the rules of Beta (Figure 9.15):

Blank Antecedent (epis) It allows to enclose any bouquet in a petal attached to an (e)mpty (pis)til. This is one direction of the rule of Blank Antecedent (Figure 10.3), which is a weaker, intuitionistic version

$$\frac{\Phi}{\cdot \mathbin{\mathsf{D}} \cdot \Phi}\ \mathsf{epis}{\downarrow}$$

Figure 10.13.: Converse of epis rule

of the classical rule Dcut↑ of Alpha and Beta. The other direction (rule epis↓ in Figure 10.13) is actually admissible, which might be related to the co-admissibility of Dcut↓ in Alpha (Corollary 9.6.4).

(De)iteration (poll↓, poll↑) Renamed *(poll)ination* rules, they correspond to the rules Iter and Deit of Alpha and Beta, but reformulated with the pollination relation (Definition 10.3.5). In fact in their textual presentation of Figure 10.14, they are more general than (de)iteration rules, because Definition 10.3.5 allows the pollinating bouquet Φ to be *scattered* in the context $\Xi\square$, i.e. its flowers need not be located in the same area. On the contrary in the graphical presentation of Figure 10.16, they are less general since only one flower can be pollinated at a time, rather than an entire bouquet of flowers residing in the same area. But it is easy to see that all these variants are equivalent in deductive power, since the pollination of a bouquet (however scattered) can always be simulated by the successive pollinations of each of its flowers.

Insertion/Deletion (grow, crop, pull, glue) They correspond to the rules Ins and Del of Alpha and Beta, but have doubled in number to account for the syntactic distinction between pistils and petals. More precisely, rules grow and crop allow to insert and delete entire flowers, while rules pull and glue deal with petals. As for pollination rules, manipulating single flowers/petals (graphical version) or entire bouquets/corollas (textual version) does not change the deductive power of the rules.

Unification (ipis, ipet, apis, apet) Rules ipis and ipet allow to (i)nstantiate a sprinkler located respectively in a (pis)til ($\forall$) and a (pet)al ($\exists$) with an arbitrary substitution, while rules apis and apet do the opposite operation of (a)bstracting a set of terms by introducing a sprinkler. They correspond respectively to a generalization of the rules Unif↑ and Unif↓ of Beta, where the variable substitution $\{z/y\}$ becomes an arbitrary substitution σ. Once again, we have twice the amount of rules to account for the pistil/petal distinction, which is not surprising since in the LoI syntax of EGs, they are special cases of Insertion/Deletion. Note that for the instantiation rules ipis/ipet to be invertible, we duplicate the whole flower/petal where the sprinkler occurs, mirroring what is done in multi-conclusion sequent calculi (see Figure 5.4).

The last two rules mainly handle the behavior of disjunctive and absurd statements, i.e. flowers with respectively $n \geq 2$ and $n = 0$ petals, and are closer to sequent-style introduction/elimination rules:

Disjunction Introduction (epet) It allows to erase any flower with an (e)mpty (pet)al. According to Oostra [193, p. 109], Peirce already identified epet as a component of his decision procedure for Alpha (it is simply called "Operation 1" in [193]). This is no coincidence, since we precisely came up with this rule when trying to design a decision procedure for flowers (see Section 10.7).

Disjunction/Absurdity Elimination (srep) It corresponds to a n-ary generalization of the left introduction rule for disjunction in sequent calculus, the 0-ary case capturing absurdity elimination (*ex falso quodlibet*). The binary case is also used in the IEGs system of [159], together with its converse. The name srep is short for (s)elf-(rep)roduction, which

Connective		$\top$	$\wedge$	$\bot$	$\neg$	$\supset$	$\vee$	$\forall$	$\exists$
Corolla		$\geq$	$=$	$<$	$<$	$=$	$>$	$=$	$=$
Bouquets	**Pistil**	–	$<$	$<$	$=$	$\leq$	$<$	$<$	$<$
	Petals	$<$	$>$	–	–	$=$	$=$	$=$	$=$
Sprinklers	**Pistil**	–	$<$	$<$	$<$	$<$	$<$	$\geq$	$<$
	Petals	$<$	$<$	–	–	$<$	$<$	$<$	$\geq$

Table 10.2.: Fragments of intuitionistic logic as cardinality constraints on flowers

is more clearly visualized in the graphical version of the rule in Figure 10.16. Through the Curry-Howard correspondence, it can be related to the *pattern-matching generator* found in modern editors of some functional programming languages, such as the Hazel structure editor and the `Agda` proof assistant [256].

[256]: Yuan et al. (2023), 'Live Pattern Matching with Typed Holes'

The rules of the flower calculus have an interesting property: they are mostly *arity-agnostic*, i.e. they work uniformly on flowers, bouquets and gardens with any number of petals, flowers and binders. In particular, this means that the same rules can be used to capture provability in almost any *fragment* of intuitionistic predicate logic, understood as any subset $\mathfrak{F} \subset \mathbb{F}$ of the set of all flowers[6]. Table 10.2 shows how all the usual symbolic connectives can be expressed by *cardinality* constraints on set-based syntactic constructs: $<, \leq, =, \geq, >$ correspond respectively to a cardinality smaller, smaller or equal, equal, greater or equal, and greater than 1; and '–' denotes the absence of constraint. These constraints are then taken *conjunctively* for a single connective (column-wise); and they can be freely mixed *disjunctively* (row-wise), in order to capture any fragment corresponding to a subset of connectives.

6: The only exception seems to be the rule (ipis) (resp. ipet), whose duplication of flowers (resp. petals) prevents $\forall$ (resp. $\exists$) from being provable without $\wedge$ (resp. $\vee$). This can be fixed by simply removing the duplication, and polarizing the context of application as for the rule apis (resp. apet), at the cost of making the rule non-invertible.

10.3.3. Proofs

Our notions of derivation and proof are essentially the same as the ones given for EGs in Section 9.3, except that we distinguish from the outset between two kinds of derivations, stemming from our partitioning of the rules into two sets: the *natural* rules denoted by ❀, and the *cultural* rules denoted by ✂. In particular, every ❀-rule is both *analytic* (i.e. every atom in the premiss already appears in the conclusion) and *invertible* (this will be shown in Section 10.5); on the contrary, all ✂-rules are *non-invertible*, and they will be shown to be *admissible* in Section 10.6.

Definition 10.3.6 (Derivation) *Given a set of rules* R, *we write* $\Phi \rightarrow_{\mathsf{R}} \Psi$ *to indicate a rewrite* step *in* R, *that is an instance of some* $r \in \mathsf{R}$ *from Figure 10.14 with* Ψ *as premiss and* Φ *as conclusion. We just write* $\Phi \rightarrow \Psi$ *to mean* $\Phi \rightarrow_{\text{❀} \cup \text{✂}} \Psi$. *A* derivation $\Phi \rightarrow^{n}_{\mathsf{R}} \Psi$ *is a sequence of rewrite steps* $\Phi_0 \rightarrow_{\mathsf{R}} \Phi_1 \dots \rightarrow_{\mathsf{R}} \Phi_n$ *with* $\Phi_0 = \Phi$, $\Phi_n = \Psi$ *and* $n \geq 0$. *Generally the length* n *of the derivation does not matter, and we just write* $\Phi \rightarrow^{*}_{\mathsf{R}} \Psi$. *Finally, natural derivations are closed under arbitrary contexts: for every context* $\Xi\square$, $\Phi \rightarrow_{\text{❀}} \Psi$ *implies* $\Xi\boxed{\Phi} \rightarrow_{\text{❀}} \Xi\boxed{\Psi}$. *We write* $\Phi \rightarrow_{\text{❀}} \Psi$ *to denote a*

Nature ❀

$$\frac{\Xi\boxed{}}{\Xi\boxed{\Phi}}\ \text{poll}{\downarrow} \qquad\qquad \frac{\Xi\boxed{\Phi}}{\Xi\boxed{}}\ \text{poll}{\uparrow}$$

$$\frac{\cdot \mathrel{\mathsf{D}} \cdot \Phi}{\Phi}\ \text{epis} \qquad\qquad \frac{}{\gamma \mathrel{\mathsf{D}} \cdot\ ; \Delta}\ \text{epet}$$

$$\frac{(\mathbf{x} \cdot \sigma(\Phi) \mathrel{\mathsf{D}} \sigma(\Delta)), (\mathbf{x}, \mathbf{y} \cdot \Phi \mathrel{\mathsf{D}} \Delta)}{\mathbf{x}, \mathbf{y} \cdot \Phi \mathrel{\mathsf{D}} \Delta}\ \text{ipis} \qquad\qquad \frac{\gamma \mathrel{\mathsf{D}} \mathbf{x} \cdot \sigma(\Phi); \mathbf{x}, \mathbf{y} \cdot \Phi; \Delta}{\gamma \mathrel{\mathsf{D}} \mathbf{x}, \mathbf{y} \cdot \Phi; \Delta}\ \text{ipet}$$

$$\frac{\mathbf{x} \cdot \Phi \mathrel{\mathsf{D}} \cdot \{\gamma_i \mathrel{\mathsf{D}} \Delta\}_i^n}{\mathbf{x} \cdot \Phi, (\cdot \mathrel{\mathsf{D}} \{\gamma_i\}_i^n) \mathrel{\mathsf{D}} \Delta}\ \text{srep}$$

Culture ✂

$$\frac{\Xi^+\boxed{\Phi}}{\Xi^+\boxed{}}\ \text{grow} \qquad\qquad \frac{\Xi^-\boxed{}}{\Xi^-\boxed{\Phi}}\ \text{crop}$$

$$\frac{\Xi^+\boxed{\gamma \mathrel{\mathsf{D}} \Delta}}{\Xi^+\boxed{\gamma \mathrel{\mathsf{D}} \Gamma; \Delta}}\ \text{pull} \qquad\qquad \frac{\Xi^-\boxed{\gamma \mathrel{\mathsf{D}} \Gamma; \Delta}}{\Xi^-\boxed{\gamma \mathrel{\mathsf{D}} \Delta}}\ \text{glue}$$

$$\frac{\Xi^+\boxed{\mathbf{x}, \mathbf{y} \cdot \Phi \mathrel{\mathsf{D}} \Delta}}{\Xi^+\boxed{\mathbf{x} \cdot \sigma(\Phi) \mathrel{\mathsf{D}} \sigma(\Delta)}}\ \text{apis} \qquad\qquad \frac{\Xi^-\boxed{\gamma \mathrel{\mathsf{D}} \mathbf{x}, \mathbf{y} \cdot \Phi; \Delta}}{\Xi^-\boxed{\gamma \mathrel{\mathsf{D}} \mathbf{x} \cdot \sigma(\Phi); \Delta}}\ \text{apet}$$

In the rules poll↓ and poll↑, we assume that $\Phi \succ \Xi\Box$.
In the rules ipis, apis (resp. ipet, apet), we assume some substitution $\sigma : \mathbf{y} \to \mathbb{T}$ that is capture-avoiding in $\cdot \Phi \mathrel{\mathsf{D}} \Delta$ (resp. Φ).

Figure 10.14.: Rules of the flower calculus

shallow *natural step, i.e. a direct instance of a natural rule in the empty context.*

The following lemma is the flower calculus equivalent of Lemma 9.6.1 for EGs:

Lemma 10.3.1 (Positive closure) *If* $\Phi \to \Psi$, *then* $\Xi^+\boxed{\Phi} \to \Xi^+\boxed{\Psi}$.

Proof. In the case of a natural step $\Phi \to_{❀} \Psi$, this is immediate by definition. Otherwise we have a cultural step $\Xi'\boxed{\Phi_0} \to_{✂} \Xi'\boxed{\Psi_0}$. Then either $\Xi'\square$ is positive, and $\mathsf{inv}(\Xi^+\boxed{\Xi'\square}) = \mathsf{inv}(\Xi^+\square) + \mathsf{inv}(\Xi'\square)$ is even since it is the sum of two even numbers; or $\Xi'\square$ is negative, and $\mathsf{inv}(\Xi^+\boxed{\Xi'\square})$ is odd since it is the sum of an even and an odd number. In both cases $\Xi^+\boxed{\Xi'\square}$ has the same polarity as $\Xi'\square$, and thus the same rule can be applied. $\square$

Now we can define our usual "goal-oriented" notion of proof:

Definition 10.3.7 (Proof) *A* proof *of a bouquet* Φ *is a derivation* $\Phi \to^* \varnothing$.

In Peircean terms, the empty bouquet is the blank SA. Then proving a bouquet amounts to erasing it completely from SA, thus reducing it to vacuous truth. Figure 10.18 shows an example of ❀-proof in the flower calculus, both in textual and graphical syntax. Note that we used a non-duplicating version of the rules ipis and ipet, in order to save some space in the graphical presentation.

If we want to speak about *relative* truth, i.e. Φ is true under the assumption that Ψ is, we can simply rely on the existence of a derivation $\Phi \to^* \Psi$ in the full flower calculus. This will be justified by the soundness of all rules (Theorem 10.5.13) as well as a *strong* completeness result (Corollary 10.6.7), that relies on the following strong deduction theorem:

Theorem 10.3.2 (Strong deduction) $\Phi \to^* \Psi$ *if and only if* $\Psi \sqsupset \Phi \to^* \varnothing$.

Proof. Suppose that $\Phi \to^* \Psi$. Then we have:

$$\begin{array}{llll}
\Psi \sqsupset \Phi & \to^* & \Psi \sqsupset \Psi & \text{(Hypothesis + Lemma 10.3.1)} \\
 & \to_{\mathsf{poll}\downarrow} & \Psi \sqsupset \cdot & \\
 & \to_{\mathsf{epet}} & \varnothing &
\end{array}$$

In the other direction, suppose that $\Psi \supset \Phi \to^* \varnothing$. Then we have:

$$\begin{array}{lll}
\Phi & \to_{\mathsf{epis}} & \supset \Phi \\
 & \to_{\mathsf{grow}} & (\Psi \supset \Phi), (\supset \Phi) \\
 & \to_{\mathsf{poll}\uparrow} & (\Psi \supset \Phi), ((\Psi \supset \Phi) \supset \Phi) \\
 & \to^* & (\Psi \supset \Phi) \supset \Phi \qquad\qquad \text{(Hypothesis + Lemma 10.3.1)} \\
 & \to_{\mathsf{grow}} & \Psi, ((\Psi \supset \Phi) \supset \Phi) \\
 & \to_{\mathsf{poll}\downarrow} & \Psi, ((\supset \Phi) \supset \Phi) \\
 & \to_{\mathsf{srep}} & \Psi, (\supset (\Phi \supset \Phi)) \\
 & \to_{\mathsf{poll}\downarrow} & \Psi, (\supset (\Phi \supset \cdot)) \\
 & \to_{\mathsf{epet}} & \Psi, (\supset \cdot) \\
 & \to_{\mathsf{epet}} & \Psi
\end{array}$$

□

Contrary to full derivability, natural derivability $\Phi \to^*_{❀} \Psi$ is too weak to satisfy a strong deduction theorem. This is a consequence of the fact that ❀-rules are *invertible*, and thus can only relate equivalent bouquets. Indeed, as soon as $\Psi \supset \Phi$ is ❀-provable but the converse $\Phi \supset \Psi$ is not, it follows from the completeness of ❀-rules that Φ and Ψ are not equivalent: thus $\Phi \not\to^*_{❀} \Psi$, contradicting the strong deduction statement.

$\Psi \supset \Phi \to^* \varnothing$. In fact this is closer to what one would find in sequent calculus, where hypothetical proofs are closed derivations of hypothetical sequents, not open derivations. The difference is that sequents capture only the *first-order* implicative structure of logic[7], while flowers capture the full structure of intuitionistic predicate logic. This allows for a nice generalization of the notion of hypothetical provability, which will be useful in the completeness proof of Section 10.6.

7: As opposed to *higher-order*, i.e. negatively nested implications.

Definition 10.3.8 (Hypothetical provability) *Given two bouquets Φ, Ψ, we say that Φ is* hypothetically provable *from Ψ in a fragment* R *of rules, written $\Psi \vdash_{\mathsf{R}} \Phi$, if for every context $\Xi\square$ such that $\Psi \succ \Xi\square$, $\Xi\boxed{\Phi} \to_{\mathsf{R}} \Xi\boxed{}$. We write $\Psi \vdash \Phi$ to denote hypothetical provability in the full flower calculus.*

Lemma 10.3.3 (Reflexivity) *For any bouquet Φ, $\Phi \vdash_{❀} \Phi$.*

Proof. For any context $\Xi\square$ such that $\Phi \succ \Xi\square$, one has the following derivation:

$$\Xi\boxed{\Phi} \to_{\mathsf{poll}\downarrow} \Xi\boxed{}$$

□

There is a subtle but important shift here with respect to the standard notions of hypothetical provability, as found in Gentzen systems or type theories: while in these settings it is characterized as the existence of a proof for a *single* hypothetical judgment $\Gamma \Rightarrow C$ which constrains the space of derivations, here we have the stronger requirement that there exist proofs for a *class* of judgments $\Xi\boxed{\Phi}$, whose hypothetical shape comes from the condition that $\Psi \succ \Xi\square$. In practice, the pollination rules $\{\mathsf{poll}{\downarrow}, \mathsf{poll}{\uparrow}\}$ and the epis rule make this equivalent to the existence of a proof for $\Psi \supset \Phi$. But we conjecture that the epis rule might be admissible modulo the addition of the distributivity rule crep[8,9] of Figure 10.15.

Thus our stronger notion of hypothetical provability makes more sense in the variant ❀ $\setminus \{\mathsf{epis}\} \cup \{\mathsf{crep}\}$ of the flower calculus, although it will still be useful in this work to make meta-theoretical proofs slightly shorter. For now we allow the epis rule, which renders the deduction theorem trivial:

Theorem 10.3.4 (Deduction) *For any pair Φ, Ψ of bouquets, $\Psi \vdash_{❀} \Phi$ if and only if $\vdash_{❀} \Psi \supset \Phi$.*

Proof. Let $\Xi\square$ be some context. If $\Psi \vdash_{❀} \Phi$, then in particular $\Xi'\boxed{\Phi} \to_{❀} \Xi'\boxed{}$ for $\Xi'\square := \Xi\boxed{\Psi \supset \square}$. Thus we have:

$$\Xi\boxed{\Psi \supset \Phi} \to_{❀} \Xi\boxed{\Psi \supset \cdot} \to_{\mathsf{epet}} \Xi\boxed{}$$

In the other direction, let Ξ be some context such that $\Psi \succ \Xi\square$. If $\vdash_{❀} \Psi \supset \Phi$, then in particular $\Xi\boxed{\Psi \supset \Phi} \to_{❀} \Xi\boxed{}$. Thus we have:

$$\Xi\boxed{\Phi} \to_{\mathsf{epis}} \Xi\boxed{\supset \Phi} \to_{\mathsf{poll}\uparrow} \Xi\boxed{\Psi \supset \Phi} \to_{❀} \Xi\boxed{}$$

$\square$

8: crep stands for (c)ross-(rep)roduction, mirroring the distinction between cross-pollination and self-pollination, but with respect to the (s)elf-(r)eproduction rule srep.

9: It seems that the question of admissibility of epis is very similar to the question of admissibility of *release* rules discussed in Section 3.7. Both can be used to enable a *local* simulation of sequent calculus rules, but do not appear in real proofs because of the *global* power of link formation (B,F) and pollination (poll↓,poll↑) rules.

$$\frac{\gamma \supset \{\mathbf{x}_i \cdot \Phi_i, \Psi\}_i^n}{(\gamma \supset \{\mathbf{x}_i \cdot \Phi_i\}_i^n), \Psi}\ \mathsf{crep}$$

Figure 10.15.: Cross-reproduction rule

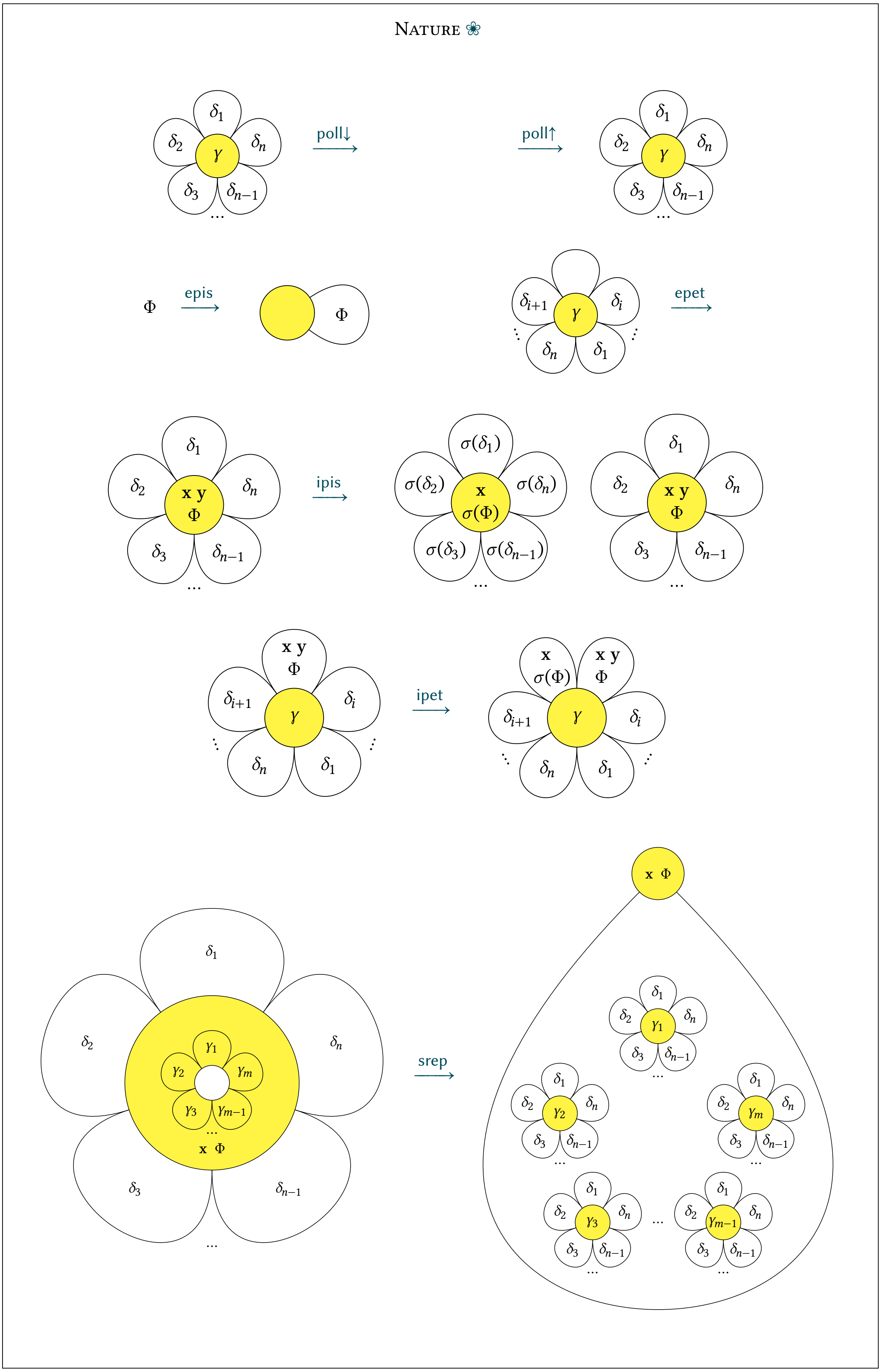


Figure 10.16.: Graphical presentation of the natural rules

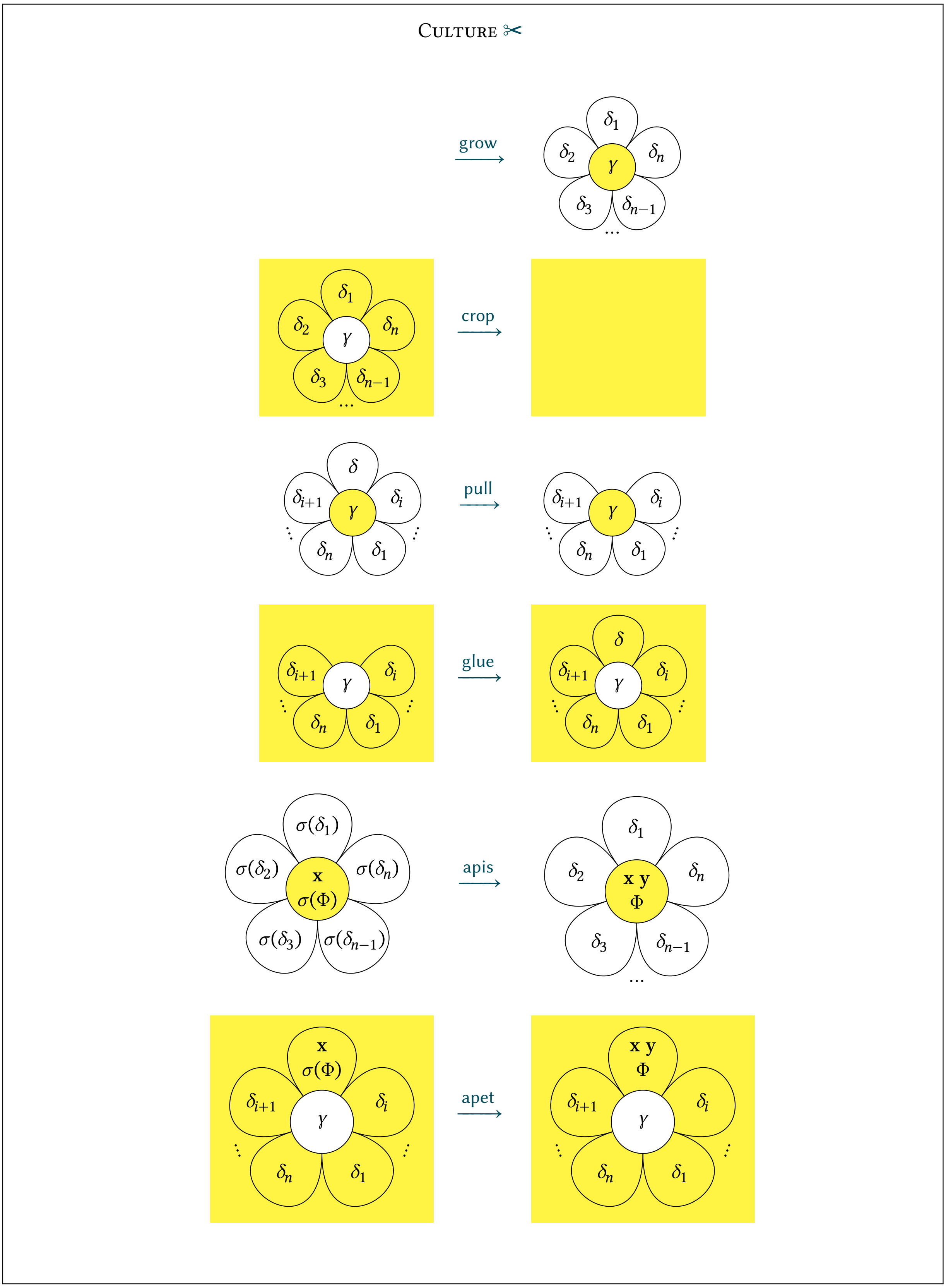


Figure 10.17.: Graphical presentation of the cultural rules

$$
\begin{array}{ll}
 & (x \cdot\ \mathsf{D}\ (p(x)\ \mathsf{D});\ q(x))\ \mathsf{D}\ (y \cdot p(y)\ \mathsf{D}\ z \cdot q(z)) \\
\rightarrow_{\mathsf{ipet}} & (x \cdot\ \mathsf{D}\ (p(x)\ \mathsf{D});\ q(x))\ \mathsf{D}\ (y \cdot p(y)\ \mathsf{D}\ q(y)) \\
\rightarrow_{\mathsf{poll}\uparrow} & (x \cdot\ \mathsf{D}\ (p(x)\ \mathsf{D});\ q(x))\ \mathsf{D}\ (y \cdot p(y), (x \cdot\ \mathsf{D}\ (p(x)\ \mathsf{D});\ q(x))\ \mathsf{D}\ q(y)) \\
\rightarrow_{\mathsf{ipis}} & (x \cdot\ \mathsf{D}\ (p(x)\ \mathsf{D});\ q(x))\ \mathsf{D}\ (y \cdot p(y), (\mathsf{D}\ (p(y)\ \mathsf{D});\ q(y))\ \mathsf{D}\ q(y)) \\
\rightarrow_{\mathsf{srep}} & (x \cdot\ \mathsf{D}\ (p(x)\ \mathsf{D});\ q(x))\ \mathsf{D}\ (y \cdot p(y)\ \mathsf{D}\ ((p(y)\ \mathsf{D})\ \mathsf{D}\ q(y)), (q(y)\ \mathsf{D}\ q(y))) \\
\rightarrow^{2}_{\mathsf{poll}\downarrow} & (x \cdot\ \mathsf{D}\ (p(x)\ \mathsf{D});\ q(x))\ \mathsf{D}\ (y \cdot p(y)\ \mathsf{D}\ ((\mathsf{D})\ \mathsf{D}\ q(y)), (q(y)\ \mathsf{D}\ \cdot)) \\
\rightarrow_{\mathsf{srep}} & (x \cdot\ \mathsf{D}\ (p(x)\ \mathsf{D});\ q(x))\ \mathsf{D}\ (y \cdot p(y)\ \mathsf{D}\ (\mathsf{D}\ \cdot), (q(y)\ \mathsf{D}\ \cdot)) \\
\rightarrow^{2}_{\mathsf{epet}} & (x \cdot\ \mathsf{D}\ (p(x)\ \mathsf{D});\ q(x))\ \mathsf{D}\ (y \cdot p(y)\ \mathsf{D}\ \cdot) \\
\rightarrow_{\mathsf{epet}} & (x \cdot\ \mathsf{D}\ (p(x)\ \mathsf{D});\ q(x))\ \mathsf{D}\ \cdot \\
\rightarrow_{\mathsf{epet}} &
\end{array}
$$

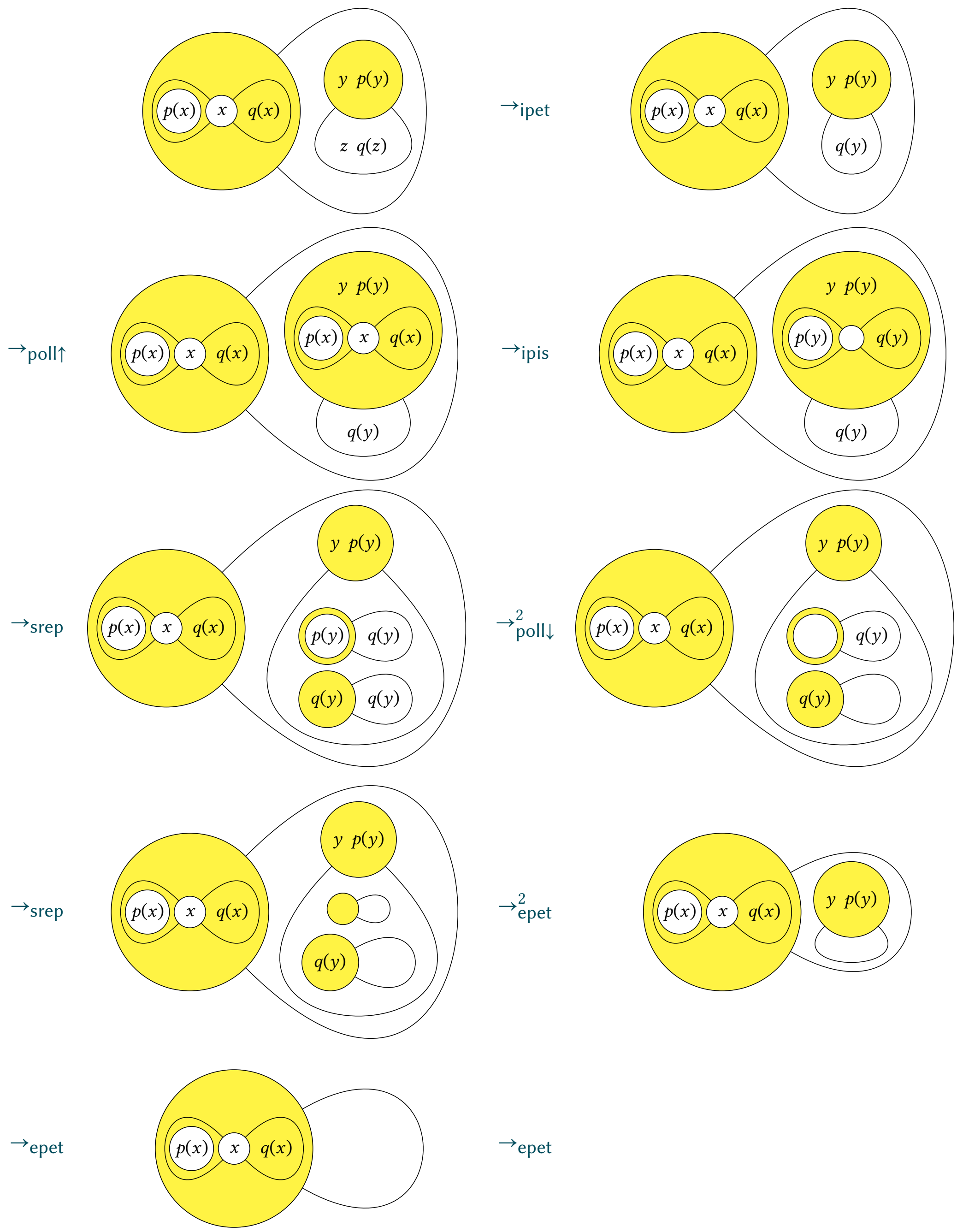


Figure 10.18.: A natural proof in the flower calculus

10.4. Kripke semantics

In Section 8.6, in order to prove the soundness of system B, we gave a bi-intuitionistic (resp. classical) semantics to bubbles in Heyting-Brouwer (resp. Boolean) algebras, and in Section 9.5 we interpreted α-graphs with simple boolean truth values. For flowers, we use a third kind of models that is standard in the literature on intuitionistic logic: Kripke structures. Our goal is not to fill as many pages as possible by introducing new definitions in each chapter, but rather to find the simplest semantics for our purposes with the specific proof system at hand. In the case of the flower calculus, there are a few reasons that led us to the choice of Kripke structures, detailed hereafter by order of increasing importance:

Generalization of EGs As mentioned in Section 10.1, the syntax of intuitionistic EGs (and thus of flowers) subsumes that of Peirce's classical EGs, by seeing the cut as a 0-ary scroll (or a flower with no petals). Similarly, it is well-known that Kripke structures enable a natural generalization of truth valuations, where classical valuations are those limited to "unary" structures with a single possible world[10].

10: It is tempting to draw a parallel between *possible worlds* and the *petals* of flowers, where the accessibility relation $\leq$ between worlds might be reflected to some extent in the pollination relation illustrated in Figure 10.12.An analogy between possible worlds and intuitionistic disjunction is also proposed by Girard in [96], although as usual he demonstrates his (free) criticism towards Kripke semantics.

Quantifiers It is easy to accomodate quantifiers in Kripke structures, by interpreting terms as individuals in the domains of the structure's worlds. In contrast, extensions of algebraic semantics that account for quantifiers like polyadic and cylindric algebras are more involved, and less studied in the literature.

Cut-free completeness Rather than just completeness, we are interested more specifically in the completeness of the natural fragment ❀ of the flower calculus, where all rules are *analytic*. In Gentzen systems, this corresponds to the well-known questions of *proof normalization* (natural deduction) and *cut admissibility* (sequent calculus). Gentzen originally proved cut admissibility through a syntactic procedure of *cut-elimination*, which is very hard to transpose in our deep inference setting, especially since there is no known internal cut-elimination procedure in the (sparse) literature on deep inference systems for full intuitionistic logic. In fact, the requirement that the procedure be *internal* is useful when studying the computational content of proofs, but too strong for our logical study of analyticity. Thus our first attempt was to devise an *external* syntactic procedure, based on the simulation of a cut-free sequent calculus as in [241]; and we do have a working, verified implementation of this procedure in `Coq` [63]. However, we realized *a posteriori* that this only proves a weak form of completeness, in the sense that we only guarantee that a true flower is provable if it is the direct translation of a symbolic formula. But formulas are based on binary connectives and unary quantifiers, and are thus less expressive syntactically than flowers and their n-ary constructs.

[241]: Tiu (2006), 'A Local System for Intuitionistic Logic'

To palliate this limitation, we turned to a more *semantic* (and not so-well known) strand of analyticity proofs, which nowadays tends to be labelled *normalization-by-evaluation* [1]. The idea is to prove completeness of the analytic fragment of the proof system with respect to some semantic models (in our case, $\vDash \phi$ implies $\vdash_{❀} \phi$), and then compose with the soundness of the full system with respect to the same models

[1]: Abel (2013), 'Normalization by Evaluation: Dependent Types and Impredicativity'

($\vdash \phi$ implies $\vDash \phi$), giving the desired admissibility result ($\vdash \phi$ implies $\vdash_{❀} \phi$). And in the case of intuitionistic logics, the models used are most of the time Kripke structures[11] [119, 120, 129].

11: We know of two exceptions: the first is the (complex) uniform completeness proof for various logics proposed by Okada [188] and based on *phase spaces*, which are the closest one can get to a truth-based (or Tarskian) semantics for *linear logic* [92]. The second is the recent completeness proof of Frumin for the logic of Bunched Implications (BI) [83], a cousin of linear logic at the heart of *separation logic*, which is a very popular framework in contemporary deductive program verification. It is based on so-called *BI algebras*, a kind of Heyting algebra with additional structure for the linear part of the logic. We should also mention the recent normalization proof for *cubical type theory* given by Sterling in his thesis [230], based on a categorical and topos-theoretic generalization of Kripke structures.

[119]: Herbelin et al. (2009), 'Forcing-Based Cut-Elimination for Gentzen-Style Intuitionistic Sequent Calculus'
[120]: Hermant (2005), 'Semantic Cut Elimination in the Intuitionistic Sequent Calculus'
[129]: Ilik (2010), 'Constructive Completeness Proofs and Delimited Control'

We now recall the standard definitions, starting with the interpretation of constants in *first-order structures*:

Definition 10.4.1 (First-order structure) *A first-order structure is a pair $(M, [\![\cdot]\!])$ where M is a non-empty set, and $[\![\cdot]\!]$ is a map called the interpretation that associates to each function symbol $f \in \mathscr{F}$ a function $[\![f]\!] : M^{\mathrm{ar}(f)} \to M$, and to each predicate symbol $p \in \mathscr{P}$ a relation $[\![p]\!] \subseteq M^{\mathrm{ar}(p)}$.*

Definition 10.4.2 (Kripke structure) *A Kripke structure is a triplet $\mathscr{K} = (W, \leq, (M_w)_{w\in W})$, where W is the set of worlds, $\leq$ is a pre-order on W called accessibility, and $(M_w)_{w\in W}$ is a family of first-order structures indexed by W. Furthermore, we require the following monotonicity conditions to hold whenever $w \leq w'$:*

- $M_w \subseteq M_{w'}$;
- *for every* $f \in \mathscr{F}$, $[\![f]\!]_w \subseteq [\![f]\!]_{w'}$;
- *for every* $p \in \mathscr{P}$, $[\![p]\!]_w \subseteq [\![p]\!]_{w'}$.

Then we need a way to interpret arbitrary terms with free variables in any given world w of a Kripke structure, which is done through the concept of *w-evaluation*:

Definition 10.4.3 (w-evaluation) *Given a Kripke structure $\mathscr{K}$ and a world w in $\mathscr{K}$, a w-evaluation is a function $e : \mathscr{V} \to M_w$. The interpretation map of M_w is extended to terms and substitutions with respect to any evaluation e as follows:*

$$[\![x]\!]_e = e(x) \qquad [\![f(\vec{t})]\!]_e = [\![f]\!]_w([\![\vec{t}]\!]_e) \qquad [\![\sigma]\!]_e(x) = [\![\sigma(x)]\!]_e$$

$$\text{Substitution } \sigma : \mathscr{V} \to \mathbb{T}$$
$$\text{Evaluation } e : \mathscr{V} \to M_w$$

Figure 10.19.: The syntax-semantics mirror

The crux of Kripke semantics is the *forcing* relation, that captures the truth-conditions of statements in Kripke structures. While it is usually defined on formulas, here we adapt the definition to flowers, which in our opinion makes it simpler and more uniform since flowers can be seen as built from essentially one big constructor:

Definition 10.4.4 (Forcing) *Given some Kripke structure $\mathscr{K}$, the forcing relation $w \Vdash \phi\,[e]$ between a world w, a flower ϕ and a w-evaluation e is defined by induction on $|\phi|$ as follows:*

(Atom) $w \Vdash p(\vec{t})\,[e]$ *iff* $[\![\vec{t}]\!]_e \in [\![p]\!]_w$;

(Flower) $w \Vdash \mathbf{x} \cdot \Phi \supset \{\mathbf{x}_i \cdot \Phi_i\}_i^n\,[e]$ *iff for every $w' \geq w$ and every w'-evaluation e', if $w' \Vdash \Phi\,[e\,|_{\mathbf{x}}\,e']$ then there is some $1 \leq i \leq n$ and*

w'-evaluation e'' such that $w' \Vdash \Phi_i\, [e \,|_{\mathbf{x}}\, e' \,|_{\mathbf{x}_i}\, e'']$.

(Bouquet) *$w \Vdash \Phi\, [e]$ iff $w \Vdash \phi\, [e]$ for every $\phi \in \Phi$.*

Lastly, we define the notion of *semantic entailment* $\Phi \vDash \Psi$ on bouquets, mirroring the syntactic entailment $\Phi \vdash \Psi$ of the last section:

Definition 10.4.5 (Semantic entailment) *Let $\mathcal{K}$ be a Kripke structure, and Φ, Ψ some bouquets. We say that Φ* semantically entails *Ψ in $\mathcal{K}$, written $\Phi \vDash_{\mathcal{K}} \Psi$, when $w \Vdash \Phi\, [e]$ implies $w \Vdash \Psi\, [e]$ for every world $w \in W$ and w-evaluation e. This entailment is* valid *if it holds for any Kripke structure $\mathcal{K}$, and in that case we simply write $\Phi \vDash \Psi$. We say that Φ is* semantically equivalent *to Ψ, written $\Phi \mathrel{=\!\!||\!\!\vDash} \Psi$, when $\Phi \vDash \Psi$ and $\Psi \vDash \Phi$.*

10.5. Soundness

In this section, we show that every rule of the flower calculus is *sound* with respect to our Kripke semantics for flowers, and thus that $\vdash \phi$ implies $\vDash \phi$ for every ϕ. We start with a few trivial facts about Definition 10.2.7:

Fact 10.5.1 (Associativity) $f \,|_R\, g \,|_S\, h = f \,|_{R \cup S}\, (g \,|_S\, h)$.

Fact 10.5.2 (Commutativity) If $R \cap S = \varnothing$ then $f \,|_R\, g \,|_S\, h = f \,|_S\, h \,|_R\, g$.

Fact 10.5.3 (Agreement) If $f(x) = g(x)$ for all $x \in R$ then $h \,|_R\, f = h \,|_R\, g$.

Fact 10.5.4 (Idempotency) $f \,|_R\, f = f$.

Semantic entailment is obviously a reflexive and transitive relation:

Fact 10.5.5 (Reflexivity) $\Phi \vDash \Phi$.

Fact 10.5.6 (Transitivity) If $\Phi \vDash \Psi$ and $\Psi \vDash \Xi$, then $\Phi \vDash \Xi$.

The two following lemmas will be useful to reason on the forcing relation (Definition 10.4.4):

Lemma 10.5.1 (Monotonicity) *If $w \leq w'$ and $w \Vdash \phi\, [e]$ then $w' \Vdash \phi\, [e]$.*

Proof. By a straightforward induction on $|\phi|$. □

Lemma 10.5.2 (Mirroring) *$w \Vdash \sigma(\phi)\,[e]$ iff $w \Vdash \phi\,[e\,|_{\mathbf{x}}\,[\![\sigma]\!]_e]$ for $\sigma : \mathbf{x} \to \mathbb{T}$ capture-avoiding in ϕ and $\mathbf{x} \cap \mathrm{bv}(\phi) = \varnothing$.*

Proof. By induction on $|\phi|$.

(Base case) Suppose $\phi = p(\vec{\mathbf{t}})$. We show that $[\![\sigma(\vec{\mathbf{t}})]\!]_e \in [\![p]\!]_w$ iff $[\![\vec{\mathbf{t}}]\!]_{e|_{\mathbf{x}}[\![\sigma]\!]_e} \in [\![p]\!]_w$ by proving that $[\![t]\!]_{e|_{\mathbf{x}}[\![\sigma]\!]_e} = [\![\sigma(t)]\!]_e$ for any term t by induction on $|t|$.

- If $t = x$, then either:
  - $x \in \mathbf{x}$, and $[\![x]\!]_{e|_{\mathbf{x}}[\![\sigma]\!]_e} = [\![\sigma]\!]_e(x) = [\![\sigma(x)]\!]_e$; or
  - $x \notin \mathbf{x}$, and $[\![x]\!]_{e|_{\mathbf{x}}[\![\sigma]\!]_e} = e(x) = [\![x]\!]_e = [\![\sigma(x)]\!]_e$.
- If $t = f(\vec{\mathbf{t}})$, then

$$\begin{array}{rcll} [\![f(\vec{\mathbf{t}})]\!]_{e|_{\mathbf{x}}[\![\sigma]\!]_e} & = & [\![f]\!]_w\left([\![\vec{\mathbf{t}}]\!]_{e|_{\mathbf{x}}[\![\sigma]\!]_e}\right) & \\ & = & [\![f]\!]_w\left([\![\sigma(\vec{\mathbf{t}})]\!]_e\right) & \text{(IH)} \\ & = & [\![f(\sigma(\vec{\mathbf{t}}))]\!]_e & \\ & = & [\![\sigma(f(\vec{\mathbf{t}}))]\!]_e & \end{array}$$

(Recursive case) Suppose $\phi = \mathbf{y} \cdot \Phi \sqsupset \{\mathbf{z}_i \cdot \Psi_i\}_i^n$. We show that $w \Vdash \mathbf{y} \cdot \sigma(\Phi) \sqsupset \{\mathbf{z}_i \cdot \sigma(\Psi_i)\}_i^n\,[e]$ implies $w \Vdash \mathbf{y} \cdot \Phi \sqsupset \{\mathbf{z}_i \cdot \Psi_i\}_i^n\,[e\,|_{\mathbf{x}}\,[\![\sigma]\!]_e]$, the argument working in both directions. Let $w' \geq w$ and e' a w'-evaluation such that $w' \Vdash \Phi\,[e|_{\mathbf{x}}\,[\![\sigma]\!]_e|_{\mathbf{y}}\,e']$. Since σ is capture-avoiding in ϕ, we know that $\mathrm{fv}(\sigma(x)) \cap \mathbf{y} = \varnothing$, and thus $[\![\sigma]\!]_e(x) = [\![\sigma(x)]\!]_e = [\![\sigma(x)]\!]_{e|_{\mathbf{y}}e'} = [\![\sigma]\!]_{e|_{\mathbf{y}}e'}(x)$ for any $x \in \mathbf{x}$. Hence by Fact 10.5.3 $w' \Vdash \Phi\,[e\,|_{\mathbf{x}}\,[\![\sigma]\!]_{e|_{\mathbf{y}}e'}\,|_{\mathbf{y}}\,e']$, and since by hypothesis $\mathbf{x} \cap \mathbf{y} = \varnothing$ we obtain $w' \Vdash \Phi\,[e\,|_{\mathbf{y}}\,e'\,|_{\mathbf{x}}\,[\![\sigma]\!]_{e|_{\mathbf{y}}e'}]$ by Fact 10.5.2. Then by IH we get $w' \Vdash \sigma(\Phi)\,[e\,|_{\mathbf{y}}\,e']$, and thus by hypothesis $w' \Vdash \sigma(\Psi_i)\,[e\,|_{\mathbf{y}}\,e'\,|_{\mathbf{z}_i}\,e'']$ for some $1 \leq i \leq n$ and w'-evaluation e''. Again by IH we get $w' \Vdash \Psi_i\,[e\,|_{\mathbf{y}}\,e'\,|_{\mathbf{z}_i}\,e''\,|_{\mathbf{x}}\,[\![\sigma]\!]_{e|_{\mathbf{y}}e'|_{\mathbf{z}_i}e''}]$, and since σ is capture-avoiding in ϕ we have $\mathrm{fv}(\sigma(x)) \cap \mathbf{z}_i = \varnothing$ for any $x \in \mathbf{x}$, and thus $w' \Vdash \Psi_i\,[e|_{\mathbf{y}}\,e'\,|_{\mathbf{z}_i}\,e''\,|_{\mathbf{x}}\,[\![\sigma]\!]_e]$ by Fact 10.5.3. Finally by hypothesis $\mathbf{x} \cap \mathbf{z}_i = \varnothing$, thus we can conclude that $w' \Vdash \Psi_i\,[e\,|_{\mathbf{x}}\,[\![\sigma]\!]_e\,|_{\mathbf{y}}\,e'\,|_{\mathbf{z}_i}\,e'']$ by Fact 10.5.2.

□

The following *functoriality* lemma is at the heart of every deep inference formalism[12]:

12: Other instances in this thesis are Lemma 3.4.2, Lemma 8.6.14, Lemma 8.6.19, and Corollary 9.5.4.

Lemma 10.5.3 (Functoriality) *If $\Phi \vDash \Psi$, then for any $\Xi\Box$ either $\Xi\boxed{\Phi} \vDash \Xi\boxed{\Psi}$ if $\Xi\Box$ is positive, or $\Xi\boxed{\Psi} \vDash \Xi\boxed{\Phi}$ if $\Xi\Box$ is negative.*

Proof. By induction on $|\Xi\Box|$. □

Lemma 10.5.4 (Weakening) $\Phi \vDash \varnothing$.

Proof. Trivial by Definition 10.4.4. □

Lemma 10.5.5 (Co-weakening) $\gamma \supset \Delta \models \gamma \supset \Gamma ; \Delta$.

Proof. Let $\gamma = \mathbf{x} \cdot \Phi$, w a world in some Kripke structure $\mathscr{K}$, $w' \geq w$, e a w-evaluation and e' a w'-evaluation such that $w \Vdash \gamma \supset \Delta\,[e]$ and $w' \Vdash \Phi\,[e|_{\mathbf{x}} e']$. Then by hypothesis there must exist some $\mathbf{y} \cdot \Psi \in \Delta$ and w'-evaluation e'' such that $w' \Vdash \Psi\,[e\,|_{\mathbf{x}}\, e'\,|_{\mathbf{y}}\, e'']$, and thus we can conclude. □

The less obvious rules in terms of soundness are the *pollination* rules $\{\mathsf{poll}{\downarrow}, \mathsf{poll}{\uparrow}\}$, because of the arbitrary context $\Xi\square$ and reliance on the pollination relation.

Lemma 10.5.6 (Cross-pollination) $\Phi, \Xi\boxed{\Phi} \dashv\models \Phi, \Xi\boxed{}$.

Proof. Let w a world in some Kripke structure $\mathscr{K}$, and e a w-evaluation. We show that $w \Vdash \Phi, \Xi\boxed{\Phi}\,[e]$ iff $w \Vdash \Phi, \Xi\boxed{}\,[e]$ by induction on $|\Xi\square|$.

Base case Suppose $\Xi\square = \Xi', \square$. Then we trivially have $w \Vdash \Phi, \Xi', \Phi\,[e]$ iff $w \Vdash \Phi, \Xi'\,[e]$ by Definition 10.4.4.

Recursive case We distinguish two cases:

Pistil Suppose $\Xi\square = \Xi', (\mathbf{x} \cdot \Xi_0 \supset \Delta)$.

1. Suppose $w \Vdash \Phi, \Xi\boxed{\Phi}\,[e]$. Then $w \Vdash \Phi\,[e]$, $w \Vdash \Xi'\,[e]$ and $w \Vdash \mathbf{x} \cdot \Xi_0\boxed{\Phi} \supset \Delta\,[e]$. Thus it remains to show that $w \Vdash \mathbf{x} \cdot \Xi_0\boxed{} \supset \Delta\,[e]$. Let $w' \geq w$ and e' a w'-evaluation such that $w' \Vdash \Xi_0\boxed{}\,[e\,|_{\mathbf{x}}\, e']$. By IH we have $\Phi, \Xi_0\boxed{} \models \Phi, \Xi_0\boxed{\Phi}$, and thus by Lemma 10.5.3 $\mathbf{x} \cdot \Phi, \Xi_0\boxed{\Phi} \supset \Delta \models \mathbf{x} \cdot \Phi, \Xi_0\boxed{} \supset \Delta$. By Lemma 10.5.4 and Lemma 10.5.3 we have $w \Vdash \mathbf{x} \cdot \Phi, \Xi_0\boxed{\Phi} \supset \Delta\,[e]$, and thus $w \Vdash \mathbf{x} \cdot \Phi, \Xi_0\boxed{} \supset \Delta\,[e]$. Then since $w' \Vdash \Xi_0\boxed{}\,[e\,|_{\mathbf{x}}\, e']$, and since by Lemma 10.5.1 (and the fact that $\mathbf{x} \cap \mathsf{fv}(\Phi) = \varnothing$) we have $w' \Vdash \Phi\,[e\,|_{\mathbf{x}}\, e']$, we can conclude that there are some $\mathbf{y} \cdot \Psi \in \Delta$ and w'-evaluation e'' such that $w' \Vdash \Psi\,[e\,|_{\mathbf{x}}\, e'\,|_{\mathbf{y}}\, e'']$.
2. $\Phi, \Xi\boxed{} \models \Phi, \Xi\boxed{\Phi}$ holds by the same argument in the other direction.

Petal Suppose $\Xi\square = \Xi', (\mathbf{x} \cdot \Psi \supset \mathbf{y} \cdot \Xi_0 ; \Delta)$.

1. Suppose $x \Vdash \Phi, \Xi\boxed{\Phi}\,[e]$. Then $w \Vdash \Phi\,[e]$, $w \Vdash \Xi'\,[e]$ and $w \Vdash \mathbf{x} \cdot \Psi \supset \mathbf{y} \cdot \Xi_0\boxed{\Phi} ; \Delta\,[e]$. Thus it remains to show that $w \Vdash \mathbf{x} \cdot \Psi \supset \mathbf{y} \cdot \Xi_0\boxed{} ; \Delta\,[e]$. Let $w' \geq w$ and e' a w'-evaluation such that $w' \Vdash \Psi\,[e\,|_{\mathbf{x}}\, e']$. Then we can deduce that there exists a w'-evaluation e'' such that either:

   - $w' \Vdash \Psi'\,[e\,|_{\mathbf{x}}\, e'\,|_{\mathbf{y}'}\, e'']$ for some $\mathbf{y}' \cdot \Psi' \in \Delta$, and we conclude immediately;

- or $w' \Vdash \Xi_0\boxed{\Phi}[e \mid_{\mathbf{x}} e' \mid_{\mathbf{y}} e'']$. By Lemma 10.5.1 (and the fact that $\mathbf{x} \cap \mathsf{fv}(\Phi) = \emptyset$ and $\mathbf{y} \cap \mathsf{fv}(\Phi) = \emptyset$) we have $w' \Vdash \Phi\,[e \mid_{\mathbf{x}} e' \mid_{\mathbf{y}} e'']$, and thus $w' \Vdash \Phi, \Xi_0\boxed{\Phi}[e \mid_{\mathbf{x}} e' \mid_{\mathbf{y}} e'']$. Then by IH we have $w' \Vdash \Phi, \Xi_0\boxed{}[e \mid_{\mathbf{x}} e' \mid_{\mathbf{y}} e'']$, and thus we can conclude in particular that $w' \Vdash \Xi_0\boxed{}[e \mid_{\mathbf{x}} e' \mid_{\mathbf{y}} e'']$.

2. $\Phi, \Xi\boxed{} \vDash \Phi, \Xi\boxed{\Phi}$ holds by the same argument in the other direction.

□

Lemma 10.5.7 (Pollination) *If $\Phi \succ \Xi\Box$, then $\Xi\boxed{\Phi} \dashv\vDash \Xi\boxed{}$.*

Proof. We show that $\phi \succ \Xi\Box$ implies $\Xi\boxed{\phi} \dashv\vDash \Xi\boxed{}$ for any flower ϕ and context $\Xi\Box$: then assuming that $\Phi = \phi_1, \ldots, \phi_n$, we get

$$\underbrace{\Xi\boxed{\phi_1, \ldots, \phi_n} \dashv\vDash \Xi\boxed{\phi_2, \ldots, \phi_n} \dashv\vDash \ldots \dashv\vDash \Xi\boxed{}}_{n \text{ times}}$$

and conclude by Fact 10.5.6.

By Definition 10.3.5, there are a bouquet Ψ and two contexts $\Xi'\Box, \Xi_0$ such that one of the two following cases holds:

Cross-pollination $\Xi\Box = \Xi'\boxed{\Psi, \phi, \Xi_0}$. Then $\phi, \Xi_0\boxed{\phi} \dashv\vDash \phi, \Xi_0\boxed{}$ by Lemma 10.5.6, and we conclude by Lemma 10.5.3.

Self-pollination $\Xi\Box = \Xi'\boxed{\mathbf{x} \cdot \Psi, \phi \mathbin{\mathsf{D}} \mathbf{y} \cdot \Xi_0\,;\, \Delta}$ for some $\mathbf{x}, \mathbf{y}, \Delta$. Let w a world in some Kripke structure $\mathscr{K}$ and e a w-evaluation. We show that $w \Vdash \mathbf{x} \cdot \Psi, \phi \mathbin{\mathsf{D}} \mathbf{y} \cdot \Xi_0\boxed{\phi}\,;\, \Delta\,[e]$ iff $w \Vdash \mathbf{x} \cdot \Psi, \phi \mathbin{\mathsf{D}} \mathbf{y} \cdot \Xi_0\boxed{}\,;\, \Delta\,[e]$, and conclude by Lemma 10.5.3.

1. Suppose that $w \Vdash \mathbf{x} \cdot \Psi, \phi \mathbin{\mathsf{D}} \mathbf{y} \cdot \Xi_0\boxed{\phi}\,;\, \Delta\,[e]$, and let $w' \geq w$ and e' a w'-evaluation such that $w' \Vdash \Psi, \phi\,[e \mid_{\mathbf{x}} e']$. Then we can deduce that there exists a w'-evaluation e'' such that either:

   - $w' \Vdash \Psi'\,[e \mid_{\mathbf{x}} e' \mid_{\mathbf{y}'} e'']$ for some $\mathbf{y}' \cdot \Psi' \in \Delta$, and we conclude immediately;
   - or $w' \Vdash \Xi_0\boxed{\phi}[e \mid_{\mathbf{x}} e' \mid_{\mathbf{y}} e'']$. Since $\mathsf{fv}(\phi) \cap \mathbf{y} = \emptyset$ we have $w' \Vdash \phi\,[e \mid_{\mathbf{x}} e' \mid_{\mathbf{y}} e'']$, and thus $w' \Vdash \phi, \Xi_0\boxed{\phi}[e \mid_{\mathbf{x}} e' \mid_{\mathbf{y}} e'']$. Then by Lemma 10.5.6 we have $w' \Vdash \phi, \Xi_0\boxed{}[e \mid_{\mathbf{x}} e' \mid_{\mathbf{y}} e'']$, and thus we can conclude in particular that $w' \Vdash \Xi_0\boxed{}[e \mid_{\mathbf{x}} e' \mid_{\mathbf{y}} e'']$.

2. $\mathbf{x} \cdot \Psi, \phi \mathbin{\mathsf{D}} \mathbf{y} \cdot \Xi_0\boxed{}\,;\, \Delta \vDash \mathbf{x} \cdot \Psi, \phi \mathbin{\mathsf{D}} \mathbf{y} \cdot \Xi_0\boxed{\phi}\,;\, \Delta$ holds by the same argument in the other direction.

□

Proving the soundness of rules involving binders (ipis, ipet, apis, apet) is also quite tedious, which can be understood as stemming from the fact that substitutions simulate the complex dynamics of the LoIs of EGs in a *global* rather than local way. In particular, one needs to be careful about the scope of bound variables, which in EGs would be handled locally with (de)iteration rules on LoIs.

Lemma 10.5.8 (Universal instantiation) *If* $\sigma : \mathbf{y} \to \mathbb{T}$ *is capture-avoiding in* $\Phi \supset \Delta$*, then* $\mathbf{x}, \mathbf{y} \cdot \Phi \supset \Delta \vDash \mathbf{x} \cdot \sigma(\Phi) \supset \sigma(\Delta)$.

Proof. Let w a world in some Kripke structure $\mathscr{K}$, $w' \geq w$, e a w-evaluation and e' a w'-evaluation such that $w \Vdash \mathbf{x}, \mathbf{y} \cdot \Phi \supset \Delta\,[e]$ and $w' \Vdash \sigma(\Phi)\,[e\,|_{\mathbf{x}}\,e']$. Therefore $w' \Vdash \Phi\,[e\,|_{\mathbf{x}}\,e'\,|_{\mathbf{y}}\,[\![\sigma]\!]_{e|_{\mathbf{x}}e'}]$ by Lemma 10.5.2, and thus $w' \Vdash \Phi\,[e\,|_{\mathbf{x}\cup\mathbf{y}}\,(e'\,|_{\mathbf{y}}\,[\![\sigma]\!]_{e|_{\mathbf{x}}e'})]$ by Fact 10.5.1. Then by hypothesis, there must be some $\mathbf{z} \cdot \Psi \in \Delta$ and w'-evaluation e'' such that $w' \Vdash \Psi\,[e\,|_{\mathbf{x}\cup\mathbf{y}}\,(e'\,|_{\mathbf{y}}\,[\![\sigma]\!]_{e|_{\mathbf{x}}e'})\,|_{\mathbf{z}}\,e'']$, and thus $w' \Vdash \Psi\,[e\,|_{\mathbf{x}}\,e'\,|_{\mathbf{y}}\,[\![\sigma]\!]_{e|_{\mathbf{x}}e'}\,|_{\mathbf{z}}\,e'']$. Since σ is capture-avoiding in $\Phi \supset \Delta$, we know that for any $x \in \mathbf{y}$ we have $\mathrm{fv}(\sigma(x)) \cap \mathbf{z} = \varnothing$, and thus $[\![\sigma(x)]\!]_{e|_{\mathbf{x}}e'|_{\mathbf{z}}e''} = [\![\sigma(x)]\!]_{e|_{\mathbf{x}}e'}$. Hence by Fact 10.5.3 and Fact 10.5.2 we get $w' \Vdash \Psi\,[e\,|_{\mathbf{x}}\,e'\,|_{\mathbf{z}}\,e''\,|_{\mathbf{y}}\,[\![\sigma]\!]_{e|_{\mathbf{x}}e'|_{\mathbf{z}}e''}]$, and by Lemma 10.5.2 we conclude that $w' \Vdash \sigma(\Psi)\,[e\,|_{\mathbf{x}}\,e'\,|_{\mathbf{z}}\,e'']$. ☐

Lemma 10.5.9 (Existential instantiation) *If* $\sigma : \mathbf{y} \to \mathbb{T}$ *is capture-avoiding in* Φ*, then* $\gamma \supset \mathbf{x} \cdot \sigma(\Phi); \Delta \vDash \gamma \supset \mathbf{x}, \mathbf{y} \cdot \Phi; \Delta$.

Proof. Let $\gamma = \mathbf{z} \cdot \Xi$, and w a world in some Kripke structure $\mathscr{K}$, $w' \geq w$, e a w-evaluation and e' a w'-evaluation such that $w \Vdash \gamma \supset \mathbf{x} \cdot \sigma(\Phi); \Delta\,[e]$ and $w' \Vdash \Xi\,[e\,|_{\mathbf{z}}\,e']$. Then by hypothesis, there must be some w'-evaluation e'' such that either:

- $w' \Vdash \Xi'\,[e\,|_{\mathbf{z}}\,e'\,|_{\mathbf{z}'}\,e'']$ for some $\mathbf{z}' \cdot \Xi' \in \Delta$, and we conclude immediately;
- or $w' \Vdash \sigma(\Phi)\,[e\,|_{\mathbf{z}}\,e'\,|_{\mathbf{x}}\,e'']$. Then by Lemma 10.5.2 we have $w' \Vdash \Phi\,[e\,|_{\mathbf{z}}\,e'\,|_{\mathbf{x}}\,e''\,|_{\mathbf{y}}\,[\![\sigma]\!]_{e|_{\mathbf{z}}e'|_{\mathbf{x}}e''}]$, and thus we can conclude with $w' \Vdash \Phi\,[e\,|_{\mathbf{z}}\,e'\,|_{\mathbf{x}\cup\mathbf{y}}\,(e''\,|_{\mathbf{y}}\,[\![\sigma]\!]_{e|_{\mathbf{z}}e'|_{\mathbf{x}}e''})]$ by Fact 10.5.3.

☐

We are now equipped with enough lemmas to prove the soundness of each rule, starting with the *shallow* version of natural rules. In fact we are able to prove more: that every ❀-rule is *invertible*, i.e. its conclusion entails its premiss.

Lemma 10.5.10 (Shallow soundness) *If* $\Phi \rightharpoonup \Psi$*, then* $\Phi \mathrel{=\!\!\!|\!\vDash} \Psi$.

Proof. Let w a world in some Kripke structure $\mathscr{K}$, $w' \geq w$, e a w-evaluation and e' a w'-evaluation. We proceed by inspection of every ❀-rule.

poll↓, poll↑ By Lemma 10.5.7.

epis

1. Suppose that $w \Vdash \Phi\,[e]$. Then by Lemma 10.5.1 we have $w' \Vdash \Phi\,[e]$, and thus we can conclude for instance with $w' \Vdash \Phi\,[e\,|_{\varnothing}\, e'\,|_{\varnothing}\, e]$.
2. Suppose that $w \Vdash\ \supset \Phi\,[e]$. Then since we trivially have $w \geq w$ and $w \Vdash \varnothing\,[e\,|_{\varnothing}\, e]$, we get that $w \Vdash \Phi\,[e\,|_{\varnothing}\, e\,|_{\varnothing}\, e'']$ for some w-evaluation e'', and thus $w \Vdash \Phi\,[e]$.

epet Let $\gamma = \mathbf{x} \cdot \Phi$. We trivially have that $w' \Vdash \varnothing\,[e\,|_{\mathbf{x}}\, e'\,|_{\varnothing}\, e]$, and thus can conclude.

ipis We trivially have $\mathbf{x}, \mathbf{y} \cdot \Phi \supset \Delta \vDash \mathbf{x}, \mathbf{y} \cdot \Phi \supset \Delta$ by Fact 10.5.5, and thus we can conclude by Lemma 10.5.8.

ipet The first direction is trivial by Lemma 10.5.5. In the other direction, let $\gamma = \mathbf{z} \cdot \Xi$, and suppose that $w \Vdash \gamma \supset \mathbf{x} \cdot \sigma(\Phi)\,;\, \mathbf{x}, \mathbf{y} \cdot \Phi\,;\, \Delta\,[e]$ and $w' \Vdash \Xi\,[e\,|_{\mathbf{z}}\, e']$. Then there must be some w'-evaluation e'' such that either:

- $w' \Vdash \Xi'\,[e|_{\mathbf{z}}e'|_{\mathbf{z}'}e'']$ for some $\mathbf{z}' \cdot \Xi' \in \Delta$, and we conclude immediately;
- $w' \Vdash \Phi\,[e\,|_{\mathbf{z}}\, e'\,|_{\mathbf{x} \cup \mathbf{y}}\, e'']$, and we also conclude immediately;
- or $w' \Vdash \sigma(\Phi)\,[e\,|_{\mathbf{z}}\, e'\,|_{\mathbf{x}}\, e'']$, and we conclude with the same argument as in the proof of Lemma 10.5.9.

srep Let $\gamma_i = \mathbf{y}_i \cdot \Psi_i$ for $1 \leq i \leq n$.

1. Suppose that $w \Vdash \mathbf{x} \cdot \Phi, (\supset \{\gamma_i\}_i^n) \supset \Delta\,[e]$ and $w' \Vdash \Phi\,[e\,|_{\mathbf{x}}\, e']$. We show that $w' \Vdash \gamma_i \supset \Delta\,[e\,|_{\mathbf{x}}\, e']$ for all $1 \leq i \leq n$, i.e. for every $w'' \geq w'$ and w''-evaluation e'', $w'' \Vdash \Psi_i\,[e\,|_{\mathbf{x}}\, e'\,|_{\mathbf{y}_i}\, e'']$ implies that there is some $\mathbf{z} \cdot \Xi \in \Delta$ and w''-evaluation e''' such that $w'' \Vdash \Xi\,[e\,|_{\mathbf{x}}\, e'\,|_{\mathbf{y}_i}\, e''\,|_{\mathbf{z}}\, e''']$. By assumption, Lemma 10.5.1 and the fact that $\mathsf{fv}(\Phi) \cap \mathbf{y}_i = \varnothing$, we have $w'' \Vdash \Phi\,[e\,|_{x}\, e'\,|_{\mathbf{y}_i}\, e'']$. Also since $w'' \Vdash \Psi_i\,[e\,|_{\mathbf{x}}\, e'\,|_{\mathbf{y}_i}\, e'']$ we immediately get $w'' \Vdash\supset \{\gamma_i\}_i^n\,[e|_{\mathbf{x}}e']$, and thus $w'' \Vdash\supset \{\gamma_i\}_i^n\,[e|_{\mathbf{x}}e'|_{\mathbf{y}_i}e'']$ since $\mathsf{fv}(\supset \{\gamma_i\}_i^n) \cap \mathbf{y}_i = \varnothing$. Thus by Fact 10.5.1 we have $w'' \Vdash \Phi, (\supset \{\gamma_i\}_i^n)\,[e\,|_{\mathbf{x} \cup \mathbf{y}_i}\, (e'\,|_{\mathbf{y}_i}\, e'')]$, and by hypothesis (and the fact that $w'' \geq w$ by transitivity) we obtain that $w'' \Vdash \Xi\,[e|_{\mathbf{x} \cup \mathbf{y}_i}(e'\,|_{\mathbf{y}_i} e'')|_{\mathbf{z}}e''']$ for some $\mathbf{z} \cdot \Xi \in \Delta$ and w''-evaluation e'''. Then we conclude again by Fact 10.5.1.
2. Suppose that $w \Vdash \mathbf{x} \cdot \Phi \supset \{\gamma_i \supset \Delta\}_i^n\,[e]$ and $w' \Vdash \Phi, (\supset \{\gamma_i\}_i^n)\,[e\,|_{\mathbf{x}}\, e']$. Then there must be some $1 \leq i \leq n$ and w'-evaluation e'' such that $w' \Vdash \Psi_i\,[e\,|_{\mathbf{x}}\, e'\,|_{\mathbf{y}_i}\, e'']$, and for all $1 \leq j \leq n$ we know that $w' \Vdash \gamma_j \supset \Delta\,[e\,|_{\mathbf{x}}\, e']$. Thus since $w' \leq w'$ by reflexivity, there must be some $\mathbf{z} \cdot \Xi \in \Delta$ and w'-evaluation e''' such that $w' \Vdash \Xi\,[e|_{\mathbf{x}}e'|_{\mathbf{y}_i}e''|_{\mathbf{z}}e''']$, and we can conclude with $w' \Vdash \Xi\,[e|_{\mathbf{x}}e'\,|_{\mathbf{y}_i \cup \mathbf{z}}\,(e''\,|_{\mathbf{z}}\, e''')]$ by Fact 10.5.1.

□

Then the soundness of the contextual closure of natural rules follows immediately from functoriality:

Lemma 10.5.11 (Natural soundness) *If* $\Phi \to_{❀} \Psi$ *then* $\Phi \dashv\vDash \Psi$.

Proof. By Lemma 10.5.10 and Lemma 10.5.3. □

The soundness of cultural rules is straightforward with the previous lemmas:

Lemma 10.5.12 (Cultural soundness) *If* $\Phi \rightarrow_{✂} \Psi$ *then* $\Psi \vDash \Phi$.

Proof. By inspection of every ✂-rule.

grow, crop By Lemma 10.5.4 and Lemma 10.5.3.

pull, glue By Lemma 10.5.5 and Lemma 10.5.3.

apis By Lemma 10.5.8 and Lemma 10.5.3.

apet By Lemma 10.5.9 and Lemma 10.5.3.

□

Then it follows that every derivation in the flower calculus is sound:

Theorem 10.5.13 (Soundness) *If* $\Phi \rightarrow^{*} \Psi$ *then* $\Psi \vDash \Phi$.

Proof. By Lemma 10.5.11, Lemma 10.5.12 and Fact 10.5.5, Fact 10.5.6. □

In particular $\vdash \phi$ implies $\vDash \phi$, i.e. every provable flower is true.

10.6. Completeness

In this section, we give a direct completeness proof for the natural fragment ❀ of the flower calculus: every true flower ϕ is naturally provable, i.e. $\vDash \phi$ implies $\vdash_{❀} \phi$. Since this fragment is analytic, we cannot adapt directly most of the completeness proofs for standard proof systems that can be found in the literature. Indeed, most of them exploit the transitivity of syntactic entailment $\vdash$, and more precisely the fact that it is easily shown syntactically with the help of a non-analytic principle for composing proofs: in Hilbert systems it is the rule of *modus ponens* mp, in sequent calculi the cut rule, and in natural deduction the *substitution theorem*.

Fortunately as mentioned in Section 10.4, a few people have noticed that with Kripke semantics, it is not too difficult to find completeness proofs that do not rely on the assumption of transitivity for $\vdash$, thus allowing for a *semantic* proof of cut-elimination. Here we propose an adaptation of this technique to our flower calculus, based on a completeness proof for cut-free sequent calculus given by Hermant in [120], which is itself close to the original completeness proof of Gödel with respect to classical Tarski models. Quite remarkably, the overall structure of the argument

[120]: Hermant (2005), 'Semantic Cut Elimination in the Intuitionistic Sequent Calculus'

is the same, even though both the forcing relation on flowers and the rules of the flower calculus differ significantly from the forcing relation on formulas and sequent calculus rules. A novelty of our proof is that it dispenses completely with the need for *Henkin witnesses*, thus avoiding some technicalities involving among others the manipulation of an infinite hierarchy of first-order languages.

10.6.1. Theories

First we need to generalize our notions of syntactic and semantic entailment to possibly *infinite* sets of flowers, so-called *theories*:

Definition 10.6.1 (Theory) *Any set $\mathcal{T} \subseteq \mathbb{F}$ of flowers is called a theory. In particular, a bouquet can be regarded as a finite theory, by forgetting the number of repetitions of its elements. We say that a bouquet Φ is* provable from *a theory $\mathcal{T}$, written $\mathcal{T} \vdash \Phi$, if there exists a bouquet $\Psi \subseteq \mathcal{T}$ such that $\Psi \vdash \Phi$. Given a Kripke structure $\mathcal{K}$, a world w in $\mathcal{K}$ and a w-evaluation e, we say that $\mathcal{T}$ is* forced *by w under e, written $w \Vdash \mathcal{T}\,[e]$, if $w \Vdash \phi\,[e]$ for all $\phi \in \mathcal{T}$. Then Φ is a* consequence *of $\mathcal{T}$, written $\mathcal{T} \vDash_{\mathcal{K}} \Phi$, if $w \Vdash \mathcal{T}\,[e]$ implies $w \Vdash \Phi\,[e]$ for every world w in $\mathcal{K}$ and w-evaluation e.*

Lemma 10.6.1 (Weakening) *If $\mathcal{T} \subseteq \mathcal{T}'$ and $\mathcal{T} \vdash \phi$, then $\mathcal{T}' \vdash \phi$.*

Proof. This follows immediately from our definition of provability from a theory (Definition 10.6.1). ☐

The following notions our crucial to define the *completion* procedure at the heart of any Gödel-style completeness proof:

Definition 10.6.2 (ψ-consistency) *A theory $\mathcal{T}$ is ψ-consistent when $\mathcal{T} \nvdash_{❀} \psi$.*

Definition 10.6.3 (ψ-completeness) *A theory $\mathcal{T}$ is ψ-complete when for all $\phi \in \mathbb{F}$, either $\mathcal{T}, \phi \vdash_{❀} \psi$ or $\phi \in \mathcal{T}$.*

Intuitively, a theory $\mathcal{T}$ is ψ-consistent when one cannot deduce ψ from it, and ψ-complete when it *decides* any formula ϕ when ψ is assumed. This is better understood by considering the special case where $\psi = \,ᗡ$ is the *absurd* flower: then ψ-consistency means that one cannot derive any contradiction from $\mathcal{T}$; and ψ-completeness that $\mathcal{T}$ either *refutes* ϕ syntactically with a proof of $\Phi, \phi\ ᗡ\ (ᗡ)$ for some $\Phi \subseteq \mathcal{T}$, or already *validates* it "semantically", i.e. without the need for a proof since $\phi \in \mathcal{T}$.

10.6.2. Completion

In the following, we suppose some enumeration $(\phi_n)_{n\in\mathbb{N}}$ of $\mathbb{F}$, which should be definable constructively given the inductive nature of flowers.

Let $\psi \in \mathbb{F}$, and $\mathcal{T}$ a ψ-consistent theory. We now define the completion procedure, which constructs an extension $\mathsf{Com}(\mathcal{T}) \supseteq \mathcal{T}$ with the property that $\mathsf{Com}(\mathcal{T})$ is ψ-consistent and ψ-complete.

Definition 10.6.4 (n-completion) *The* n-completion $\mathsf{Com}^n(\mathcal{T})$ *of* $\mathcal{T}$ *is defined recursively as follows:*

$$\mathsf{Com}^0(\mathcal{T}) = \mathcal{T}$$

$$\mathsf{Com}^{n+1}(\mathcal{T}) = \begin{cases} \mathsf{Com}^n(\mathcal{T}) \cup \phi_n & \text{if } \mathsf{Com}^n(\mathcal{T}) \cup \phi_n \text{ is } \psi\text{-consistent} \\ \mathsf{Com}^n(\mathcal{T}) & \text{otherwise} \end{cases}$$

Definition 10.6.5 (Completion) *The* completion $\mathsf{Com}(\mathcal{T})$ *of* $\mathcal{T}$ *is the denumerable union of all* n*-completions:*

$$\mathsf{Com}(\mathcal{T}) = \bigcup_{n\in\mathbb{N}} \mathsf{Com}^n(\mathcal{T})$$

Lemma 10.6.2 $\mathsf{Com}(\mathcal{T})$ *is* ψ*-consistent and* ψ*-complete.*

Proof. For ψ-consistency, it is immediate by induction on n that $\mathsf{Com}^n(\mathcal{T})$ is ψ-consistent. Then suppose that $\mathsf{Com}(\mathcal{T}) \vdash_{❀} \psi$, that is there is some bouquet $\Phi \subseteq \mathsf{Com}(\mathcal{T})$ such that $\Phi \vdash_{❀} \psi$. For each $\phi \in \Phi$, there is some rank n such that $\phi \in \mathsf{Com}^n(\mathcal{T})$. Let m be the greatest such rank. Then $\Phi \subseteq \mathsf{Com}^m(\mathcal{T})$, and thus by weakening (Lemma 10.6.1) $\Phi \nvdash_{❀} \psi$. Contradiction.

For ψ-completeness, suppose that there is some ϕ such that $\mathsf{Com}(\mathcal{T}), \phi \nvdash_{❀} \psi$ and $\phi \notin \mathsf{Com}(\mathcal{T})$, and let $\phi = \phi_n$. Then $\mathsf{Com}(\mathcal{T}) \cup \phi_n$ is ψ-consistent, and thus by weakening (Lemma 10.6.1) so is $\mathsf{Com}^n(\mathcal{T}) \cup \phi_n$. This entails that $\phi_n \in \mathsf{Com}^{n+1}(\mathcal{T}) \subseteq \mathsf{Com}(\mathcal{T})$. Contradiction. ☐

10.6.3. Adequacy

The following two propositions constitute the central argument that allows the completeness proof to go through despite the analyticity of ❀-rules. They are a direct adaptation of [120, Proposition 7], which Hermant identifies as "an important property of any A-consistent, A-complete theory, [...] that it enjoys some form of the subformula property".

Roughly, the first proposition captures the (intuitionistic) truth-conditions that make a flower *valid* (i.e. true in every model) by modelling them on material implication, just like Peirce would do with his scroll (see Section 10.1): ϕ is true if the content Φ_i of one of its petals (consequents) is, or if the content Φ of its pistil (antecedant) is not.

Proposition 10.6.3 (Analytic truth) *Let $\psi \in \mathbb{F}$, $\mathcal{T}$ some ψ-consistent and ψ-complete theory, and $\phi = \mathbf{x} \cdot \Phi \mathbin{\mathsf{D}} \Delta$ with $\Delta = \{\delta_i\}_i^n = \{\mathbf{x}_i \cdot \Phi_i\}_i^n$ such that $\phi \in \mathcal{T}$. Then for every substitution $\sigma : \mathbf{x} \to \mathbb{T}$, either $\sigma(\Phi_i) \subseteq \mathcal{T}$ for some $1 \leq i \leq n$, or $\mathcal{T} \nvdash_{❀} \sigma(\Phi)$.*

Proof. Suppose the contrary, i.e. there is a substitution σ such that $\mathcal{T} \vdash_{❀} \sigma(\Phi)$ and for all $1 \leq i \leq n$, there is some $\phi_i \in \Phi_i$ ① such that $\sigma(\phi_i) \notin \mathcal{T}$. Thus by ψ-completeness of $\mathcal{T}$, we get $\mathcal{T}, \sigma(\phi_i) \vdash_{❀} \psi$. So there are $\Psi \subseteq \mathcal{T}$ and $\Psi_i \subseteq \mathcal{T} \cup \sigma(\phi_i)$ such that $\Psi \vdash_{❀} \sigma(\Phi)$ ② and $\Psi_i \vdash_{❀} \psi$ ③. Now it cannot be the case that $\Psi_i \subseteq \mathcal{T}$, otherwise by weakening and ψ-consistency of $\mathcal{T}$ we would have $\Psi_i \nvdash_{❀} \psi$. So there must exist $\Psi_i' \subseteq \mathcal{T}$ such that $\Psi_i = \Psi_i' \cup \sigma(\phi_i)$ ④. Again by weakening and ψ-consistency of $\mathcal{T}$, we get $\Psi, \bigcup_{i=1}^n \Psi_i', \phi \nvdash_{❀} \psi$. Now we derive a contradiction by showing $\Psi, \bigcup_{i=1}^n \Psi_i', \phi \vdash_{❀} \psi$. Let $\Xi\square$ be a context such that $\Psi, \bigcup_{i=1}^n \Psi_i', \phi \succ \Xi$ ⑤. Then $\Xi\boxed{\psi} \to_{❀}^{*} \Xi\boxed{}$ with the following derivation:

$$
\begin{array}{lll}
\Xi\boxed{\psi} & \to_{\mathsf{epis}} & \Xi\boxed{\cdot \mathbin{\mathsf{D}} \cdot \psi} \\
 & \to_{\mathsf{poll}\uparrow} & \Xi\boxed{\cdot \phi \mathbin{\mathsf{D}} \cdot \psi} \qquad (⑤) \\
 & \to_{\mathsf{ipis}} & \Xi\boxed{\cdot (\cdot \sigma(\Phi) \mathbin{\mathsf{D}} \sigma(\Delta)), \phi \mathbin{\mathsf{D}} \cdot \psi} \\
 & \to_{\mathsf{poll}\downarrow} & \Xi\boxed{\cdot (\cdot \mathbin{\mathsf{D}} \sigma(\Delta)), \phi \mathbin{\mathsf{D}} \cdot \psi} \qquad (②, ⑤) \\
 & \to_{\mathsf{srep}} & \Xi\boxed{\cdot \phi \mathbin{\mathsf{D}} \cdot \{\sigma(\delta_i) \mathbin{\mathsf{D}} \cdot \psi\}_i^n} \\
 & = & \Xi\boxed{\cdot \phi \mathbin{\mathsf{D}} \cdot \{\mathbf{x}_i \cdot \sigma(\Phi_i) \mathbin{\mathsf{D}} \cdot \psi\}_i^n} \\
 & \to_{\mathsf{poll}\downarrow}^{n} & \Xi\boxed{\cdot \phi \mathbin{\mathsf{D}} \cdot \{\mathbf{x}_i \cdot \sigma(\Phi_i) \mathbin{\mathsf{D}} \cdot \}_i^n} \qquad (①, ③, ④, ⑤) \\
 & \to_{\mathsf{epet}}^{n} & \Xi\boxed{\cdot \phi \mathbin{\mathsf{D}} \cdot} \\
 & \to_{\mathsf{epet}} & \Xi\boxed{}
\end{array}
$$

□

Dually, the second proposition captures the grounds on which a flower can be deemed *invalid* (i.e. false in at least one model): ϕ is not true if assuming that its pistil Φ is true is not sufficient to conclude that one of its petals Φ_i is.

Proposition 10.6.4 (Analytic refutation) *Let $\psi \in \mathbb{F}$, $\mathcal{T}$ some ψ-consistent and ψ-complete theory, and $\phi = \mathbf{x} \cdot \Phi \mathbin{\mathsf{D}} \Delta$ with $\Delta = \{\delta_i\}_i^n = \{\mathbf{x}_i \cdot \Phi_i\}_i^n$ such that $\mathcal{T} \nvdash_{❀} \phi$. Then for every $1 \leq i \leq n$ and substitution $\sigma : \mathbf{x}_i \to \mathbb{T}$, there is some $\phi_i \in \Phi_i$ such that $\mathcal{T}, \Phi \nvdash_{❀} \sigma(\phi_i)$.*

Proof. Suppose the contrary, i.e. there are some $1 \leq i \leq n$ and $\sigma : \mathbf{x}_i \to \mathbb{T}$ such that $\mathcal{T}, \Phi \vdash_{❀} \sigma(\Phi_i)$. Therefore there must exist $\Psi \subseteq \mathcal{T}$ and $\Phi_0 \subseteq \Phi$ ① such that $\Psi, \Phi_0 \vdash_{❀} \sigma(\Phi_i)$ ②. By hypothesis, for every $\Phi' \subseteq \mathcal{T}$ there is a context Ξ such that $\Phi' \succ \Xi$ and $\Xi\boxed{\phi} \nrightarrow_{❀}^{*} \Xi\boxed{}$. We now derive a contradiction by showing $\Xi\boxed{\phi} \xrightarrow{}^{*}_{❀} \Xi\boxed{}$ for all Ξ such that $\Psi \succ \Xi$ ③:

$$\begin{aligned}
\Xi\boxed{\phi} &\rightarrow_{\mathrm{ipet}} && \Xi\boxed{\mathbf{x} \cdot \Phi \mathbin{\mathsf{D}} \ \cdot \sigma(\Phi_i)\,;\,\Delta} \\
&\rightarrow_{\mathrm{poll}\downarrow} && \Xi\boxed{\mathbf{x} \cdot \Phi \mathbin{\mathsf{D}} \ \cdot\,;\,\Delta} && (①, ②, ③) \\
&\rightarrow_{\mathrm{epet}} && \Xi\boxed{}
\end{aligned}$$

□

Digression

Note that both Proposition 10.6.3 and Proposition 10.6.4 are proved indirectly by contradiction, but that the contradiction is obtained by exhibiting a concrete derivation, exploiting respectively the universal instantiation of pistils with the ipis rule, and the existential instantiation of petals with the ipet rule. Thus there seems to be some constructive content to these proofs, despite their use of a classical reasoning principle. Also, this is the only place in the completeness proof where we build derivations, and all ❀-rules (and only them) seem to be required to conclude.

Lastly, we define the so-called *universal Kripke structure* ✿(ψ) relative to a flower ψ:

Definition 10.6.6 (Universal Kripke structure) *Let $\psi \in \mathbb{F}$. The universal Kripke structure ✿(ψ) has:*

- *The set of ψ-consistent and ψ-complete theories as its worlds;*
- *Set inclusion as its accessibility relation;*
- *For each world $\mathcal{T}$, a first-order structure whose domain is the set of terms $\mathbb{T}$, and whose interpretation map is given by:*
  - $[\![f]\!](\vec{\mathbf{t}}) = f(\vec{\mathbf{t}})$
  - $[\![p]\!] = \left\{\vec{\mathbf{t}} \;\middle|\; p(\vec{\mathbf{t}}) \in \mathcal{T}\right\}$

One can easily check that the monotonicity conditions of Kripke structures hold for ✿(ψ).

We are now equipped to prove the main *adequacy* lemma, which relates forcing in ✿(ψ) with ψ-consistency and ψ-completeness:

Lemma 10.6.5 (Adequacy) *Let $\phi, \psi \in \mathbb{F}$, $\mathcal{T}$ a ψ-consistent and ψ-complete theory, and σ a subtitution. Then*

1. *$\sigma(\phi) \in \mathcal{T}$ implies $\mathcal{T} \Vdash \phi\,[\sigma]$, and*
2. *$\mathcal{T} \nvdash_{❀} \sigma(\phi)$ implies $\mathcal{T} \nVdash \phi\,[\sigma]$*

Proof. By induction on $|\phi|$.

- Suppose $\phi = p(\vec{\mathbf{t}})$.
  1. By definition of forcing (Definition 10.4.4) and ✿(ψ) (Definition 10.6.6), $\mathcal{T} \Vdash p(\vec{\mathbf{t}})\,[\sigma]$ precisely when $\sigma(p(\vec{\mathbf{t}})) \in \mathcal{T}$.
  2. Suppose that $\mathcal{T} \Vdash \phi[\sigma]$, that is $\sigma(\phi) \in \mathcal{T}$. Then by weakening (Lemma 10.6.1), we get $\sigma(\phi) \nvdash_{❀} \sigma(\phi)$. But this is impossible by reflexivity of $\vdash$ (Lemma 10.3.3).
- Suppose $\phi = \mathbf{x} \cdot \Phi \mathbin{\mathsf{D}} \{\mathbf{x}_i \cdot \Phi_i\}_i^n$.
  1. Let $\mathcal{U} \supseteq \mathcal{T}$ be a ψ-consistent and ψ-complete theory. Obviously $\sigma(\phi) = \mathbf{x} \cdot \sigma|_{\mathbf{x}}(\Phi) \mathbin{\mathsf{D}} \{\mathbf{x}_i \cdot \sigma|_{\mathbf{x} \cup \mathbf{x}_i}(\Phi_i)\}_i^n \in \mathcal{U}$, and thus by Proposition 10.6.3, for every substitution τ, either $|_{\mathbf{x}}\tau \circ \sigma|_{\mathbf{x} \cup \mathbf{x}_i}(\Phi_i) = \sigma|_{\mathbf{x}}\tau|_{\mathbf{x}_i}(\Phi_i) \subseteq \mathcal{U}$ for some $1 \leq i \leq n$, or $\mathcal{U} \nvdash_{❀} |_{\mathbf{x}}\tau \circ \sigma|_{\mathbf{x}}(\Phi) = \sigma|_{\mathbf{x}}\tau(\Phi)$. In the first case,

we get $\mathcal{U} \Vdash \Phi_i\,[\sigma|_{\mathbf{x}}\,\tau|_{\mathbf{x}_i}]$ by IH. In the second case, we get $\mathcal{U} \nVdash \Phi\,[\sigma|_{\mathbf{x}}\tau]$ by IH. In other words, $\mathcal{U} \Vdash \Phi\,[\sigma|_{\mathbf{x}}\tau]$ implies $\mathcal{U} \Vdash \Phi_i\,[\sigma\,|_{\mathbf{x}}\,\tau\,|_{\mathbf{x}_i}]$, that is $\mathcal{T} \Vdash \phi\,[\sigma]$.

2. By Proposition 10.6.4, for every $1 \leq i \leq n$ and substitution τ, there is some $\phi_i \in \Phi_i$ such that $\mathcal{T}, \sigma|_{\mathbf{x}}(\Phi) \nvdash_{❀}\ |_{\mathbf{x}_i}\,\tau \circ \sigma|_{\mathbf{x} \cup \mathbf{x}_i}(\phi_i) = \sigma\,|_{\mathbf{x}}\ |_{\mathbf{x}_i}\,\tau(\phi_i)$. By the completion procedure, we get a theory $\mathcal{U} = \mathrm{Com}(\mathcal{T} \cup \sigma|_{\mathbf{x}}(\Phi)) \supseteq \mathcal{T} \cup \sigma|_{\mathbf{x}}(\Phi)$ which is both $\sigma\,|_{\mathbf{x}}\ |_{\mathbf{x}_i}\,\tau(\phi_i)$-consistent and $\sigma\,|_{\mathbf{x}}\ |_{\mathbf{x}_i}\,\tau(\phi_i)$-complete. Then by IH, $\mathcal{U} \Vdash \Phi\,[\sigma\,|_{\mathbf{x}}\,]$ since $\sigma\,|_{\mathbf{x}}\,(\Phi) \subseteq \mathcal{U}$, and $\mathcal{U} \nVdash \phi_i\,[\sigma\,|_{\mathbf{x}}\ |_{\mathbf{x}_i}\,\tau]$ since $\mathcal{U}$ is $\sigma\,|_{\mathbf{x}}\ |_{\mathbf{x}_i}\,\tau(\phi_i)$-consistent, that is $\mathcal{T} \nVdash \phi\,[\sigma]$.

□

As a near-direct consequence, we get:

Theorem 10.6.6 (Completeness) *$\Phi \vDash \Psi$ implies $\Phi \vdash_{❀} \Psi$.*

Proof. Let $\mathcal{T}$ be a ψ-consistent theory. We prove that $\mathcal{T} \nvDash \psi$ by showing in particular that $\mathcal{T} \nvDash_{✿(\psi)} \psi$, and more specifically that $\mathrm{Com}(\mathcal{T}) \Vdash \mathcal{T}\,[1]$ but $\mathrm{Com}(\mathcal{T}) \nVdash \psi\,[1]$. Then it follows by (classical) contraposition that $\mathcal{T} \vDash \psi$ implies $\mathcal{T} \vdash_{❀} \psi$ for any ψ and any $\mathcal{T}$, and thus we can conclude.

- Let $\phi \in \mathcal{T}$. Then $1(\phi) = \phi \in \mathrm{Com}(\mathcal{T})$, thus by ψ-consistency and ψ-completeness of the completion (Lemma 10.6.2), we can apply adequacy (Lemma 10.6.5) to get $\mathrm{Com}(\mathcal{T}) \Vdash \phi\,[1]$.
- Similarly, we can apply adequacy to get $\mathrm{Com}(\mathcal{T}) \nVdash \psi\,[1]$.

□

Combined with strong deduction (Theorem 10.3.2), this also yields a strong completeness theorem for the full flower calculus[13]:

13: Actually it already works for the fragment ❀ ∪ {grow}, thanks to the proof of Theorem 10.3.2.

Corollary 10.6.7 (Strong completeness) *$\Phi \vDash \Psi$ implies $\Psi \rightarrow^* \Phi$.*

Finally, the composition of the soundness, completeness and deduction theorems (Theorem 10.5.13, Theorem 10.6.6 and Theorem 10.3.4) gives the admissibility of ✂-rules, and thus the analyticity of the flower calculus:

Corollary 10.6.8 (Analyticity) *If $\Phi \vdash \Psi$ then $\Phi \vdash_{❀} \Psi$.*

10.7. Automated proof search

10.7.1. Preliminary definitions

Analyticity and invertibility The completeness of ❀-rules (Theorem 10.6.6) suggests that some efficient proof search procedure might be devised for the flower calculus, and even a complete one for the propositional fragment where every sprinkler is empty[14]. Indeed in sequent calculi, analyticity (the subformula property) greatly reduces the search space when looking for a proof of a given sequent, and one might expect the same to apply in the natural fragment ❀ of the flower calculus. Also, the invertibility of ❀-rules implies that the procedure will not need to perform *backtracking*, a great source of non-determinism in proof search[15], and particularly so in intuitionistic logic[16].

14: It is well-known that provability in intuitionistic propositional logic is decidable, a result first demonstrated by Gentzen [88].

15: See also Section 5.1 for a discussion on this topic.

16: The only fully invertible proof system for intuitionistic (predicate) logic that we know of is the labelled sequent calculus G3IntQ from Lyon's thesis [156] (we thank Lutz Straßburger for pointing that to us). However, a peculiarity of labelled sequent calculi is that they need to incorporate *semantical* notions like the worlds and accessibility relation of Kripke structures into the syntax of sequents, in order to make every rule invertible [97]. We do not need such devices in the flower calculus, which will be crucial from a user interface standpoint in Section 10.8.

Depth and interaction However, there is another great source of non-determinism that does not arise in Gentzen systems, but is inherent to any deep inference formalism: it is precisely the fact that rules can be applied in contexts of *arbitrary* depth, thus inducing a number of choices that is exponential in the latter. Kahramanogullari has proposed some attempts to tame this problem in [137, 139], and our approach bears some similarities to his. In particular, we propose in the following a (sketch of a) proof search algorithm for the propositional flower calculus, whose core rests on an *interaction* relation between contexts which is a generalization of the pollination relation (Definition 10.3.5), and is close to the *structural* relation of Kahramanogullari [137, Definition 2.13].

[137]: Kahramanogullari (2014), 'Interaction and Depth against Nondeterminism in Proof Search'
[139]: Kahramanoğulları (2006), 'Reducing Nondeterminism in the Calculus of Structures'

Definition 10.7.1 (Interaction) *Let $\Phi\square, \Psi\square$ be two contexts. We say that $\Phi\square$ can* interact *with $\Psi\square$ when there exist contexts $\Xi\square, \Phi_0, \Psi_0$ such that either:*

(Cross-interaction)

$$\Phi\square = \Xi\boxed{\Phi_0, \Psi_0\boxed{\varnothing}}$$
$$\text{and} \quad \Psi\square = \Xi\boxed{\Phi_0\boxed{\varnothing}, \Psi_0}$$

In this case we write $\Phi\square \bowtie \Psi\square$;

(Self-interaction) *there is some Δ such that*

$$\Phi\square = \Xi\boxed{\Phi_0 \mathbin{\mathsf{D}} \Psi_0\boxed{\varnothing} ; \Delta}$$
$$\text{and} \quad \Psi\square = \Xi\boxed{\Phi_0\boxed{\varnothing} \mathbin{\mathsf{D}} \Psi_0 ; \Delta}$$

In this case we write $\Phi\square \mathrel{⧜} \Psi\square$.

It is clear that $\bowtie$ is a symmetric relation, while $\mathrel{⧜}$ is not. Also, we write $\overset{+}{\bowtie}$ and $\overset{+}{⧜}$ (resp. $\overset{-}{\bowtie}$ and $\overset{-}{⧜}$) to further specify when $\Xi\square$ is positive (resp. negative).

Causality Our algorithm will be driven by the search for dual *occurrences*[17] of atoms, a recurring idea in this thesis.

17: What we call an occurrence here corresponds to our notion of path in Definition 3.1.2.

Definition 10.7.2 (Occurrence) *An occurrence is a pair $(\Phi\square, \phi)$ of a flower ϕ and a context $\Phi\square$. It is said to be* atomic *if ϕ is an atom.*

Then, two atomic occurrences can interact when their contexts can. In the (linear) classical setting of [137], there is no preferred direction in the information flow between two interacting atoms, which can be seen as the essence of the (linear) law of excluded middle $A \parr A^{\perp}$. However in our intuitionistic setting, implication forces a direction from premisses to conclusions, which is reflected in our distinction between the symmetric cross-interaction relation $\bowtie$, and the asymmetric self-interaction relation $\looparrowleft$ [18]. In turn, the asymmetry of $\looparrowleft$ will impose a causality on atoms, where one atom can potentially be *justified* by the other:

18: Arguably, the linear heart of this phenomenon can be studied in the logic BV, which adds to the commutative connective $\parr$ a non-commutative or *sequential* operator $\triangleright$. Interestingly, BV is the logic that motivated the whole development of the deep inference methodology by Guglielmi [110], because it could not be accurately expressed in shallow Gentzen formalisms.

Definition 10.7.3 (Justification) *A justification is an oriented pair of atomic occurrences $\mathfrak{a} = (\Phi\square, a) \hookrightarrow (\Psi\square, b)$.*

One can then naturally aggregate justifications into *arguments*:

Definition 10.7.4 (Argument) *An argument is a finite set $\mathfrak{A}$ of justifications. Every argument $\mathfrak{A}$ can be seen as the set of edges of a directed graph, whose vertices are the occurrences appearing in the justifications of $\mathfrak{A}$. Conversely, every directed graph on a set of occurrences can be seen as an argument, whose justifications are the edges of the graph.*

We identify two essential arguments inherent to any bouquet:

Definition 10.7.5 (Vehicle and Anchor) *Let Φ a bouquet, and $\mathcal{A}(\Phi) = \{(\Psi\square, a) \mid \Phi = \Psi\boxed{a}\}$ its set of atomic occurrences.*

The vehicle of Φ is the argument $\mathcal{V}(\Phi) \subseteq \mathcal{A}(\Phi) \times \mathcal{A}(\Phi)$ such that $(\Psi\square, a) \hookrightarrow (\Psi'\square, a') \in \mathcal{V}(\Phi)$ iff the following conditions hold:

(Identity) *$a = a'$;*

(Polarity) *$\Psi\square$ is negative and $\Psi'\square$ is positive;*

(Interaction) *either $\Psi\square \stackrel{-}{\bowtie} \Psi'\square$, $\Psi\square \stackrel{+}{\looparrowleft} \Psi'\square$ and $\Psi\square = \square$, or $\Psi'\square \stackrel{+}{\looparrowleft} \Psi\square$.*

Dually, the anchor of Φ is the argument $\Lambda(\Phi) \subseteq \mathcal{A}(\Phi) \times \mathcal{A}(\Phi)$ such that $(\Psi\square, a) \hookrightarrow (\Psi'\square, a') \in \Lambda(\Phi)$ iff the following conditions hold:

(Polarity) *$\Psi\square$ is positive and $\Psi'\square$ is negative;*

(Interaction) *$\Psi\square \stackrel{-}{\looparrowleft} \Psi'\square$.*

Intuitively, the vehicle of a bouquet Φ is the set of possible justifications that *can* be performed by the prover/player, whose goal is to *justify* every positive atom; while the anchor is the set of justifications that *must* be provided on demand by the environment/opponent, whose task is to *be able to justify* every negative atom. The reason that we identify this structure, is that we want crucially to avoid *cycles* in our justification attempts during proof search. These may arise in the dialogical interaction between the prover and its environment of hypotheses, when justifications of the former are composed with those of the latter. Formally, this interaction is expressed by the *union* of arguments:

Definition 10.7.6 (Compatibility) *Given a justification $\mathfrak{a}$ and an argument $\mathfrak{A}$, we say that $\mathfrak{a}$ is compatible with $\mathfrak{A}$, written $\mathfrak{a} \downarrow \mathfrak{A}$, when there is no cycle in $\mathfrak{A} \cup \{\mathfrak{a}\}$.*

10.7.2. The algorithm

Pollination The vehicle and anchor of any bouquet Φ can easily by computed, by noticing that the context $\Xi\square$ in Definition 10.7.1 is simply the *least common ancestor* of the atoms $\Psi\boxed{a}$ and $\Psi'\boxed{a'}$ in the tree representation of Φ, which we write $\mathsf{lca}(\Psi\square, \Psi'\square)$. This forms the basis for the first *phase* of our algorithm, that we call the *pollination phase*[19]. Roughly, the idea is that given a goal Φ, we want to perform eagerly every justification $\mathfrak{a}$ in the vehicle $\mathcal{V}(\Phi)$, on the condition that $\mathfrak{a}$ is compatible with the anchor $\mathcal{A}(\Phi)$ and all justifications $\mathfrak{H}$ of the previous phases. It turns out that every such justification can be implemented by an instance of the $\mathsf{poll}{\downarrow}$ rule, sometimes preceded by an instance of the $\mathsf{poll}{\uparrow}$ rule. This is best described by the following *imperative* procedure, that takes as input and modifies in-place both the goal Φ, and the history $\mathfrak{H}$ of justifications performed so far:

19: It corresponds roughly to a combination of the *application* and *linking* phases of the proof search algorithm for $\mathsf{B}_{\mathsf{inv}}$ described in Subsection 8.8.2.

Procedure pollination($\Phi, \mathfrak{H}$)

---

foreach $\mathfrak{a} = (\Psi\square, a) \hookrightarrow (\Psi'\square, a') \in \mathcal{V}(\Phi) \setminus \mathfrak{H}$ **do**
  if $\mathfrak{a} \downarrow \mathcal{A}(\Phi) \cup \mathfrak{H}$ **then**
    $\phi\square \longleftarrow$ the direct child flower context of $\mathsf{lca}(\Psi\square, \Psi'\square)$ where a occurs
    $\phi'\square \longleftarrow$ the direct child flower context of $\mathsf{lca}(\Psi\square, \Psi'\square)$ where a' occurs
    $\Xi\square \longleftarrow \Psi'\square$
    if $\phi\square \neq \square$ **then**
      append a deep copy of $\phi'\boxed{a'}$ to the bouquet located by $\Psi\square$
      $\Xi\square \longleftarrow \Psi\boxed{\phi'\square}$
    remove a' from $\Xi\square$
    add $\mathfrak{a}$ to $\mathfrak{H}$

While this might not be obvious at first glance, the two steps of the `pollination` procedure that modify the goal Φ are logically *sound*: indeed, the `copy` operation on line 7 corresponds to an instance of the poll↑ rule, while the `remove` operation on line 9 corresponds to an instance of the poll↓ rule.

Reproduction The second phase of our algorithm is the *reproduction phase*. As its name suggests, it consists in repeatedly applying the reproduction rule srep on the current goal, until there is no more context in which it is applicable[20]. The reproduction phase can be computed by the following tail recursive procedure:

20: In the algorithm of Subsection 8.8.2, this corresponds to the part of the *decomposition* phase that treats negative ∨ formulas.

Procedure reproduction(Φ)

```
foreach ϕ = (Ψ ⊃ Δ) ∈ Φ do
    if there is at least one flower ψ = (⊃ {γᵢ}ᵢⁿ) ∈ Ψ then
        remove ψ from Ψ
        remove every petal from ϕ
        append the petal {γᵢ ⊃ Δ}ᵢⁿ to ϕ
        reproduction(ϕ)
    else
        foreach Ξ ∈ Δ do
            reproduction(Ξ)
```

Decomposition The third and last phase of our proof search algorithm, called *decomposition phase*, is also the most trivial: it simply consists in applying the epet rule in every applicable context in the goal, which amounts to erasing every flower with an empty petal[21]. Unlike the reproduction phase, it cannot be computed directly by a tail recursive procedure, because erasing flowers in the petals of a flower ϕ might make one of the petals of ϕ itself empty. One option is to iterate a tail recursive procedure until a fixpoint is reached, meaning that there are no more flowers with an empty petal. We give instead a more efficient, but non-tail recursive solution:

21: It corresponds to the *popping* phase of bubbles in Subsection 8.8.2, not the decomposition phase: here the term "decomposition" refers metaphorically to the biological process that applies to organic matter, not to the analysis of formulas into their components.

Procedure decomposition(Φ)

```
foreach ϕ = (Ψ ⊃ Δ) ∈ Φ do
    if ∅ ∈ Δ then
        remove ϕ from Φ
    else
        has_empty_petal ⟵ false
        foreach Ξ ∈ Δ do
            decomposition(Ξ)
            if Ξ = ∅ then
                has_empty_petal ⟵ true
                break
        if has_empty_petal then
            remove ϕ from Φ
        else
            decomposition(Ψ)
```

$$
\begin{array}{ll}
 & (a \mathsf{D} ((a \mathsf{D} b) \mathsf{D} b)) \\
\hline
\xrightarrow{\mathsf{P}} & (a \mathsf{D} ((\mathsf{D}\, b) \mathsf{D} b)) \\
\xrightarrow{\mathsf{RD}} & (a \mathsf{D} (\mathsf{D}\, (b \mathsf{D} b))) \\
\hline
\xrightarrow{\mathsf{P}} & (a \mathsf{D} (\mathsf{D}\, (b \mathsf{D} \cdot))) \\
\xrightarrow{\mathsf{RD}} &
\end{array}
\qquad
\begin{array}{ll}
 & ((\mathsf{D}\, a; b) \mathsf{D} (\mathsf{D}\, b; a)) \\
\hline
\xrightarrow{\mathsf{P}} & ((\mathsf{D}\, a; b) \mathsf{D} (\mathsf{D}\, b; a)) \\
\xrightarrow{\mathsf{RD}} & (\mathsf{D}\, (a \mathsf{D} (\mathsf{D}\, b; a)), (b \mathsf{D} (\mathsf{D}\, b; a))) \\
\hline
\xrightarrow{\mathsf{P}} & (\mathsf{D}\, (a \mathsf{D} (\mathsf{D}\, b; \cdot)), (b \mathsf{D} (\mathsf{D} \cdot; a))) \\
\xrightarrow{\mathsf{RD}} &
\end{array}
$$

Figure 10.20.: Life traces for *modus ponens* (left) and identity expansion of disjunction (right)

Cycle of life We can now put together the three phases to form a so-called *lifecycle*. We decompose this in two steps:

1. first we perform one pollination phase;
2. then we cycle reproduction and decomposition phases until a fixpoint is reached.

Procedure lifecycle(Φ, $\mathfrak{H}$)

```
pollination(Φ, 𝕳)
Φ′ ⟵ deep copy of Φ
reproduction(Φ′)
decomposition(Φ′)
while Φ ≠ Φ′ do
    Φ ⟵ deep copy of Φ′
    reproduction(Φ′)
    decomposition(Φ′)
```

Note that the specific order in which phases are sequenced does not matter. The final algorithm, that we call the `life` procedure, is then the fixpoint of the `lifecycle` procedure:

Procedure life(Φ)

```
𝕳 ⟵ ∅
Φ′ ⟵ deep copy of Φ
lifecycle(Φ′, 𝕳)
while Φ ≠ Φ′ do
    Φ ⟵ deep copy of Φ′
    lifecycle(Φ′, 𝕳)
```

Figure 10.20 shows the *traces* obtained by executing the algorithm on two simple tautologies. Lifecycles are separated by horizontal lines, while an arrow with superscript `P` (resp. `RD`) indicates a pollination (resp. reproduction/decomposition) phase.

Assuming that the `life` procedure terminates on every input, the decision procedure is immediate: if $\Phi = \varnothing$ after executing `life(Φ)`, then Φ is a tautology, otherwise it is not.

10.7.3. Empirical assessments

Implementation We have implemented the `life` procedure in the `OCaml` programming language [64]. In fact, we refined the design of the algorithm iteratively, by testing it on a set of 62 tautologies taken from `Edukera`'s logic course [217] (the full set is available in Figure 10.23). Since those are expressed with symbolic formulas, we rely on a straightforward translation of formulas into (bouquets of) flowers, given in Figure 10.21. Out of the 62 tautologies, 53 were proved by the algorithm, and among the 9 unproved tautologies, only the 2 formulas $\neg\neg(\neg\neg A \supset A)$ and $\neg\neg(((A \supset B) \supset A) \supset A)$ are intuitionistically valid, while the other 7 formulas are only classically valid.

$$\begin{aligned} \top^\bullet &= \varnothing \\ \bot^\bullet &= \mathsf{D} \\ (A \wedge B)^\bullet &= A^\bullet, B^\bullet \\ (A \vee B)^\bullet &= \mathsf{D}\ A^\bullet ; B^\bullet \\ (A \supset B)^\bullet &= A^\bullet\ \mathsf{D}\ B^\bullet \\ (\neg A)^\bullet &= A^\bullet\ \mathsf{D} \\ (\forall x.A)^\bullet &= x \cdot\ \mathsf{D}\ A^\bullet \\ (\exists x.A)^\bullet &= \mathsf{D}\ x \cdot A^\bullet \end{aligned}$$

Figure 10.21.: Translation $(-)^\bullet$ of formulas into bouquets

Incompleteness Interestingly on this dataset, the algorithm only fails on formulas that correspond to the double-negation of a classical law. Of course, Theorem 10.6.6 guarantees that there must exist proofs for these formulas, using only ❀-rules. While this may include the epis rules, we conjecture that it is not actually needed to get a complete decision algorithm. Instead, we have implemented an extension of the `life` procedure that accounts specifically for doubly-negated formulas, by first duplicating the negative negation as a special additional case of the pollination phase. In the case of the double-negation elimination law $((((a\ \mathsf{D})\ \mathsf{D})\ \mathsf{D}\ a)\ \mathsf{D})\ \mathsf{D}$, this would give the flower

$$((((a\ \mathsf{D})\ \mathsf{D})\ \mathsf{D}\ a)\ \mathsf{D}), ((((a\ \mathsf{D})\ \mathsf{D})\ \mathsf{D}\ a)\ \mathsf{D})\ \mathsf{D}$$

which is then provable by the algorithm as shown in Figure 10.22.

Encouraged by these positive results, we attempted to validate the algorithm on a more challenging dataset: the ILTP problem library for intuitionistic logic, version 1.1.2 [208]. Unfortunately, it turns out that neither the basic `life` procedure nor its extension are able to prove the following theorem of intuitionistic logic (problem `SYJ106+1`):

[208]: Raths et al. (2007), 'The ILTP Problem Library for Intuitionistic Logic'

$$(((\neg(t \supset r) \supset p) \wedge s) \supset (\neg((p \supset q) \wedge (t \supset r)) \supset (\neg\neg p \wedge (s \wedge s))))$$

$$(((((a\ \mathsf{D})\ \mathsf{D})\ \mathsf{D}\ a)\ \mathsf{D})\ \mathsf{D})$$

$$\begin{aligned} &\xrightarrow{\mathsf{P}} (((((a\ \mathsf{D})\ \mathsf{D})\ \mathsf{D}\ a)\ \mathsf{D}), ((((a\ \mathsf{D})\ \mathsf{D})\ \mathsf{D}\ a)\ \mathsf{D})\ \mathsf{D}) \\ &\xrightarrow{\mathsf{RD}} (((((a\ \mathsf{D})\ \mathsf{D})\ \mathsf{D}\ a)\ \mathsf{D}), ((((a\ \mathsf{D})\ \mathsf{D})\ \mathsf{D}\ a)\ \mathsf{D})\ \mathsf{D}) \end{aligned}$$

$$\begin{aligned} &\xrightarrow{\mathsf{P}} ((((((((((((a\ \mathsf{D})\ \mathsf{D})\ \mathsf{D}\ \cdot)\ \mathsf{D}), a\ \mathsf{D})\ \mathsf{D})\ \mathsf{D}\ \cdot)\ \mathsf{D}), a\ \mathsf{D})\ \mathsf{D})\ \mathsf{D}\ a)\ \mathsf{D}), ((((((((a\ \mathsf{D})\ \mathsf{D})\ \mathsf{D}\ \cdot)\ \mathsf{D}), a\ \mathsf{D})\ \mathsf{D})\ \mathsf{D}\ a)\ \mathsf{D})\ \mathsf{D}) \\ &\xrightarrow{\mathsf{RD}} \end{aligned}$$

Figure 10.22.: Life trace for doubly-negated double-negation elimination law

Worse, the extension seems to loop indefinitely on it.

Non-termination In fact, even the basic version of the `life` procedure does not terminate (at least in reasonable time, i.e. less that 3 minutes) on another intuitionistic theorem (problem `SYJ201+1.001`):

$$\begin{aligned}
&(((((p3 \Leftrightarrow p1) \\
&\quad \supset (p1 \wedge (p2 \wedge p3))) \\
&\quad \wedge ((p2 \Leftrightarrow p3) \supset (p1 \wedge (p2 \wedge p3)))) \\
&\quad \wedge ((p1 \Leftrightarrow p2) \supset (p1 \wedge (p2 \wedge p3)))) \\
&\quad \supset (p1 \wedge (p2 \wedge p3)))
\end{aligned}$$

We did not have the opportunity yet to investigate the sources of incompleteness and non-termination of our algorithm. Indeed, the analysis of life traces on formulas of this size is quite time-consuming, if not untractable for a human. Concerning non-termination, one could hypothesize at this stage, although we find this unlikely, that the algorithm *does* terminate theoretically, but is simply extremely time-inefficient[22]. For instance, it takes 6 whole seconds on a laptop equipped with an i7-10610U processor (8 cores, upto 4.90 GHz per core) and 16 GB of RAM to find a proof of the following formula (problem `SYJ107+1004`):

22: And also space-inefficient, since the bottleneck is the pollination phase that can duplicate flowers in an exponential fashion.

$$\begin{aligned}
&(((((((b \vee a) \vee b) \wedge (b \supset ((b1 \vee a1) \vee b1))) \\
&\quad \wedge (b1 \supset ((b2 \vee a2) \vee b2))) \wedge (b2 \supset ((b3 \vee a3) \vee b3))) \wedge a4) \\
&\quad \supset (a \vee ((b \wedge a1) \vee ((b1 \wedge a2) \vee ((b2 \wedge a3) \vee (b3 \wedge a4))))))
\end{aligned}$$

Implication

$$(A \supset (B \supset A))$$
$$(A \supset ((A \supset B) \supset B))$$
$$((B \supset C) \supset ((A \supset B) \supset (A \supset C)))$$

Conjunction

$$(A \supset (B \supset (A \wedge B)))$$
$$(A \supset (B \supset (C \supset (A \wedge (B \wedge C)))))$$
$$((A \wedge B) \supset (C \supset A))$$
$$((A \wedge B) \supset (C \supset B))$$
$$((A \wedge B) \supset (B \wedge A))$$

Disjunction

$$(A \supset (A \vee B))$$
$$(B \supset (A \vee B))$$
$$(B \supset (A \vee (B \vee C)))$$
$$((A \vee B) \supset ((A \supset B) \supset B))$$
$$((A \vee B) \supset (B \vee A))$$

Negation

$$(\bot \supset \neg A)$$
$$(\neg A \supset (A \supset \bot))$$
$$(A \supset \neg\neg A)$$
$$(\bot \supset A)$$
$$(\neg A \supset (A \supset B))$$
$$(A \supset ((C \supset \neg B) \supset ((A \supset B) \supset \neg C)))$$
$$(((A \supset B) \wedge (A \supset \neg B)) \supset \neg A)$$
$$(((A \vee \neg B) \wedge B) \supset A)$$
$$((\neg A \vee B) \supset (A \supset B))$$
$$((A \supset B) \supset (\neg B \supset \neg A))$$

Constructive logic

$$((A \wedge B) \supset (C \supset A))$$
$$((A \wedge B) \supset (C \supset (B \wedge C)))$$
$$(A \supset (A \wedge (A \vee B)))$$
$$((A \vee (B \wedge C)) \supset (A \vee B))$$
$$((A \vee B) \supset ((A \supset B) \supset B))$$
$$((A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C)))$$
$$((A \supset (B \supset C)) \supset ((A \wedge B) \supset C))$$
$$((A \supset B) \supset ((A \supset C) \supset (A \supset (B \wedge C))))$$
$$((A \supset B) \supset ((A \vee C) \supset (B \vee C)))$$
$$(((A \wedge B) \vee (A \vee B)) \supset (A \vee B))$$
$$(((A \supset C) \wedge (B \supset C)) \supset ((A \vee B) \supset C))$$
$$((\neg A \vee B) \supset (A \supset B))$$
$$((A \supset (B \supset C)) \supset ((A \wedge B) \supset C))$$
$$(A \supset ((A \supset B) \supset (((A \supset B) \supset (B \supset C)) \supset C)))$$
$$(((A \wedge B) \supset C) \Leftrightarrow (A \supset (B \supset C)))$$
$$(((A \vee B) \supset C) \Leftrightarrow ((A \supset C) \wedge (B \supset C)))$$

Associativity

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

Distributivity

$$((A \wedge (B \supset C)) \supset ((A \wedge B) \supset (A \wedge C)))$$
$$((A \wedge (B \vee C)) \Leftrightarrow ((A \wedge B) \vee (A \wedge C)))$$
$$((A \vee (B \wedge C)) \Leftrightarrow ((A \vee B) \wedge (A \vee C)))$$
$$(((A \supset B) \wedge (A \supset C)) \supset (A \supset (B \wedge C)))$$
$$((A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C)))$$
$$(((A \supset B) \vee (A \supset C)) \supset (A \supset (B \vee C)))$$
$$((A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C)))$$
$$(((A \supset B) \supset (A \supset C)) \supset (A \supset (B \supset C)))$$
$$(((A \vee B) \supset C) \Leftrightarrow ((A \supset C) \wedge (B \supset C)))$$

Classical logic

$$(((A \wedge B) \supset (A \wedge C)) \supset (A \wedge (B \supset C)))$$
$$((A \supset (B \vee C)) \supset ((A \supset B) \vee (A \supset C)))$$
$$(A \vee \neg A)$$
$$(A \vee (A \supset B))$$
$$((A \supset B) \vee (B \supset A))$$
$$(\neg\neg A \supset A)$$
$$(((A \supset B) \supset A) \supset A)$$
$$\neg\neg(A \vee \neg A)$$
$$\neg\neg(A \vee (A \supset B))$$
$$\neg\neg((A \supset B) \vee (B \supset A))$$
$$\neg\neg(\neg\neg A \supset A)$$
$$\neg\neg(((A \supset B) \supset A) \supset A)$$

Figure 10.23.: Testing dataset of tautologies from `Edukera`

10.8. The Flower Prover

Proof-by-Action While having a complete and efficient proof search procedure is a nice desideratum, our focus in this thesis is not on full automation — which is not possible anyway as soon as we leave the propositional fragment — but rather on automation that integrates well with, and even improves the experience of building proofs *interactively*. More specifically, the paradigm of interaction we are interested in is that of *direct manipulation* in a GUI. This was our initial motivation for studying the graphical formalism of EGs, and we always kept this objective in mind when developing the flower calculus. In this section, we present ongoing work on the `Flower Prover`, a prototype of GUI in the Proof-by-Action paradigm based on the direct manipulation of flowers, that builds upon the various concepts, rules and metatheory of the previous sections.

Note

Currently, the `Flower Prover` only handles propositional flowers with empty sprinklers.

Implementation The prototype is implemented in `Elm` [55], a modern functional reactive programming language that is particularly well-suited for building GUIs that are based on complex compositional data-structures like flowers. It also natively compiles to `HTML` and `JavaScript`, making it easy to run and test the interface on any device equipped with a web browser. The source code is available on GitHub [65], and the interface is currently hosted online as a simple `HTML` page[23]. The reader is invited to try out the `Flower Prover` in her own browser, although we will try to give self-contained explanations illustrated through various screenshots.

[55]: Czaplicki et al. (2013), 'Asynchronous Functional Reactive Programming for GUIs'

23: `https://www.lix.polytechnique.fr/Labo/Pablo.DONATO/flowerprover/`

Originality We are not the first to identify the potential of EGs for graphical proof building interfaces [154, 244]. However, we believe that the `Flower Prover` is highly original in mainly two respects:

[154]: Loo (2022), *Cross-platform React app for solving existential graph-based proofs*
[244]: Unknown author (2001), *Visual Logic: Peirce's Existential Graphs*

Intuitionistic and goal-oriented it is based on the flower calculus, which proposes a quite unusual, non-exegetic take on EGs. It does so both at the level of *statements*, by building upon the intuitionistic icon of the n-ary scroll; and at the level of *proofs*, by focusing on a goal-oriented reading of rules that emphasizes the importance of *invertibility* and *analyticity* in the inference process, through the distinction between ❀-rules and ✂-rules.

Mobile-friendly it has been designed from the outset as a *responsive*, *touch*-based, *mobile*-friendly interface. Responsivity means that it can be run on screens of varying formats and resolutions, all with the same layout providing a uniform experience across devices. Touch-based means that every pointing interaction is optimized for touch in addition to mouse gestures, so that there is no loss in precision. The combination of these two properties makes the interface perfectly usable on mobile devices like phones and tablets[24], which are becoming increasingly ubiquitous in personal computing. While the current generation of proof assistants targets mostly technical and expert users with text-

24: This is not the case currently for `Actema`, which does not provide a vertical layout, and has poor support for touch interactions. While the former issue could be solved easily, the latter is somehow inherent to the textual nature of formulas: in scenarios where one needs to refer to a specific subterm, it is difficult to design a touch-based selection mechanism that is as precise and straightforward as a mouse-based one.

based, keyboard-driven interactions, we believe that a broadening of audience — especially in educational settings — if at all possible, will require a reinvention of our means of interaction with formal proofs, that is more in phase with contemporary usages of digital devices.

10.8.1. Interaction principles

Modal interface The `Flower Prover` is organized in two main *modes* of interaction, providing different sets of direct manipulation actions on goals[25]:

25: Another example of modal interface is the popular text editor `vim`, with its *normal* mode for high-level manipulation of text through commands and macros, and its *edit* mode for low-level insertion and deletion of characters.

PROOF mode this is the main mode, where goals can be proved by reducing them to the empty bouquet $\varnothing$. It corresponds in purpose to the interactive proof mode of modern proof assistants like `Coq`, `Lean` and `Isabelle`, and to the interactive proof view provided by `Actema`. As will be detailed shortly, there is almost a one-to-one correspondence between PROOF actions, and the ❀-rules of the flower calculus.

EDIT mode in this mode, the user can construct arbitrary flowers by clicking on buttons and filling text fields, or modify the goal by clicking on and dragging flowers around. It corresponds in purpose to the text editor used for writing theories in the *vernacular* language of a proof assistant[26]. EDIT actions implement exactly the ✂-rules of the flower calculus.

26: The term "vernacular" was used for the first time in the context of proof assistants by the founder of the field, N. G. de Bruijn [57].

Remark 10.8.1 A nice analogy for these two modes can be found in the video game *Minecraft*: the *survival* mode, where the player has to gather and craft resources to survive, corresponds to the PROOF mode, where the user has to combine and justify existing statements with the analytic ❀-rules; while the *creative* mode, where the player can build and destroy anything instantly with unlimited resources, corresponds to the EDIT mode, where the user can insert or delete arbitrary flowers with the (synthetic?) ✂-rules.

Figure 10.24 shows side-by-side the same goal representing the flower $(a \supset c), (b \supset c) \supset ((\supset a\,;\, b) \supset c)$, but viewed through the two different modes. At any point during a proof, the user can switch between the two modes by clicking on the corresponding button in the *mode selection bar*, located at the bottom of the screen: PROOF and EDIT modes are mapped respectively to the left button with a checkmark icon, and the middle button with a pencil icon.

It is also possible to Undo/Redo any action, whichever mode it was done in, by clicking on the arrow buttons located on the bottom-right corner of the screen. This is implemented by a simple stack recording the entire state of the application, that is updated every time the user performs an action, and popped/pushed when the user clicks on the Undo/Redo buttons. Since the application state includes the current interaction mode, undoing/redoing an action will automatically switch to the mode in which the action was performed.

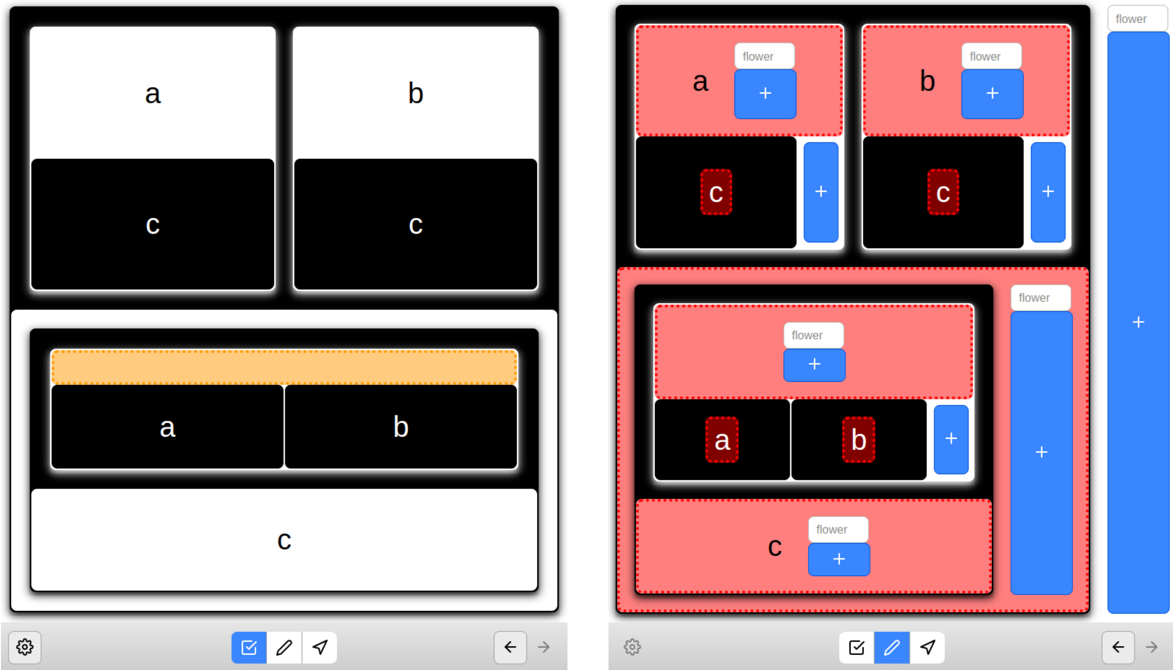


Figure 10.24.: Proof mode (left) and Edit mode (right) of the `Flower Prover`

Proof actions Table 10.2a shows the precise mapping of Proof actions to ❀-rules, together with the associated gestures for triggering them; and Figure 10.25 shows a sequence of screenshots of the `Flower Prover`, capturing the execution of a sequence of Proof actions reducing the subgoal $(\supset a; b) \supset c$ to $b \supset c$. A few comments are in order:

Pollination The most central actions are those implementing the pollination rules poll↓ and poll↑, called respectively Justify and Import. In fact, they are *less* general than those rules: one can only Justify flowers that are *atomic* by clicking on them, and Import *non-atomic* flowers by dragging and dropping them at the desired location. We conjecture that these restrictions do not jeopardize the completeness of ❀-rules, and correspond to the process of η-expansion in λ-calculus.

Suggestions The fourth screenshot in Figure 10.25 shows the flower $\phi \coloneqq a \supset c$ being dragged in the process of an Import action. If you look closely, you will notice that there are many areas whose border is highlighted with dashed yellow lines: these correspond to all contexts $\Xi\square$ where ϕ can be imported, i.e. such that $\phi \succ \Xi\square$. This is a first form of *suggestion* in the `Flower Prover`, indicating available valid actions to the user through visual feedback[27].

In fact, every Proof action has an associated visual cue, guiding interactive proof search by suggesting to the user areas of the goal where she might want to focus her attention. Since every action other than Import is performed by a *click* gesture, the area that is highlighted corresponds precisely to the area that can be clicked for triggering the action: either a green box enclosing the justifiable atom for Justify actions, an orange box covering the empty pistil for EFQ, Case and Unlock actions, or a green box covering the empty petal for QED actions.

27: A similar mechanism is implemented in `Actema`, where subterms that are possible drop targets for DnD actions are also highlighted (see Section 3.2).

Fencing We have already mentioned that we suspect that the epis rule might be admissible. However if it is not, one needs a corresponding graphical action in PROOF mode. While it is currently not implemented, we plan to add a *selection* mechanism that allows the user to select a set of flowers in the goal. Then, we could add a Fence action, whose effect is to enclose the selected flowers in a petal attached to an empty pistil. This action could be mapped to a dedicated button in the toolbar that is visible only in PROOF mode, and enabled only when the selected flowers are juxtaposed in the same garden.

Since every PROOF action implements a ❀-rule, it is guaranteed to be *invertible*: the user never needs to Undo a PROOF action in order to complete a proof, because it always preserves the provability of the goal. Of course it might still be desirable to do so in specific cases, such as Import actions that may create unneeded copies of flowers[28].

28: In this specific case, one might want to relax the atomic restriction on Justify actions, in order to avoid the recourse to Undo actions, which can only be performed at the top of the history stack.

EDIT actions Table 10.2b shows the precise mapping of EDIT actions to ✂-rules, together with the associated gestures for triggering them. Like the edit mode of `vim`, EDIT actions are used to *insert* and *delete* arbitrary flowers in the goal:

Insertion The main interface mechanism that is currently implemented is the `Add` button: since the grow rule allows to insert any flower in a positive bouquet (i.e. add a new subgoal, just like the cut rule in sequent calculus), we expose buttons in all the corresponding areas (blue "+" buttons in Figure 10.24), that can be clicked to insert a new flower precisely at the location of the button. There are two usage scenarios:

- if the user wants to insert an atomic flower, she can enter the name of the atom in a text field placed above the button. Clicking on the button will then insert an occurrence of this atom;
- if the user wants to insert a non-atomic flower, she can leave the text field empty. Clicking on the button will then insert an empty flower with a single petal.

In both cases, the inserted flower is marked internally by the system with a `grown` tag: this means that as long as the user does not leave EDIT mode, she can perform arbitrary insertions and deletions inside of the `grown` flower, disregarding any polarity constraint normally imposed by ✂-rules.

Dually, the glue rule is implemented by exposing `Add` buttons in all negative corollas: those have the effect of growing a new empty petal, that can be further edited through arbitrary insertions and deletions.

`Grown` flowers/petals are distinguished visually by having their border painted in blue. Leaving EDIT mode then has the effect of *committing* every EDIT action, i.e. removing every `grown` tag in the entire goal. This mechanism enables an incremental, step-by-step construction of flowers, that is still sound logically with respect to ✂-rules.

Deletion Deleting flowers and petals is a more straightforward process: one just has to click on the corresponding area, which is highlighted in red. To avoid overlap, the area of a flower is identified with its pistil.

Thus areas subject to deletion are negative atoms and flowers (crop rule), positive petals (pull rule), as well as any area marked as `grown`.

Since every Edit action implements a ✂-rule, it is guaranteed to be *non-invertible*: the user might need to Undo an Edit action in order to complete a proof, because it may break the provability of the goal.

Navigation mode The reader might have noticed that there is a third button with a navigation icon on the right of the mode selection bar. It can be used to enter Navigation mode, the last mode of interaction that we intend to implement in the future. The idea is that on real-life goals, both the size and level of nesting of flowers will quickly render the interface unusable, both for reading/understanding the content and structure of goals, and manipulating them through pointing.

The purpose of the Navigation mode is then to enable the user to *focus* on a specific subgoal, by simply clicking on the corresponding nested flower. This would make the subgoal take up the whole screen, hiding the outer context from view. Dually, it should also be possible to unfocus a previously focused subgoal — e.g. by clicking again on it — so that the full tree structure of the goal can be freely navigated. Proof-theoretically, the Navigation mode implements the *functoriality* of rules, i.e. the fact that they can be applied in contexts of *arbitrary* depth.

Remark 10.8.2 This way of navigating tree structures represented as nested areas is typical of zoomable user interfaces, a strand of GUI that has been developed by many pioneers in the field of human-computer interaction such as Ivan Sutherland in his `Sketchpad` system [234], and Alan Kay in his `Smalltalk` system [99].

[234]: Sutherland (1964), 'Sketchpad: A Man-machine Graphical Communication System'

[99]: Goldberg et al. (1976), *SMALLTALK-72 INSTRUCTION MANUAL*

Automation The last feature of the `Flower Prover` that we have implemented is the Auto Proof action. It is similar in purpose to the `auto` tactic of `Coq`, that tries to simplify the goal by performing a limited (but customizable) amount of automation. The Auto action is mapped to a dedicated button in the bottom-left corner of the screen, which is only enabled in Proof mode (see Figure 10.24).

The idea is quite simple: since all click actions available to the user are pre-computed by highlighting the corresponding areas, there can only be a finite number of them. So why not try to apply them all automatically? Applying a click action might generate new ones in the resulting goal, so we have to perform this until a fixpoint is reached. This is very much like the reproduction and decomposition phases from the `life` procedure of Section 10.7, except that we also apply the poll↓ rule (Justify actions) wherever possible. The only Proof action that is not considered is the only DnD action, Import. This is not surprising, since it corresponds to the poll↑ rule, which is the main source of complexity in the pollination phase of the `life` procedure, because of its ability to duplicate flowers of arbitrary size.

In fact, one could fine-tune the level of automation by considering only a *subset* of all types of click actions. This is already what we do by default, by leaving the application of Case actions to the user. This is motivated by the fact that the latter can induce an explosion in the size of the goal. One could even leave the configuration of automated action types to the user with a dedicated interface. This could include an additional option for executing Auto systematically after every (other) PROOF action, removing the need to click on the Auto button. In this setting, any proof in the `Flower Prover` could be reduced to a sequence of Import and Case actions.

10.8.2. Towards a unified workflow

Theories and goals In the proof view of modern proof assistants like `Coq` and `Lean`, there is no distinction between local and global contexts: a subgoal will inherit automatically every hypothesis from its parent subgoals, which are flattened into a big unstructured list. To recreate this distinction and reduce the size of goals to a manageable level, the only interface mechanism offered to the user is to exit interactive proof mode, and outsource chunks of the local context as additional global lemmas and definitions in the current theory file.

Thus the user has to juggle between two different interfaces that manipulate two distinct data structures: a traditional text editor for modifying *theories*, and an IDE for writing and executing proof scripts that modify *goals*, themselves visualized in a separate proof view. This results in a duplication of means to achieve essentially the same things: for instance, reordering two lemmas will require to cut and paste one of them in the theory file, while reordering two hypotheses will require the use of a dedicated `move` tactic. Other examples can be found for renaming definitions, applying lemmas, constructing functions, etc. Crucially, the two interfaces cannot communicate straightforwardly with eachother. In fact, communication is completely one-way: the user can only invoke definitions and lemmas of the theory from her proof script, by referring to their names.

The `Flower Prover` can theoretically solve this divide, because it works on a single data structure: flowers represent *at the same time* the current goal to be proved in PROOF mode, and the theories that are being built in EDIT mode. Thus there are still two distinct modes/interfaces, but they work in unison on the same data. The only (major) current limitation, is that we do not have any way to *save* proved lemmas for later reuse, because proving a flower amounts to *erasing* it from the current goal. In a sense, theories built in EDIT mode are only *transient*: they live in *working* memory, and are disposed of as soon as they become justified in PROOF mode; while we would like them to be *persistent*, recorded in *long-term* memory along with their justifications (which would stay hidden from the user by default). We will discuss in Section 10.9 some research directions that we envision to achieve the latter.

Statements and proofs Our above example of "redundant" manipulations targets imperative tactic languages, but the argument equally applies to more declarative languages like `Isar`: the point is that the *proof* language, be it imperative or declarative, is separated both conceptually and through its available means of interaction from the language of *statements* used to build theories. And this separation between proofs and statements is a natural one that is hard to question, since it is rooted in what is arguably the most important inspiration of formal logic, and also the form in which informal mathematics present themselves: *natural language*. Indeed, symbolic formulas reproduce the grammatical structure of sentences expressing logical propositions, and formal proofs reproduce the inferential structure of arguments built from sequences of sentences.

Context navigation After this little conceptual *aparté*, let us come back to the problem of managing contexts in proofs. In the `Flower Prover`, the *local* context is naturally represented as *everything that is displayed on-screen*. This includes hypotheses that are available from pistils at various levels, but also potentially alternative goals (adjacent petals) and further subgoals (positively nested flowers). Then rather than being segregated in a separate interface (the text buffer of the theory), the *global* context is simply the *entire* goal. In fact, there is no reason anymore to make a terminological distinction between goals and theories: a goal is just a theory that has yet to be justified, which can itself be identified with a partial proof (or a "proof term with holes" in type-theoretical parlance)[29].

29: We will come back to this idea of merging proofs and statements in the same data structure in the conclusion, when discussing *development calculi* and the Curry-Howard correspondence. Note however that it is already at work in *dependent* type theory, where proof terms can freely occur inside types. This is exemplified in the `Agda` proof assistant, where all manipulations are done directly on the partial proof/program text.

It would still be useful to be able to aggregate automatically the set of all lemmas, definitions and hypotheses available in a focused subgoal located in $\Phi\square$, so that the user does not need to navigate up and down the goal/proof tree all the time. This can be done with the help of the pollination relation (Definition 10.3.5), by defining the set of *available* flowers of $\Phi\square$ as the the union $\mathsf{ctxt}(\Phi\square) \triangleq \{\phi \mid \phi \succ \Phi\square\}$.

In terms of UI, we could then add a so-called *shelf* that displays $\mathsf{ctxt}(\Phi\square)$ in all interaction modes. We anticipate only two kinds of interaction with any flower $\phi \in \mathsf{ctxt}(\Phi\square)$ in the shelf:

Pollination (in Proof mode) the user can perform an Import action by dragging ϕ;

Jump to definition (in Navigation mode) the user can focus on the subgoal where ϕ originates by clicking on ϕ.

Since the shelf might contain *a lot* of hypotheses, it will be important to provide efficient ways to *filter* or *search* through its content. We imagine three main ways of doing so[30]:

30: The first two types of filtering are already available in most proof assistants, e.g. the `Search` command of `Coq`.

By name the user can type the *name* of a hypothesis in a search bar. This implies that flowers have the ability to be named by the user.

By structure the user can specify a *pattern* that must be satisfied by all hypotheses in the shelf. A pattern is just a flower that contains pattern variables, which can match any flower. Thus patterns might be built with the same tools offered in Edit mode.

By selection the user can select subterms of the goal, and then ask the system to display only hypotheses that can *interact* with these subterms[31].

31: Here we imagine something along the lines of what we did for `Actema` (see Section 4.2).

Table 10.3.: Graphical actions of the `Flower Prover`

(a) Proof actions

Action	Gesture	❀-rule
Justify	Click on a	$\dfrac{\Xi\boxed{}}{\Xi\boxed{a}}\ \text{poll}\!\downarrow$
Import	DnD of ϕ into $\Xi\Box$	ϕ non-atomic $\dfrac{\Xi\boxed{\phi}}{\Xi\boxed{}}\ \text{poll}\!\uparrow$
QED	Click on empty petal	$\dfrac{}{\gamma \supset \cdot\,;\Delta}\ \text{epet}$
EFQ	Click on empty pistil	$\dfrac{\dfrac{}{\Phi \supset \cdot}\ \text{epet}}{\Phi, (\cdot \supset) \supset \Delta}\ \text{srep}$
Case	Click on empty pistil	$n \geq 2\ \dfrac{\Phi \supset \{\gamma_i \supset \Delta\}_i^n}{\Phi, (\cdot \supset \{\gamma_i\}_i^n) \supset \Delta}\ \text{srep}$
Unlock	Click on empty pistil	$\dfrac{\Phi, \Psi \supset \Delta}{\Phi, (\cdot \supset \Psi) \supset \Delta}\ \text{epis}\!\downarrow$

(b) Edit actions

Action	Gesture	✂-rule
Grow	Add button in positive bouquet	$\dfrac{\Xi^{+}\boxed{\supset \cdot}}{\Xi^{+}\boxed{}}\ \text{grow}$
Glue	Add button in negative corolla	$\dfrac{\Xi^{-}\boxed{\gamma \supset \cdot\,;\Delta}}{\Xi^{-}\boxed{\gamma \supset \Delta}}\ \text{glue}$
Insert	Add button in grown bouquet/corolla	grow/glue
Delete	Click on grown flower/petal	grow/glue
Crop	Click on negative flower	$\dfrac{\Xi^{-}\boxed{}}{\Xi^{-}\boxed{\phi}}\ \text{crop}$
Pull	Click on positive petal	$\dfrac{\Xi^{+}\boxed{\gamma \supset \Delta}}{\Xi^{+}\boxed{\gamma \supset \delta;\Delta}}\ \text{pull}$

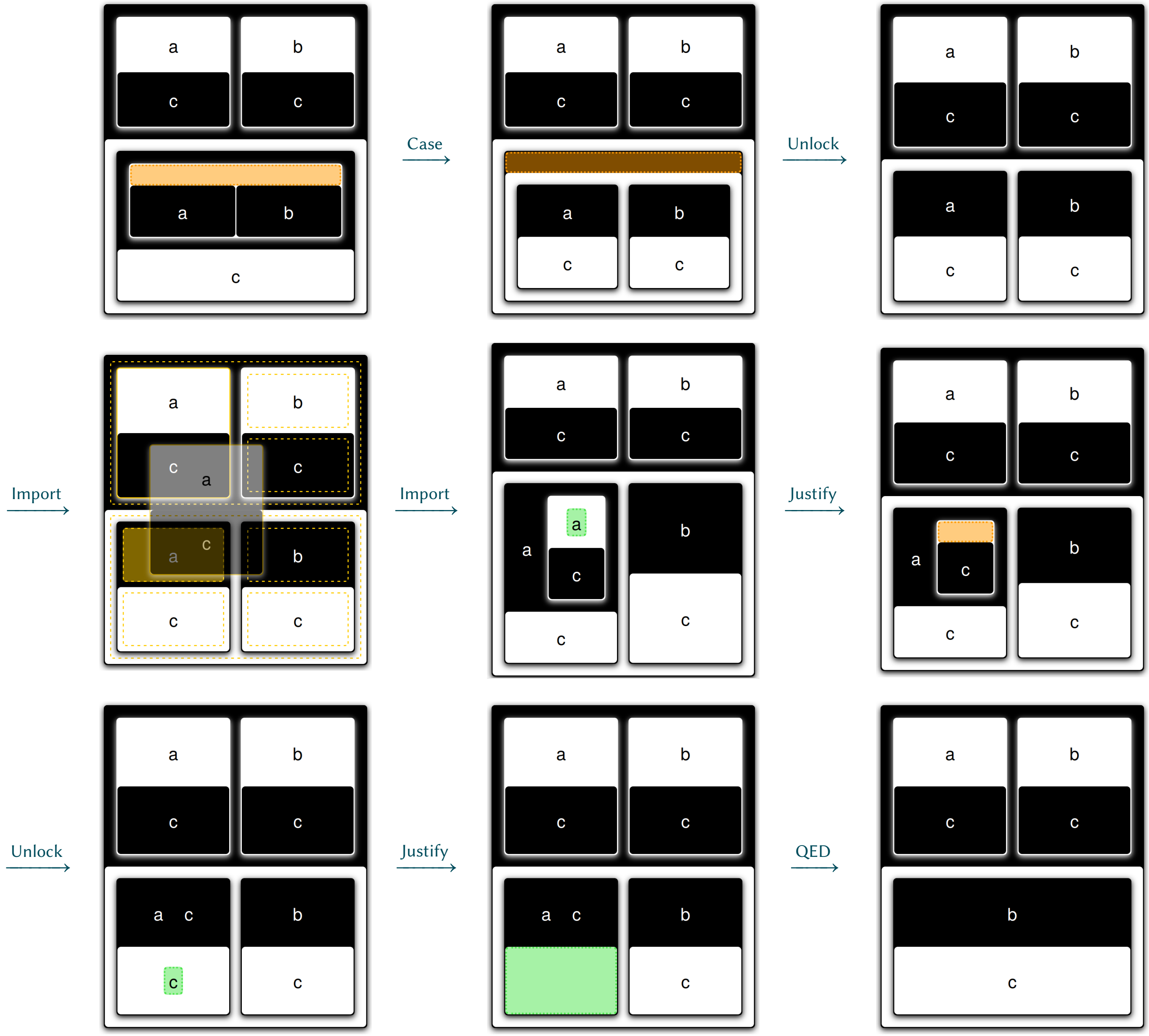


Figure 10.25.: A sequence of Proof actions in the `Flower Prover`

10.9. Conclusion

10.9.1. Related works

Intuitionistic EGs In the original IEGs system of Oostra introduced in [190], the srep rule is replaced by an extended (de)iteration rule, that allows to duplicate/merge not only identical flowers, but also identical *petals*, under the condition that they are attached to the same pistil[32] (rules iter and deit in Figure 10.26). Thus in a sense, (de)iteration on petals is not as *deep* as on whole flowers, where the two identical flowers can be separated by an arbitrary number of layers; which might seem like an arbitrary restriction. A posteriori, we rationalize this choice by seeing it as an attempt to stay close to the original system Alpha of Peirce. In particular, (de)iteration on petals is compatible with the quest for *illative atomicity*, where all rules should be expressed in terms of insertions and omissions (Section 9.4); while the srep rule is not. In our case, this is justified by our quest for an *invertible* calculus (natural fragment): indeed to simulate srep with petal (de)iteration, one also needs the non-invertible ✂-rule crop (as well as the rule epis↓ of Figure 10.13), as illustrated by the srep-free proof of Figure 10.27.

$$\frac{\gamma \supset \delta ; \Delta}{\gamma \supset \delta ; \delta ; \Delta}\ \text{iter}$$

$$\frac{\gamma \supset \delta ; \delta ; \Delta}{\gamma \supset \delta ; \Delta}\ \text{deit}$$

Figure 10.26.: (De)iteration rules for petals

[190]: Oostra (2010), *Los gráficos Alfa de Peirce aplicados a la lógica intuicionista*

32: "Cualquier lazo puede iterarse adherido *al mismo corte*" [190, p. 46].

In addition to his seminal work in [191], Oostra describes in [191] a natural extension of intuitionistic Alpha with LoIs, in order to get an intuitionistic version of Beta. He also gives in [192] formal soundness and completeness proofs for intuitionistic Alpha, based on a linear notation for graphs.

[191]: Oostra (2011), *Gráficos existenciales Beta intuicionistas*

Ma and Pietarinen have developed in [159] their own system of intuitionistic EGs for propositional logic, with a different set of inference rules than Oostra's. They give a more systematic proof theory, including deduction, soundness and completeness theorems with respect to Heyting algebras.

Our work brings several new contributions on top of those:

Variadicity Our multiset-based definition of flowers captures faithfully the *variadic* nature of juxtaposition and n-ary scrolls in the diagrammatic syntax. In contrast, previous formalizations rely on a restricted inductive syntax which only captures graphs that are isomorphic to formulas built with binary connectives[33].

33: Thus we reject the claim made in [159] that their system is "solving the problem of defining a sequent calculus in the style of deep inference for intuitionistic propositional logic". In our opinion, flowers are closer to a form of nested sequent, although there is no consensus in the literature on what makes some inductive data structure a nested sequent.

$$\begin{array}{cl}
\hline
(a \supset c), (b \supset c), c \supset \cdot & \text{epet} \\
\hline
(a \supset c), (b \supset c), c \supset c & \text{poll}\downarrow \\
\hline\hline
(a \supset c), (b \supset c), (\supset (\supset c)) \supset c & \text{epis}\downarrow \\
\hline
(a \supset c), (b \supset c), (\supset (\supset c) ; (\supset c)) \supset c & \text{iter} \\
\hline\hline
(a \supset c), (b \supset c), (\supset (\supset c), a ; (\supset c), b) \supset c & \text{crop} \\
\hline\hline
(a \supset c), (b \supset c), (\supset (a \supset c), a ; (b \supset c), b) \supset c & \text{poll}\downarrow \\
\hline\hline
(a \supset c), (b \supset c), (\supset a ; b) \supset c & \text{poll}\uparrow
\end{array}$$

Figure 10.27.: Simulating the srep rule by iterating petals

Intuitionistic binders While replacing LoIs with binders and variables has already been done by Sowa in the context of classical EGs [226], it seems like we are the first to adapt the idea to the intuitionistic setting.

Analyticity To our knowledge, we are the first to give a Kripke semantics to a syntax based on EGs, and to use this to obtain an analyticity result, as discussed in Section 10.6.

Invertibility The natural fragment of the flower calculus appears to be the first proof system based on EGs where all rules are *invertible*.

Focusing There is a formal connection between the poll↑ rule of the flower calculus, and the *absorption* rule [A] of the dyadic system Σ_2 of Andreoli, that handles the focusing behavior of exponentials in linear logic [5]. Indeed, both rules duplicate a formula available in the (non-linear) context of the sequent/location where the rule is applied, in order to enable further usage of the formula at said location. While the absorption rule removes the need for permutation-equivalences between proofs involving the *contraction* rule, the identity rule [I] of Σ_2 removes the need for the *weakening* rule by discarding the non-linear context in one go, just as the epet rule of the flower calculus renders the crop rule admissible.

[5]: Andreoli (1992), 'Logic Programming with Focusing Proofs in Linear Logic'

Sonia Marin has noticed the connection between *bipoles* in focused proofs, and the class of geometric/coherent formulas, where the former are seen as a generalization of the latter [160]. This is to be related to our own identification of n-ary scrolls/flowers as a recursive generalization of coherent formulas at the end of Section 10.1.

[160]: Marin et al. (2022), 'From axioms to synthetic inference rules via focusing'

Two years earlier, Brock-Nannestad and Ilik had already made some implicit connections between focused proofs and coherent formulas, through their *exponential normal form* for intuitionistic formulas based on Tarski's highschool identities [28]. Quite remarkably, first-order formulas in their exponential normal form have the exact same structure as flowers [28, Definition 4.2]. However, the sequent calculus HS based on them makes the tradeoff opposite to that of the natural fragment ❀ of the flower calculus: every inference rule is *non*-invertible, but the calculus is contraction-free. One advantage of this tradeoff is that they can easily show termination of proof search, while we have not found a terminating procedure yet for the flower calculus. The authors also mention that HS could be turned into a deep inference calculus in the style of G4ip[34].

[28]: Brock-Nannestad et al. (2019), 'An Intuitionistic Formula Hierarchy Based on High-School Identities'

34: This is just another name for the system LJT of Dyckhoff already mentioned in Chapter 8.

Development calculi In Section 10.8, we have seen how the rules of the flower calculus can be understood as a set of (graphical) tactics for building partial proofs interactively. In Chapter 3 of his thesis [11], Ayers calls such systems *development calculi*. In particular, he presents his own development calculus inspired by McBride's OLEGs system [167] and G&G's prover [87] called the `Box` calculus, where both goals and partial proofs are represented by the same `Box` data structure. Once again, `Box`es seem to share a very similar structure with flowers, which was here motivated by the need to avoid backtracking by having the ability to maintain a disjunction of goals with so-called *disjunctive pairs*, corresponding to the petals of flowers. The main difference is that the `Box` calculus is based on dependent type theory instead of first-order logic: this allows to store the

[11]: Ayers (2021), 'A Tool for Producing Verified, Explainable Proofs.'

[167]: McBride (2000), 'Dependently Typed Functional Programs and their Proofs'

[87]: Ganesalingam et al. (2017), 'A Fully Automatic Theorem Prover with Human-Style Output'

partial proof terms inside of the Boxes themselves, while this information is lost during the construction of flowers (but might be reconstructed from the sequence of graphical actions and the initial goal).

Ayers also mentions the category-theoretical treatment of development calculi by Sterling and Harper [231], that abstracts from any particular type of judgment. Thus it might be possible to fit the flower calculus into this framework, by identifying the set of flowers $\mathbb{F}$ as a category of *nested judgments*[35].

[231]: Sterling et al. (2017), *Algebraic Foundations of Proof Refinement*

35: Nested judgments are already considered in some recent categorical semantics of type theory, and in particular those in Sterling's thesis [230]. See also [125] for a (technical) introduction to the subject.

Subformula linking Our notion of *vehicle* (Definition 10.7.5) takes its terminology from Girard, who started giving this name to the set of axiom links of a proof structure in his transcendental syntax[36] [95]. But the idea of connecting dual occurrences of atoms, and thus forming a graph with an associated adjacency matrix whose structure can be exploited in proof search, really dates back to the *connection method* developed independently by Bibel and Andrews in the 1970s [20]. Otten and Kreitz have adapted the connection method to intuitionistic logic [197], stating that it is especially well-suited in an interactive theorem proving environment. Thus it might be instructive to learn from their proof search algorithm to fix ours.

36: Also, it conveys nicely the idea that the vehicle is the fundamental structure that *drives* the proof search algorithm.

[95]: Girard (2017), 'Transcendental syntax I: deterministic case'

[20]: Bibel et al. (2009), 'Connection method'

[197]: Otten et al. (1995), 'A connection based proof method for intuitionistic logic'

In fact all proof search procedures designed in this thesis, whether for bubble calculi (Subsection 8.8.2) or the flower calculus (Section 10.7), rest on the fundamental observation coming from the subformula linking methodology of Chaudhuri [37], that the construction of proofs in deep inference systems can be driven efficiently and incrementally by the connection of dual atoms. With its pollination rules, the flower calculus allows for a particularly elegant implementation of subformula linking that abstracts away from the syntactic bureaucracy of symbolic connectives, as witnessed by the pollination phase of our search procedure. In Subsection 8.8.2, we sketched some ideas that blur the frontier between automated and interactive proof search, notably with the so-called *rule of thumb* which is another manifestation of subformula linking. This integration of automated and interactive aspects is also at work in the `Flower Prover`, and it would be interesting to investigate further how to incorporate our drag-and-drop proof tactic (Chapter 2), but also other symbolic manipulation techniques introduced in the first part of this thesis, into the iconic framework of the `Flower Prover`.

[37]: Chaudhuri (2013), 'Subformula Linking as an Interaction Method'

Analyticity We have not discussed the rationale behind our notion of analyticity, be it historical or formal arguments explaining its origins, motivations and consequences. In a recent article [32], Bruscoli and Guglielmi propose such a detailed discussion around a precise and generic definition of analyticity for deep inference proof systems (especially the calculus of structures), which at a glance seems to encompass our own definition. It would be interesting to study more deeply their work, and related parts of the deep inference literature concerned with analyticity and its applications to efficient proof search procedures [41, 43, 107, 137, 139].

[32]: Bruscoli et al. (2019), 'On Analyticity in Deep Inference'

[41]: Chaudhuri et al. (2011), 'The Focused Calculus of Structures'

[43]: Chaudhuri et al. (2016), 'Focused and Synthetic Nested Sequents'

[107]: Guenot (2011), 'Nested Proof Search as Reduction in the Lambda-Calculus'

[137]: Kahramanogullari (2014), 'Interaction and Depth against Nondeterminism in Proof Search'

[139]: Kahramanoğulları (2006), 'Reducing Nondeterminism in the Calculus of Structures'

10.9.2. Future works

Metatheory In Section 10.3, we already mentioned the variant ❀ ∖ {epis} ∪ {crep} of the natural fragment, that we conjecture to enjoy both soundness, completeness and a deduction theorem. But these last two results shall prove particularly harder to prove, and we currently have very few insights into how to extend the proofs of this chapter to this setting. Also, this is not withstanding the fact that we do not really see any practical applications for such results as of yet. Our initial motivation was to show the admissibility of the epis rule, because it never appears in concrete proofs. But if this requires adding the crep rule instead, then it greatly reduces the pratical interest of the whole endeavor, since the crep rule does not look particularly well-suited to either automated or interactive theorem proving.

Another line of research would concern properties of *locality*, in the sense coined by the deep inference community with systems like SKS (see Section 9.6). As mentioned in Section 10.8, we conjecture that the poll↓ and poll↑ rules can be restricted respectively to atomic and non-atomic flowers. But this is less satisfying than in the calculus of structures, where one component of poll↑, the duplicating *contraction* rule, can be restricted to atomic formulas. This probably comes from the fact that poll↑ also serves the purpose of *moving* flowers deeper, as witnessed by the DnD Import action of the `Flower Prover`: in the calculus of structures, this role is fulfilled by *switch* rules, which cannot be restricted to atomic structures. The only solution might be to *simulate* a local calculus of structures for intuitionistic logic, like the system ISp of Tiu [241].

[241]: Tiu (2006), 'A Local System for Intuitionistic Logic'

Lastly, it would be interesting to exhibit an *internal*, syntactic procedure for eliminating cultural ✂-rules in proofs, just like Gentzen showed cut-elimination in sequent calculus[37]. In this work we preferred a more semantic approach, because it was simpler and at the right level of abstraction for our needs. We might be able to take some inspiration from the cut-elimination proofs of calculi of structures, which are indeed notoriously involved.

37: A sort of *cult*-elimination, so to speak.

Automated proof search We shall investigate the current sources of non-termination and incompleteness for our `life` proof search procedure, through further testing on the ILTP dataset. If we succeed in passing all tests, the natural continuation will be to provide formal proofs of termination and completeness. A follow-up direction would be to extend our algorithm to the first-order setting by adding heuristics for handling sprinklers, thus losing completeness.

Another direction of research would consist in comparing our algorithm to existing search procedures for EGs, in particular one that was originally developed by Peirce, and described by Oostra in [193].

[193]: Oostra (2022), 'Advances in Peircean Mathematics: The Colombian School'

Curry-Howard We have begun to sketch some ideas for a Curry-Howard correspondence, where flowers and Proof actions for justifying them (❀-rules) are identified respectively with *normal* and *neutral* terms of the simply-typed λ-calculus. For instance, the computational counterpart of the rule poll↓ in pistils would be a kind of *function application* expressed by the following app rule, which is highly reminiscent of the instantiation rule ipis:

$$\dfrac{\Xi\boxed{t\,u : (\Phi\{u/x\} \supset \Delta\{u/x\})}}{\Xi\boxed{t : (\Phi, x : \phi \supset \Delta)}}\ \mathsf{app}$$

Given a flower ϕ and a neutral term t, i.e. an n-ary function application of the form $x\,t_1 \ldots t_n$, the expression $t : \phi$ is a *term annotation*, that should be read in context as "this occurrence of ϕ is justified by t". Interestingly, ϕ itself may contain term annotations, mimicking the fact that normal and neutral terms can be defined by mutual recursion. Then in the app rule, we do not just erase the formula ϕ as in the poll↓ rule, but also keep track of the flow of information by appending the argument u to the justification t (where $(u : \phi) \succ \Xi\square$), and substituting u to every occurrence of the hypothetical justification x of ϕ in Φ and Δ.

As of now the syntax of annotated flowers is not yet stable, and it is unclear what would be the computational interpretation of flowers with $n \neq 1$ petals. In particular for disjunctive flowers ($n > 1$), it seems that we are closer to a notion of non-deterministic or parallel computation, than to the usual branching computation of sum or inductive types.

If our intuition is right, then the fact that flowers correspond to (normal) λ-terms would embody syntactically a recent motto from Miquel stemming from his study of the foundations of forcing and realizability in implicative algebras, where "elements can be seen both as truth values and as (generalized) realizers, thus **blurring the frontier between proofs and types**"[38] [176]. Or as he put it in a recent talk [177], we get the ultimate Curry-Howard identification:

Realizer = Program = Formula = Type

38: Another recent, related incarnation of this phenomenon is the correspondence uncovered by Haydon between a linear version of EGs already appearing in Peirce's writings, and the proof nets of Girard for linear logic [114].

[176]: Miquel (2020), 'Implicative algebras'

This could also form the basis for further studies on the connections between EGs and (dependent) type theory, and ultimately lead to a tight integration of the `Flower Prover` with proof assistants based on the latter such as `Coq`, `Lean` and `Agda`. Existing explorations of the links between type theory and deep inference include, in historical order:

- a first attempt by Brünnler and McKinnley to devise a Curry-Howard correspondence for a simple intuitionistic deep inference calculus with conjunction and implication [30];
- the thesis of Nicolas Guenot, and more precisely the part on "Nested Proofs as Programs" where he gives a correspondence between simply-typed λ-calculi with explicit substitutions at one end, and calculi of structures (Chapter 6) and nested sequent calculi (Chapter 5) for the implicational fragment of intuitionistic logic at the other end [108];
- the atomic λ-calculus, a simply-typed λ-calculus with explicit sharing that has a Curry-Howard correspondence with proofs in the formalism of *open deduction* [113];
- a type system for interaction nets based on a calculus of structures for Multiplicative Exponential Linear Logic [91];

[30]: Brünnler et al. (2008), 'An Algorithmic Interpretation of a Deep Inference System'

[113]: Gundersen et al. (2013), 'Atomic Lambda Calculus'

[91]: Gimenez et al. (2013), 'The Structure of Interaction'

- the thesis of Fanny He, that explores a classical variant of the atomic λ-calculus based on Saurin's $\Lambda\mu$-calculus [116];
- the *spinal* atomic λ-calculus, an extension and improvement on the atomic λ-calculus based on a computational interpretation of the *switch* rule [220];
- the *collection calculus* that subsumes resource, intersection-typed and simply-typed λ-calculi, with a type system again in open deduction [109];
- Ongoing research by Kaustuv Chaudhuri to extend subformula linking to dependent type theory[39].

[116]: He (2018), 'The Atomic Lambda-Mu Calculus'

[220]: Sherratt et al. (2020), 'Spinal Atomic Lambda-Calculus'

[109]: Guerrieri et al. (2021), 'A Deep Quantitative Type System'

39: Private communication.

The work of Guenot on computational interpretations of nested sequent calculi seems closest to the syntax of flowers. Indeed, nested sequents are variadic by nature, and his version of nested sequents in particular exploits the possibility to have *negative* occurrences of sequents. Combined with the dependently-typed Boxes of Ayers mentioned earlier (which cannot be nested negatively), this should provide great insights for the powerful, dependently-typed version of the flower calculus that we seek for.

Flower Prover We have already described many features in Section 10.8 that we intend to implement in the future. This includes the Navigation mode, the ability to *select* flowers, the Fence Proof action, and the *shelf* mechanism.

The next step would be to support first-order reasoning by adding sprinklers and first-order terms, and devising graphical actions for the rules {ipis, ipet} in Proof mode, and {apis, apet} in Edit mode.

Last but not least, we want to provide a way to *save* proved lemmas along with their proof, so that they can be reused and read statically. This will be crucial for *proof evolution*[40], and will probably rely on the computational, dependently-typed version of the flower calculus sketched above, where proof terms can appear inside flowers.

40: See also Subsection 6.6.4.

10.9.3. Theory vs. Practice

Finally, it should be noted that Peirce did not think of EGs as a calculus that could aid in performing reasoning *per se*, but rather as a tool for analyzing the finer structure of logical endeavor [214, pp. 110–111]:

> [...] the purpose which the system was designed to fulfill was "to enable us to separate reasoning into its smallest steps so that each one may be examined by itself" (Ms 455, p. 2). The aim was not to facilitate reasoning, but to facilitate the study of reasoning.

The various achievements presented in this chapter incite us to depart from this conception. Indeed, our particular viewpoint on the illative transformations, that emphasizes goal-reduction through invertible and analytic rules, enabled us to design a novel and promising type of graphical interface for interactive proof building, which integrates easily and elegantly some (limited) forms of automation. This was also made possible by the use of variables instead of lines of identity, trading a heavy graphical apparatus with local inference rules for a simple, well-known textual syntax with complex (but automated) global dynamics in the form of substitutions. Hence we believe the *opposite*, that EGs *can* form the basis for an ergonomic calculus of logical deduction, in addition to being a powerful tool for meta-logical analysis.

Appendix

Symmetric Bubble Calculi A.

A.1 Soundness 248
A.2 Completeness 255

A.1. Soundness

Lemma A.1.1 (Generalized weakening) $[\![S]\!]^+ \leq [\![S \uplus (\Gamma \Rightarrow \Delta)]\!]^+$.

Proof. Let $S = \Gamma' \rhd \Delta'$. We proceed by induction on $|\rhd|$.

Base case

$$\begin{aligned}
[\![\Gamma' \Rightarrow \Delta']\!]^+ &= [\![\Gamma]\!]^- \supset [\![\Delta]\!]^+ \\
&\leq [\![\Gamma]\!]^- \supset [\![\Delta]\!]^+ \vee [\![\Delta']\!]^+ \\
&\leq [\![\Gamma']\!]^- \wedge [\![\Gamma]\!]^- \supset [\![\Delta]\!]^+ \vee [\![\Delta']\!]^+ \\
&= [\![\Gamma', \Gamma \Rightarrow \Delta, \Delta']\!]^+
\end{aligned}$$

Recursive case

$$\begin{aligned}
[\![\Gamma' \langle \mathcal{S} \rangle \Delta']\!]^+ &= \textstyle\bigwedge_{T \in \mathcal{S}} [\![T \uplus (\Gamma' \Rightarrow \Delta')]\!]^+ \\
&\leq \textstyle\bigwedge_{T \in \mathcal{S}} [\![(T \uplus (\Gamma' \Rightarrow \Delta')) \uplus (\Gamma \Rightarrow \Delta)]\!]^+ && \text{(IH)} \\
&= \textstyle\bigwedge_{T \in \mathcal{S}} [\![T \uplus ((\Gamma' \Rightarrow \Delta') \uplus (\Gamma \Rightarrow \Delta))]\!]^+ \\
&= \textstyle\bigwedge_{T \in \mathcal{S}} [\![T \uplus (\Gamma', \Gamma \Rightarrow \Delta, \Delta')]\!]^+ \\
&= [\![\Gamma', \Gamma \langle \mathcal{S} \rangle \Delta, \Delta']\!]^+
\end{aligned}$$

□

Lemma A.1.2 (Generalized contraction) $[\![S \uplus (\Rightarrow I, I)]\!]^+ \simeq [\![S \uplus (\Rightarrow I)]\!]^+$ *and* $[\![S \uplus (I, I \Rightarrow)]\!]^+ \simeq [\![S \uplus (I \Rightarrow)]\!]^+$.

Proof. Let $S = \Gamma \rhd \Delta$. We proceed by induction on $|\rhd|$.

Base case

$$\begin{aligned}
[\![\Gamma \Rightarrow I, I, \Delta]\!]^+ &= [\![\Gamma]\!]^- \supset ([\![I]\!]^+ \wedge [\![I]\!]^+) \vee [\![\Delta]\!]^+ \\
&\simeq [\![\Gamma]\!]^- \supset [\![I]\!]^+ \vee [\![\Delta]\!]^+ && \text{(Fact 8.6.3)} \\
&= [\![\Gamma \Rightarrow I, \Delta]\!]^+
\end{aligned}$$

$$\begin{aligned}
[\![\Gamma, I, I \Rightarrow \Delta]\!]^+ &= [\![\Gamma]\!]^- \wedge ([\![I]\!]^- \wedge [\![I]\!]^-) \supset [\![\Delta]\!]^+ \\
&\simeq [\![\Gamma]\!]^- \wedge [\![I]\!]^- \supset [\![\Delta]\!]^+ && \text{(Fact 8.6.3)} \\
&= [\![\Gamma, I \Rightarrow \Delta]\!]^+
\end{aligned}$$

Recursive case

$$\begin{aligned}
[\![\Gamma \langle \mathcal{S} \rangle I, I, \Delta]\!]^+ &= \textstyle\bigwedge_{T \in \mathcal{S}} [\![T \uplus (\Gamma \Rightarrow I, I, \Delta)]\!]^+ \\
&= \textstyle\bigwedge_{T \in \mathcal{S}} [\![(T \uplus (\Gamma \Rightarrow \Delta)) \uplus (\Rightarrow I, I)]\!]^+ \\
&= \textstyle\bigwedge_{T \in \mathcal{S}} [\![(T \uplus (\Gamma \Rightarrow \Delta)) \uplus (\Rightarrow I)]\!]^+ && \text{(IH)} \\
&= \textstyle\bigwedge_{T \in \mathcal{S}} [\![T \uplus (\Gamma \Rightarrow I, \Delta)]\!]^+ \\
&= [\![\Gamma \langle \mathcal{S} \rangle I, \Delta]\!]^+
\end{aligned}$$

$$
\begin{aligned}
[\![\Gamma, I, I \langle \mathcal{S} \rangle \Delta]\!]^{+} &= \textstyle\bigwedge_{T \in \mathcal{S}} [\![T \uplus (\Gamma, I, I \Rightarrow \Delta)]\!]^{+} \\
&= \textstyle\bigwedge_{T \in \mathcal{S}} [\![(T \uplus (\Gamma \Rightarrow \Delta)) \uplus (I, I \Rightarrow)]\!]^{+} \\
&= \textstyle\bigwedge_{T \in \mathcal{S}} [\![(T \uplus (\Gamma \Rightarrow \Delta)) \uplus (I \Rightarrow)]\!]^{+} && \text{(IH)} \\
&= \textstyle\bigwedge_{T \in \mathcal{S}} [\![T \uplus (\Gamma, I \Rightarrow \Delta)]\!]^{+} \\
&= [\![\Gamma, I \langle \mathcal{S} \rangle \Delta]\!]^{+}
\end{aligned}
$$

□

Lemma A.1.3 (Generalized weak distributivity)

$$[\![\Gamma \rhd \Delta]\!]^{+} \vee [\![I]\!]^{+} \leq [\![\Gamma \rhd I, \Delta]\!]^{+} \tag{A.1}$$

$$[\![\Gamma \rhd I, \Delta]\!]^{+} \leq_{\mathcal{C}} [\![\Gamma \rhd \Delta]\!]^{+} \vee [\![I]\!]^{+} \tag{A.2}$$

$$[\![\Gamma, I \rhd \Delta]\!]^{-} \leq_{\mathcal{B}} [\![I]\!]^{-} \wedge [\![\Gamma \rhd \Delta]\!]^{-} \tag{A.3}$$

$$[\![I]\!]^{-} \wedge [\![\Gamma \rhd \Delta]\!]^{-} \leq_{\mathcal{C}} [\![\Gamma, I \rhd \Delta]\!]^{-} \tag{A.4}$$

Proof. We only prove (A.1): the proof of (A.2) is the same, except that we use the converse inequality of Fact 8.6.6 that holds in Boolean algebras. (A.3) and (A.4) hold by duality from (A.1) and (A.2), i.e. for (A.3) we have

$$
\begin{aligned}
& [\![\Delta^{\dagger} \rhd^{\dagger} \Gamma^{\dagger}]\!]^{+} \vee [\![I^{\dagger}]\!]^{+} && \leq && [\![\Delta^{\dagger} \rhd^{\dagger} I^{\dagger}, \Gamma^{\dagger}]\!]^{+} && \text{(A.1)} \\
\text{iff}\quad & [\![\Delta^{\dagger} \rhd^{\dagger} I^{\dagger}, \Gamma^{\dagger}]\!]^{+\dagger} && \leq_{\mathcal{B}} && [\![\Delta^{\dagger} \rhd^{\dagger} \Gamma^{\dagger}]\!]^{+} \vee [\![I^{\dagger}]\!]^{+\dagger} && \text{(Fact 8.6.1)} \\
\text{iff}\quad & [\![\Gamma^{\dagger\dagger}, I^{\dagger\dagger} \rhd^{\dagger\dagger} \Delta^{\dagger\dagger}]\!]^{-} && \leq_{\mathcal{B}} && [\![\Gamma^{\dagger\dagger} \rhd^{\dagger\dagger} \Delta^{\dagger\dagger}]\!]^{-} \wedge [\![I^{\dagger\dagger}]\!]^{-} && \text{(Lemma 8.6.4)} \\
\text{iff}\quad & [\![\Gamma, I \rhd \Delta]\!]^{-} && \leq_{\mathcal{B}} && [\![\Gamma \rhd \Delta]\!]^{-} \wedge [\![I]\!]^{-} && \text{(Lemma 8.6.1)}
\end{aligned}
$$

We prove (A.1) by induction on $|\rhd|$.

Base case

$$
\begin{aligned}
[\![\Gamma \Rightarrow \Delta]\!]^{+} \vee [\![I]\!]^{+} &= \left([\![\Gamma]\!]^{-} \supset [\![\Delta]\!]^{+}\right) \vee [\![I]\!]^{+} \\
&\leq [\![\Gamma]\!]^{-} \supset [\![\Delta]\!]^{+} \vee [\![I]\!]^{+} && \text{(Fact 8.6.6)} \\
&= [\![\Gamma \Rightarrow I, \Delta]\!]^{+}
\end{aligned}
$$

Recursive case

$$
\begin{aligned}
[\![\Gamma \langle \mathcal{S} \rangle \Delta]\!]^{+} \vee [\![I]\!]^{+} &= \textstyle\bigwedge_{(\Gamma' \blacktriangleright \Delta') \in \mathcal{S}} [\![(\Gamma' \blacktriangleright \Delta') \uplus (\Gamma \Rightarrow \Delta)]\!]^{+} \vee [\![I]\!]^{+} \\
&= \textstyle\bigwedge_{(\Gamma' \blacktriangleright \Delta') \in \mathcal{S}} [\![\Gamma, \Gamma' \blacktriangleright \Delta', \Delta]\!]^{+} \vee [\![I]\!]^{+} \\
&\simeq_{\mathcal{L}} \textstyle\bigwedge_{(\Gamma' \blacktriangleright \Delta') \in \mathcal{S}} [\![\Gamma, \Gamma' \blacktriangleright \Delta', \Delta]\!]^{+} \vee \bigwedge_{(\Gamma' \blacktriangleright \Delta') \in \mathcal{S}} [\![I]\!]^{+} && \text{(Fact 8.6.3)} \\
&\simeq_{\mathcal{L}} \textstyle\bigwedge_{(\Gamma' \blacktriangleright \Delta') \in \mathcal{S}} \left([\![\Gamma, \Gamma' \blacktriangleright \Delta', \Delta]\!]^{+} \vee [\![I]\!]^{+}\right) && \text{(Fact 8.6.5)} \\
&\leq \textstyle\bigwedge_{(\Gamma' \blacktriangleright \Delta') \in \mathcal{S}} \left([\![\Gamma, \Gamma' \blacktriangleright I, \Delta', \Delta]\!]^{+}\right) && \text{(IH)} \\
&= \textstyle\bigwedge_{(\Gamma' \blacktriangleright \Delta') \in \mathcal{S}} [\![(\Gamma' \blacktriangleright \Delta') \uplus (\Gamma \Rightarrow I, \Delta)]\!]^{+} \\
&= [\![\Gamma \langle \mathcal{S} \rangle I, \Delta]\!]^{+}
\end{aligned}
$$

□

Lemma A.1.4 (Generalized currying)

$$[\![\Gamma, I \rhd \Delta]\!]^+ \simeq [\![I]\!]^- \supset [\![\Gamma \rhd \Delta]\!]^+ \tag{A.5}$$
$$[\![\Gamma \rhd I, \Delta]\!]^- \simeq_{\mathcal{B}} [\![\Gamma \rhd \Delta]\!]^- \subset [\![I]\!]^+ \tag{A.6}$$

Proof. We only prove (A.5), as (A.6) holds by duality as in Lemma 8.6.8.
We proceed by induction on $|\rhd|$.

Base case

$$\begin{array}{rcll} [\![\Gamma, I \Rightarrow \Delta]\!]^+ & = & [\![\Gamma]\!]^- \wedge [\![I]\!]^- \supset [\![\Delta]\!]^+ & \\ & \simeq & [\![I]\!]^- \wedge [\![\Gamma]\!]^- \supset [\![\Delta]\!]^+ & \text{(Fact 8.6.2)} \\ & \simeq & [\![I]\!]^- \supset [\![\Gamma]\!]^- \supset [\![\Delta]\!]^+ & \text{(Fact 8.6.4)} \\ & = & [\![I]\!]^- \supset [\![\Gamma \Rightarrow \Delta]\!]^+ & \end{array}$$

Recursive case

$$\begin{array}{rcll} [\![\Gamma, I \langle \mathcal{S} \rangle \Delta]\!]^+ & = & \bigwedge_{(\Gamma' \blacktriangleright \Delta') \in \mathcal{S}} [\![(\Gamma' \blacktriangleright \Delta') \uplus (\Gamma, I \Rightarrow \Delta)]\!]^+ & \\ & = & \bigwedge_{(\Gamma' \blacktriangleright \Delta') \in \mathcal{S}} [\![\Gamma', \Gamma, I \blacktriangleright \Delta, \Delta']\!]^+ & \\ & \simeq & \bigwedge_{(\Gamma' \blacktriangleright \Delta') \in \mathcal{S}} ([\![I]\!]^- \supset [\![\Gamma', \Gamma \blacktriangleright \Delta, \Delta']\!]^+) & \text{(IH)} \\ & \simeq & [\![I]\!]^- \supset \bigwedge_{(\Gamma' \blacktriangleright \Delta') \in \mathcal{S}} [\![\Gamma', \Gamma \blacktriangleright \Delta, \Delta']\!]^+ & \text{(Fact 8.6.5)} \\ & = & [\![I]\!]^- \supset \bigwedge_{(\Gamma' \blacktriangleright \Delta') \in \mathcal{S}} [\![(\Gamma' \blacktriangleright \Delta') \uplus (\Gamma \Rightarrow \Delta)]\!]^+ & \\ & = & [\![I]\!]^- \supset [\![\Gamma \langle \mathcal{S} \rangle \Delta]\!]^+ & \end{array}$$

□

Lemma A.1.5 (Local soundness) *If* $S \rightharpoonup T$ *then* $[\![T \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \leq_{\mathcal{C}} [\![S \uplus (\Gamma \Rightarrow \Delta)]\!]^+$.

Proof. We show that $S \rightharpoonup T$ implies $[\![T]\!]^+ \leq_{\mathcal{C}} [\![S]\!]^+$ by inspection of each rule of system B. That we can mix an arbitrary top-level context $\Gamma \Rightarrow \Delta$ into S and T follows from Fact 8.6.9.

i↓

$$\begin{array}{rcl} [\![\Gamma \langle\rangle \Delta]\!]^+ & = & \bigwedge_{U \in \varnothing} [\![U \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \\ & = & \top \\ & \simeq & [\![\Gamma]\!]^- \wedge A \supset A \vee [\![\Delta]\!]^+ \\ & = & [\![\Gamma, A \Rightarrow A, \Delta]\!]^+ \end{array}$$

i↑

$$\begin{array}{rcll} [\![\Gamma \langle \Rightarrow A; A \Rightarrow \rangle \Delta]\!]^+ & = & [\![\Gamma \Rightarrow A, \Delta]\!]^+ \wedge [\![\Gamma, A \Rightarrow \Delta]\!]^+ & \\ & = & ([\![\Gamma]\!]^- \supset A \vee [\![\Delta]\!]^+) \wedge ([\![\Gamma]\!]^- \wedge A \supset [\![\Delta]\!]^+) & \\ & \simeq & ([\![\Gamma]\!]^- \supset A \vee [\![\Delta]\!]^+) \wedge ([\![\Gamma]\!]^- \supset A \supset [\![\Delta]\!]^+) & \text{(Fact 8.6.4)} \\ & \simeq & [\![\Gamma]\!]^- \supset (A \vee [\![\Delta]\!]^+) \wedge (A \supset [\![\Delta]\!]^+) & \text{(Fact 8.6.5)} \\ & \simeq & [\![\Gamma]\!]^- \supset [\![\Delta]\!]^+ & \text{(Fact 8.6.7)} \\ & = & [\![\Gamma \Rightarrow \Delta]\!]^+ & \end{array}$$

w−, w+ By Lemma 8.6.6.

c−, c+ By Lemma 8.6.7.

f↑

$$
\begin{aligned}
[\![\Gamma \langle \mathcal{S}; \Gamma' \langle \mathcal{S}' \rangle \Delta' ; S \rangle \Delta]\!]^+ &= \bigwedge_{T \in \mathcal{S}} [\![T \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \wedge [\![\Gamma, \Gamma' \langle \mathcal{S}' \rangle \Delta', \Delta]\!]^+ \wedge [\![S \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \\
&\leq \bigwedge_{T \in \mathcal{S}} [\![T \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \wedge [\![\Gamma, \Gamma' \langle \mathcal{S}' \rangle \Delta', \Delta]\!]^+ \wedge [\![S \uplus (\Gamma, \Gamma' \Rightarrow \Delta', \Delta)]\!]^+ && \text{(Lemma 8.6.6)} \\
&= [\![\Gamma \langle \mathcal{S}; \Gamma' \langle \mathcal{S}' ; S \rangle \Delta' \rangle \Delta]\!]^+
\end{aligned}
$$

f−↓

$$
\begin{aligned}
[\![\Gamma \langle \Gamma', I \blacktriangleright \Delta' ; \mathcal{S} \rangle \Delta]\!]^+ &= [\![\Gamma, \Gamma', I \blacktriangleright \Delta', \Delta]\!]^+ \wedge \bigwedge_{S \in \mathcal{S}} [\![S \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \\
&= [\![\Gamma, I \langle \Gamma' \blacktriangleright \Delta' ; \mathcal{S} \rangle \Delta]\!]^+
\end{aligned}
$$

f+↓

$$
\begin{aligned}
[\![\Gamma \langle \mathcal{S}; \Gamma' \blacktriangleright I, \Delta' \rangle \Delta]\!]^+ &= \bigwedge_{S \in \mathcal{S}} [\![S \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \wedge [\![\Gamma, \Gamma' \blacktriangleright I, \Delta', \Delta]\!]^+ \\
&= [\![\Gamma \langle \mathcal{S}; \Gamma' \blacktriangleright \Delta' \rangle I, \Delta]\!]^+
\end{aligned}
$$

f−+↓ We show that $[\![\Gamma \rhd (\Gamma', I \blacktriangleright \Delta'), \Delta]\!]^+ \leq [\![\Gamma, I \rhd (\Gamma' \blacktriangleright \Delta'), \Delta]\!]^+$ by induction on $|\rhd|$.

Base case

$$
\begin{aligned}
[\![\Gamma \Rightarrow (\Gamma', I \blacktriangleright \Delta'), \Delta]\!]^+ &= [\![\Gamma]\!]^- \supset [\![\Gamma', I \blacktriangleright \Delta']\!]^+ \vee [\![\Delta]\!]^+ \\
&\simeq [\![\Gamma]\!]^- \supset \big([\![I]\!]^- \supset [\![\Gamma' \blacktriangleright \Delta']\!]^+\big) \vee [\![\Delta]\!]^+ && \text{(Lemma 8.6.9)} \\
&\leq [\![\Gamma]\!]^- \supset \big([\![I]\!]^- \supset [\![\Gamma' \blacktriangleright \Delta']\!]^+ \vee [\![\Delta]\!]^+\big) && \text{(Fact 8.6.6)} \\
&\simeq [\![\Gamma]\!]^- \wedge [\![I]\!]^- \supset [\![\Gamma' \blacktriangleright \Delta']\!]^+ \vee [\![\Delta]\!]^+ && \text{(Fact 8.6.4)} \\
&= [\![\Gamma, I \Rightarrow (\Gamma' \blacktriangleright \Delta'), \Delta]\!]^+
\end{aligned}
$$

Recursive case

$$
\begin{aligned}
[\![\Gamma \langle \mathcal{S} \rangle (\Gamma', I \blacktriangleright \Delta'), \Delta]\!]^+ &= \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![(\Gamma'' \rhd \Delta'') \uplus (\Gamma \Rightarrow (\Gamma', I \blacktriangleright \Delta'), \Delta)]\!]^+ \\
&= \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![\Gamma'', \Gamma \rhd (\Gamma', I \blacktriangleright \Delta'), \Delta, \Delta'']\!]^+ \\
&\leq \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![\Gamma'', \Gamma, I \rhd (\Gamma' \blacktriangleright \Delta'), \Delta, \Delta'']\!]^+ && \text{(IH)} \\
&= \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![(\Gamma'' \rhd \Delta'') \uplus (\Gamma, I \Rightarrow (\Gamma' \blacktriangleright \Delta'), \Delta)]\!]^+ \\
&= [\![\Gamma, I \langle \mathcal{S} \rangle (\Gamma' \blacktriangleright \Delta'), \Delta]\!]^+
\end{aligned}
$$

f+−↓ We show that $[\![\Gamma, (\Gamma' \blacktriangleright I, \Delta') \rhd \Delta]\!]^+ \leq_{\mathcal{HB}} [\![\Gamma, (\Gamma' \blacktriangleright \Delta') \rhd I, \Delta]\!]^+$ by induction on $|\rhd|$.

Base case

$$
\begin{aligned}
[\![\Gamma, (\Gamma' \blacktriangleright I, \Delta') \Rightarrow \Delta]\!]^+ &= [\![\Gamma]\!]^- \wedge [\![\Gamma' \blacktriangleright I, \Delta']\!]^- \supset [\![\Delta]\!]^+ \\
&\simeq_{\mathcal{HB}} [\![\Gamma]\!]^- \wedge \big([\![\Gamma' \blacktriangleright \Delta']\!]^- \subset [\![I]\!]^+\big) \supset [\![\Delta]\!]^+ && \text{(Lemma 8.6.9)} \\
&\leq_{\mathcal{HB}} \big([\![\Gamma]\!]^- \wedge [\![\Gamma' \blacktriangleright \Delta']\!]^- \subset [\![I]\!]^+\big) \supset [\![\Delta]\!]^+ && \text{(Fact 8.6.6)} \\
&\leq_{\mathcal{HB}} [\![\Gamma]\!]^- \wedge [\![\Gamma' \blacktriangleright \Delta']\!]^- \supset [\![I]\!]^+ \vee [\![\Delta]\!]^+ && \text{(Fact 8.6.8)} \\
&= [\![\Gamma, (\Gamma' \blacktriangleright \Delta') \Rightarrow I, \Delta]\!]^+
\end{aligned}
$$

Recursive case

$$\begin{array}{rcll} [\![\Gamma, (\Gamma' \blacktriangleright I, \Delta') \, \langle \mathcal{S} \rangle \, \Delta]\!]^+ & = & \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![(\Gamma'' \rhd \Delta'') \uplus (\Gamma, (\Gamma' \blacktriangleright I, \Delta') \Rightarrow \Delta)]\!]^+ & \\ & = & \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![\Gamma'', \Gamma, (\Gamma' \blacktriangleright I, \Delta') \rhd \Delta, \Delta'']\!]^+ & \\ & \leq_{\mathcal{HB}} & \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![\Gamma'', \Gamma, (\Gamma' \blacktriangleright \Delta') \rhd I, \Delta, \Delta'']\!]^+ & \text{(IH)} \\ & = & \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![(\Gamma'' \rhd \Delta'') \uplus (\Gamma, (\Gamma' \blacktriangleright \Delta') \Rightarrow I, \Delta)]\!]^+ & \\ & = & [\![\Gamma, (\Gamma' \blacktriangleright \Delta') \, \langle \mathcal{S} \rangle \, I, \Delta]\!]^+ & \end{array}$$

f++↑ We show that $[\![\Gamma \rhd (\Gamma' \blacktriangleright \Delta'), I, \Delta]\!]^+ \leq [\![\Gamma \rhd (\Gamma' \blacktriangleright I, \Delta'), \Delta]\!]^+$ by induction on $|\rhd|$.

Base case

$$\begin{array}{rcll} [\![\Gamma \Rightarrow (\Gamma' \blacktriangleright \Delta'), I, \Delta]\!]^+ & = & [\![\Gamma]\!]^- \supset [\![\Gamma' \blacktriangleright \Delta']\!]^+ \vee [\![I]\!]^+ \vee [\![\Delta]\!]^+ & \\ & \leq & [\![\Gamma]\!]^- \supset [\![\Gamma' \blacktriangleright I, \Delta']\!]^+ \vee [\![\Delta]\!]^+ & \text{(Lemma 8.6.8)} \\ & = & [\![\Gamma \Rightarrow (\Gamma' \blacktriangleright I, \Delta'), \Delta]\!]^+ & \end{array}$$

Recursive case

$$\begin{array}{rcll} [\![\Gamma \, \langle \mathcal{S} \rangle \, (\Gamma' \blacktriangleright \Delta'), I, \Delta]\!]^+ & = & \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![(\Gamma'' \rhd \Delta'') \uplus (\Gamma \Rightarrow (\Gamma' \blacktriangleright \Delta'), I, \Delta)]\!]^+ & \\ & = & \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![\Gamma'', \Gamma \rhd (\Gamma' \blacktriangleright \Delta'), I, \Delta, \Delta'']\!]^+ & \\ & \leq & \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![\Gamma'', \Gamma \rhd (\Gamma' \blacktriangleright I, \Delta'), \Delta, \Delta'']\!]^+ & \text{(IH)} \\ & = & \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![(\Gamma'' \rhd \Delta'') \uplus (\Gamma \Rightarrow (\Gamma' \blacktriangleright I, \Delta'), \Delta)]\!]^+ & \\ & = & [\![\Gamma \, \langle \mathcal{S} \rangle \, (\Gamma' \blacktriangleright I, \Delta'), \Delta]\!]^+ & \end{array}$$

f−−↑ We show that $[\![\Gamma, I, (\Gamma' \blacktriangleright \Delta') \rhd \Delta]\!]^+ \leq_{\mathcal{HB}} [\![\Gamma, (\Gamma', I \blacktriangleright \Delta') \rhd \Delta]\!]^+$ by induction on $|\rhd|$.

Base case

$$\begin{array}{rcll} [\![\Gamma, I, (\Gamma' \blacktriangleright \Delta') \Rightarrow \Delta]\!]^+ & = & [\![\Gamma]\!]^- \wedge [\![I]\!]^- \wedge [\![\Gamma' \blacktriangleright \Delta']\!]^- \supset [\![\Delta]\!]^+ & \\ & \leq_{\mathcal{HB}} & [\![\Gamma]\!]^- \wedge [\![\Gamma', I \blacktriangleright \Delta']\!]^- \supset [\![\Delta]\!]^+ & \text{(Lemma 8.6.8)} \\ & = & [\![\Gamma, (\Gamma', I \blacktriangleright \Delta') \Rightarrow \Delta]\!]^+ & \end{array}$$

Recursive case

$$\begin{array}{rcll} [\![\Gamma, I, (\Gamma' \blacktriangleright \Delta') \, \langle \mathcal{S} \rangle \, \Delta]\!]^+ & = & \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![(\Gamma'' \rhd \Delta'') \uplus (\Gamma, I, (\Gamma' \blacktriangleright \Delta') \Rightarrow \Delta)]\!]^+ & \\ & = & \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![\Gamma'', \Gamma, I, (\Gamma' \blacktriangleright \Delta') \rhd \Delta, \Delta'']\!]^+ & \\ & \leq_{\mathcal{HB}} & \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![\Gamma'', \Gamma, (\Gamma', I \blacktriangleright \Delta') \rhd \Delta, \Delta'']\!]^+ & \text{(IH)} \\ & = & \bigwedge_{(\Gamma'' \rhd \Delta'') \in \mathcal{S}} [\![(\Gamma'' \rhd \Delta'') \uplus (\Gamma, (\Gamma', I \blacktriangleright \Delta') \Rightarrow \Delta)]\!]^+ & \\ & = & [\![\Gamma, (\Gamma', I \blacktriangleright \Delta') \, \langle \mathcal{S} \rangle \, \Delta]\!]^+ & \end{array}$$

f−+↑, f+−↑ Converse of f−+↓ (resp. f+−↓), using the converse inequality of Fact 8.6.6 which only holds in Boolean algebras.

f++↓, f−−↓ Converse of f++↑ (resp. f−−↑), using the converse inequality of Lemma 8.6.8 which only holds in Boolean algebras.

p

$$
\begin{aligned}
[\![\Gamma \langle \mathcal{S} \rangle \Delta]\!]^+ &= \textstyle\bigwedge_{S \in \mathcal{S}} [\![S \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \\
&\simeq_{\mathcal{L}} \textstyle\bigwedge_{S \in \mathcal{S}} [\![S \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \wedge \top \\
&= \textstyle\bigwedge_{S \in \mathcal{S}} [\![S \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \wedge [\![\Gamma \langle\rangle \Delta]\!]^+ \\
&= [\![\Gamma \langle \mathcal{S}; \langle\rangle \rangle \Delta]\!]^+
\end{aligned}
$$

p−

$$
\begin{aligned}
[\![\Gamma \langle\rangle \Delta]\!]^+ &= \top \\
&\simeq [\![\Gamma]\!]^- \wedge \bot \supset [\![\Delta]\!]^+ \\
&= [\![\Gamma, (\langle\rangle) \Rightarrow \Delta]\!]^+
\end{aligned}
$$

p+

$$
\begin{aligned}
[\![\Gamma \langle\rangle \Delta]\!]^+ &= \top \\
&\simeq [\![\Gamma]\!]^- \supset \top \vee [\![\Delta]\!]^+ \\
&= [\![\Gamma \Rightarrow (\langle\rangle), \Delta]\!]^+
\end{aligned}
$$

a

$$
\begin{aligned}
[\![\Gamma \langle S \rangle \Delta]\!]^+ &= [\![\, \langle S \rangle \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \\
&= [\![\Gamma \langle\, \langle S \rangle \,\rangle \Delta]\!]^+
\end{aligned}
$$

a−, a+ We only do the proof for a−, the proof for a+ is symmetric. We show that $[\![\Gamma, S \rhd \Delta]\!]^+ = [\![\Gamma, (\langle S \rangle) \rhd \Delta]\!]^+$ by induction on $|\rhd|$.

Base case

$$
\begin{aligned}
[\![\Gamma, S \Rightarrow \Delta]\!]^+ &= [\![\Gamma]\!]^- \wedge [\![S]\!]^- \supset [\![\Delta]\!]^+ \\
&= [\![\Gamma]\!]^- \wedge [\![\, \langle S \rangle \,]\!]^- \supset [\![\Delta]\!]^+ \\
&= [\![\Gamma, (\langle S \rangle) \Rightarrow \Delta]\!]^+
\end{aligned}
$$

Recursive case

$$
\begin{aligned}
[\![\Gamma, S \langle \mathcal{S} \rangle \Delta]\!]^+ &= \textstyle\bigwedge_{T \in \mathcal{S}} [\![T \uplus (\Gamma, S \Rightarrow \Delta)]\!]^+ \\
&= \textstyle\bigwedge_{T \in \mathcal{S}} [\![T \uplus (\Gamma, (\langle S \rangle) \Rightarrow \Delta)]\!]^+ \quad \text{(IH)} \\
&= [\![\Gamma, (\langle S \rangle) \langle \mathcal{S} \rangle \Delta]\!]^+
\end{aligned}
$$

⊤−, ⊥+ We only do the proof for ⊤−, the proof for ⊥+ is symmetric. We show that $[\![\Gamma \rhd \Delta]\!]^+ \simeq [\![\Gamma, \top \rhd \Delta]\!]^+$ by induction on $|\rhd|$.

Base case

$$
\begin{aligned}
[\![\Gamma \Rightarrow \Delta]\!]^+ &= [\![\Gamma]\!]^- \supset [\![\Delta]\!]^+ \\
&\simeq [\![\Gamma]\!]^- \wedge \top \supset [\![\Delta]\!]^+ \\
&= [\![\Gamma, \top \Rightarrow \Delta]\!]^+
\end{aligned}
$$

Recursive case

$$
\begin{aligned}
[\![\Gamma \langle \mathcal{S} \rangle \Delta]\!]^+ &= \textstyle\bigwedge_{S \in \mathcal{S}} [\![S \uplus (\Gamma \Rightarrow \Delta)]\!]^+ \\
&\simeq \textstyle\bigwedge_{S \in \mathcal{S}} [\![S \uplus (\Gamma, \top \Rightarrow \Delta)]\!]^+ \quad \text{(IH)} \\
&= [\![\Gamma, \top \langle \mathcal{S} \rangle \Delta]\!]^+
\end{aligned}
$$

$\top+$

$$\begin{aligned} [\![\Gamma \langle\rangle \Delta]\!]^+ &= \top \\ &\simeq [\![\Gamma]\!]^- \supset \top \vee [\![\Delta]\!]^+ \\ &= [\![\Gamma \Rightarrow \top, \Delta]\!]^+ \end{aligned}$$

$\bot-$

$$\begin{aligned} [\![\Gamma \langle\rangle \Delta]\!]^+ &= \top \\ &\simeq [\![\Gamma]\!]^- \wedge \bot \supset [\![\Delta]\!]^+ \\ &= [\![\Gamma, \bot \Rightarrow \Delta]\!]^+ \end{aligned}$$

$\wedge-$, $\vee+$ We only do the proof for $\wedge-$, the proof for $\vee+$ is symmetric. We show that $[\![\Gamma, A, B \rhd \Delta]\!]^+ = [\![\Gamma, A \wedge B \rhd \Delta]\!]^+$ by induction on $|\rhd|$.

Base case

$$\begin{aligned} [\![\Gamma, A, B \Rightarrow \Delta]\!]^+ &= [\![\Gamma]\!]^- \wedge A \wedge B \supset [\![\Delta]\!]^+ \\ &= [\![\Gamma, A \wedge B \Rightarrow \Delta]\!]^+ \end{aligned}$$

Recursive case

$$\begin{aligned} [\![\Gamma, A, B \langle \mathcal{S} \rangle \Delta]\!]^+ &= \bigwedge_{S \in \mathcal{S}} [\![S \uplus (\Gamma, A, B \Rightarrow \Delta)]\!]^+ \\ &= \bigwedge_{S \in \mathcal{S}} [\![S \uplus (\Gamma, A \wedge B \Rightarrow \Delta)]\!]^+ && \text{(IH)} \\ &= [\![\Gamma, A \wedge B \langle \mathcal{S} \rangle \Delta]\!]^+ \end{aligned}$$

$\wedge+$

$$\begin{aligned} [\![\Gamma \langle \Rightarrow A; \Rightarrow B \rangle \Delta]\!]^+ &= [\![\Gamma \Rightarrow A, \Delta]\!]^+ \wedge [\![\Gamma \Rightarrow B, \Delta]\!]^+ \\ &= \left([\![\Gamma]\!]^- \supset A \vee [\![\Delta]\!]^+\right) \wedge \left([\![\Gamma]\!]^- \supset B \vee [\![\Delta]\!]^+\right) \\ &\simeq [\![\Gamma]\!]^- \supset \left(A \vee [\![\Delta]\!]^+\right) \wedge \left(B \vee [\![\Delta]\!]^+\right) && \text{(Fact 8.6.5)} \\ &\simeq [\![\Gamma]\!]^- \supset \left([\![\Delta]\!]^+ \vee A\right) \wedge \left([\![\Delta]\!]^+ \vee B\right) && \text{(Fact 8.6.2)} \\ &\simeq [\![\Gamma]\!]^- \supset [\![\Delta]\!]^+ \vee (A \wedge B) && \text{(Fact 8.6.5)} \\ &\simeq [\![\Gamma]\!]^- \supset (A \wedge B) \vee [\![\Delta]\!]^+ && \text{(Fact 8.6.2)} \\ &= [\![\Gamma \Rightarrow A \wedge B, \Delta]\!]^+ \end{aligned}$$

$\vee-$

$$\begin{aligned} [\![\Gamma \langle A \Rightarrow; B \Rightarrow \rangle \Delta]\!]^+ &= [\![\Gamma, A \Rightarrow \Delta]\!]^+ \wedge [\![\Gamma, B \Rightarrow \Delta]\!]^+ \\ &= \left([\![\Gamma]\!]^- \wedge A \supset [\![\Delta]\!]^+\right) \wedge \left([\![\Gamma]\!]^- \wedge B \supset [\![\Delta]\!]^+\right) \\ &\simeq ([\![\Gamma]\!]^- \wedge A) \vee ([\![\Gamma]\!]^- \wedge B) \supset [\![\Delta]\!]^+ && \text{(Fact 8.6.5)} \\ &\simeq [\![\Gamma]\!]^- \wedge (A \vee B) \supset [\![\Delta]\!]^+ && \text{(Fact 8.6.5)} \\ &= [\![\Gamma, A \vee B \Rightarrow \Delta]\!]^+ \end{aligned}$$

$\supset+$, $\subset-$ We only do the proof for $\supset+$, the proof for $\subset-$ is symmetric. We show that $[\![\Gamma \rhd (A \Rightarrow B), \Delta]\!]^+ = [\![\Gamma \rhd A \supset B, \Delta]\!]^+$ by induction on $|\rhd|$.

Base case

$$\begin{aligned} [\![\Gamma \Rightarrow (A \Rightarrow B), \Delta]\!]^+ &= [\![\Gamma]\!]^- \supset [\![A \Rightarrow B]\!]^+ \vee [\![\Delta]\!]^+ \\ &= [\![\Gamma]\!]^- \supset (A \supset B) \vee [\![\Delta]\!]^+ \\ &= [\![\Gamma \Rightarrow A \supset B, \Delta]\!]^+ \end{aligned}$$

Recursive case

$$\begin{aligned}
[\![\Gamma \langle \mathcal{S} \rangle (A \Rightarrow B), \Delta]\!]^{+} &= \textstyle\bigwedge_{S \in \mathcal{S}} [\![S \uplus (\Gamma \Rightarrow (A \Rightarrow B), \Delta)]\!]^{+} \\
&= \textstyle\bigwedge_{S \in \mathcal{S}} [\![S \uplus (\Gamma \Rightarrow A \supset B, \Delta)]\!]^{+} && \text{(IH)} \\
&= [\![\Gamma \langle \mathcal{S} \rangle A \supset B, \Delta]\!]^{+}
\end{aligned}$$

$\supset-$

$$\begin{aligned}
[\![\Gamma \langle \Rightarrow A; B \Rightarrow \rangle \Delta]\!]^{+} &= [\![\Gamma \Rightarrow A, \Delta]\!]^{+} \wedge [\![\Gamma, B \Rightarrow \Delta]\!]^{+} \\
&= \left([\![\Gamma]\!]^{-} \supset A \vee [\![\Delta]\!]^{+}\right) \wedge \left([\![\Gamma]\!]^{-} \wedge B \supset [\![\Delta]\!]^{+}\right) \\
&\simeq \left([\![\Gamma]\!]^{-} \supset A \vee [\![\Delta]\!]^{+}\right) \wedge \left([\![\Gamma]\!]^{-} \supset B \supset [\![\Delta]\!]^{+}\right) && \text{(Fact 8.6.4)} \\
&\simeq [\![\Gamma]\!]^{-} \supset \left(A \vee [\![\Delta]\!]^{+}\right) \wedge \left(B \supset [\![\Delta]\!]^{+}\right) && \text{(Fact 8.6.5)} \\
&\leq [\![\Gamma]\!]^{-} \supset (A \supset B) \supset [\![\Delta]\!]^{+} \vee [\![\Delta]\!]^{+} && \text{(Fact 8.6.7)} \\
&\simeq [\![\Gamma]\!]^{-} \supset (A \supset B) \supset [\![\Delta]\!]^{+} && \text{(Fact 8.6.3)} \\
&\simeq [\![\Gamma]\!]^{-} \wedge (A \supset B) \supset [\![\Delta]\!]^{+} && \text{(Fact 8.6.4)} \\
&= [\![\Gamma, A \supset B \Rightarrow \Delta]\!]^{+}
\end{aligned}$$

$\subset+$

$$\begin{aligned}
[\![\Gamma \langle \Rightarrow A; B \Rightarrow \rangle \Delta]\!]^{+} &= [\![\Gamma \Rightarrow A, \Delta]\!]^{+} \wedge [\![\Gamma, B \Rightarrow \Delta]\!]^{+} \\
&= \left([\![\Gamma]\!]^{-} \supset A \vee [\![\Delta]\!]^{+}\right) \wedge \left([\![\Gamma]\!]^{-} \wedge B \supset [\![\Delta]\!]^{+}\right) \\
&\simeq \left([\![\Gamma]\!]^{-} \supset A \vee [\![\Delta]\!]^{+}\right) \wedge \left([\![\Gamma]\!]^{-} \supset B \supset [\![\Delta]\!]^{+}\right) && \text{(Fact 8.6.4)} \\
&\simeq [\![\Gamma]\!]^{-} \supset \left(A \vee [\![\Delta]\!]^{+}\right) \wedge \left(B \supset [\![\Delta]\!]^{+}\right) && \text{(Fact 8.6.5)} \\
&\leq [\![\Gamma]\!]^{-} \supset (A \subset B) \vee [\![\Delta]\!]^{+} \vee [\![\Delta]\!]^{+} && \text{(Fact 8.6.7)} \\
&\simeq [\![\Gamma]\!]^{-} \supset (A \subset B) \vee [\![\Delta]\!]^{+} && \text{(Fact 8.6.3)} \\
&= [\![\Gamma \Rightarrow A \subset B, \Delta]\!]^{+}
\end{aligned}$$

□

A.2. Completeness

In the following proofs, we will denote a sequence of applications of a set of rules by a double inference line, and the use of a derivation obtained by induction hypothesis by a dotted line.

Lemma A.2.1 (Simulation of DBiInt) *If* $X \vdash_{\mathsf{DBiInt}} Y$ *then* $X \vdash_{\mathsf{B}_{\mathcal{HB}} \setminus \{\mathsf{i}\uparrow\}} Y$.

Proof. By induction on the derivation of $X \vdash_{\mathsf{DBiInt}} Y$.

$$\dfrac{}{X, A \Rightarrow A, Y}\,\mathsf{id} \quad \mapsto \quad \dfrac{\dfrac{\langle\rangle}{X \langle\rangle Y}\,\mathsf{w-},\mathsf{w+}}{X, A \Rightarrow A, Y}\,\mathsf{i}\downarrow$$

$$\dfrac{\begin{array}{c}\vdots\,\pi_1\\ X, A, (X', A \Rightarrow Y') \Rightarrow Y\end{array}}{X, (X', A \Rightarrow Y') \Rightarrow Y}\Rightarrow_{\mathsf{L1}} \quad\mapsto\quad \dfrac{\dfrac{\dfrac{\langle\rangle}{X, A, (X', A \Rightarrow Y') \Rightarrow Y}\pi_1}{X, (X', A, A \Rightarrow Y') \Rightarrow Y}\mathsf{f}{-}{-}{\uparrow}}{X, (X', A \Rightarrow Y') \Rightarrow Y}\mathsf{c}{-}$$

$$\dfrac{\begin{array}{c}\vdots\,\pi_1\\ X \Rightarrow (X' \Rightarrow A, Y'), A, Y\end{array}}{X \Rightarrow (X' \Rightarrow A, Y'), Y}\Rightarrow_{\mathsf{R1}} \quad\mapsto\quad \dfrac{\dfrac{\dfrac{\langle\rangle}{X \Rightarrow (X' \Rightarrow A, Y'), A, Y}\pi_1}{X \Rightarrow (X' \Rightarrow A, A, Y'), Y}\mathsf{f}{+}{+}{\uparrow}}{X \Rightarrow (X' \Rightarrow A, Y'), Y}\mathsf{c}{+}$$

$$\dfrac{\begin{array}{c}\vdots\,\pi_1\\ X, A \Rightarrow (X', A \Rightarrow Y'), Y\end{array}}{X, A \Rightarrow (X' \Rightarrow Y'), Y}\Rightarrow_{\mathsf{L2}} \quad\mapsto\quad \dfrac{\dfrac{\dfrac{\langle\rangle}{X, A \Rightarrow (X', A \Rightarrow Y'), Y}\pi_1}{X, A, A \Rightarrow (X' \Rightarrow Y'), Y}\mathsf{f}{-}{+}{\downarrow}}{X, A \Rightarrow (X' \Rightarrow Y'), Y}\mathsf{c}{-}$$

$$\dfrac{X, (X' \Rightarrow A, Y') \Rightarrow A, Y}{X, (X' \Rightarrow Y') \Rightarrow A, Y}\Rightarrow_{\mathsf{R2}} \quad\mapsto\quad \dfrac{\dfrac{\dfrac{\langle\rangle}{X, (X' \Rightarrow A, Y') \Rightarrow A, Y}\pi_1}{X, (X' \Rightarrow Y') \Rightarrow A, A, Y}\mathsf{f}{+}{-}{\downarrow}}{X, (X' \Rightarrow Y') \Rightarrow A, Y}\mathsf{c}{+}$$

$$\dfrac{}{X, \bot \Rightarrow Y}\bot_{\mathsf{L}} \quad\mapsto\quad \dfrac{\dfrac{\langle\rangle}{X \langle\rangle Y}\mathsf{w}{-},\mathsf{w}{+}}{X, \bot \Rightarrow Y}\bot{-}$$

$$\dfrac{}{X \Rightarrow \top, Y}\top_{\mathsf{R}} \quad\mapsto\quad \dfrac{\dfrac{\langle\rangle}{X \langle\rangle Y}\mathsf{w}{-},\mathsf{w}{+}}{X \Rightarrow \top, Y}\top{+}$$

$$\dfrac{\begin{array}{c}\vdots\,\pi_1\\ X, A \wedge B, A, B \Rightarrow Y\end{array}}{X, A \wedge B \Rightarrow Y}\wedge_{\mathsf{L}} \quad\mapsto\quad \dfrac{\dfrac{\dfrac{\langle\rangle}{X, A \wedge B, A, B \Rightarrow Y}\pi_1}{X, A \wedge B, A \wedge B \Rightarrow Y}\wedge{-}}{X, A \wedge B \Rightarrow Y}\mathsf{c}{-}$$

$$\dfrac{\begin{array}{c}\vdots\,\pi_1\\ X \Rightarrow A, B, A \vee B, Y\end{array}}{X \Rightarrow A \vee B, Y}\vee_{\mathsf{R}} \quad\mapsto\quad \dfrac{\dfrac{\dfrac{\langle\rangle}{X \Rightarrow A, B, A \vee B \Rightarrow Y}\pi_1}{X \Rightarrow A \vee B, A \vee B, Y}\vee{+}}{X \Rightarrow A \vee B, Y}\mathsf{c}{+}$$

$$\dfrac{\overset{\vdots\;\pi_1}{X \Rightarrow A, A \wedge B, Y} \quad \overset{\vdots\;\pi_2}{X \Rightarrow B, A \wedge B, Y}}{X \Rightarrow A \wedge B, Y}\;\wedge_{\mathsf{R}} \quad\mapsto\quad \begin{array}{cl}
\langle\rangle & \\ \hline\hline
\langle\,\langle\rangle;\langle\rangle\,\rangle & \mathsf{p} \\
\vdots & \pi_2 \\
\langle\,\langle\rangle; X \Rightarrow B, A \wedge B, Y\rangle & \\
\vdots & \pi_1 \\
\langle X \Rightarrow A, A \wedge B, Y; X \Rightarrow B, A \wedge B, Y\rangle & \\ \hline\hline
X, X\,\langle\, \Rightarrow A; \Rightarrow B\rangle\; A \wedge B, A \wedge B, Y, Y & \mathsf{f}{-}\!\downarrow,\mathsf{f}{+}\!\downarrow \\ \hline\hline
X\,\langle\, \Rightarrow A; \Rightarrow B\rangle\; A \wedge B, Y & \mathsf{c}{-},\mathsf{c}{+} \\ \hline
X \Rightarrow A \wedge B, A \wedge B, Y & \wedge{+} \\ \hline
X \Rightarrow A \wedge B, Y & \mathsf{c}{+}
\end{array}$$

$$\dfrac{\overset{\vdots\;\pi_1}{X, A \vee B, A \Rightarrow Y} \quad \overset{\vdots\;\pi_2}{X, A \vee B, B \Rightarrow Y}}{X, A \vee B \Rightarrow Y}\;\vee_{\mathsf{L}} \quad\mapsto\quad \begin{array}{cl}
\langle\rangle & \\ \hline\hline
\langle\,\langle\rangle;\langle\rangle\,\rangle & \mathsf{p} \\
\vdots & \pi_2 \\
\langle\,\langle\rangle; X, A \vee B, B \Rightarrow Y\rangle & \\
\vdots & \pi_1 \\
\langle X, A \vee B, A \Rightarrow Y; X, A \vee B, B \Rightarrow Y\rangle & \\ \hline\hline
X, X, A \vee B, A \vee B\,\langle A \Rightarrow; B \Rightarrow\,\rangle\; Y, Y & \mathsf{f}{-}\!\downarrow,\mathsf{f}{+}\!\downarrow \\ \hline\hline
X, A \vee B\,\langle A \Rightarrow; B \Rightarrow\,\rangle\; Y & \mathsf{c}{-},\mathsf{c}{+} \\ \hline
X, A \vee B, A \vee B \Rightarrow Y & \vee{-} \\ \hline
X, A \vee B \Rightarrow Y & \mathsf{c}{-}
\end{array}$$

$$\dfrac{\overset{\vdots\;\pi_1}{X \Rightarrow (A \Rightarrow B), A \supset B, Y}}{X \Rightarrow A \supset B, Y}\;\supset_{\mathsf{R}} \quad\mapsto\quad \begin{array}{cl}
\langle\rangle & \\
\vdots & \pi_1 \\
X \Rightarrow (A \Rightarrow B), A \supset B, Y & \\ \hline
X \Rightarrow A \supset B, A \supset B, Y & \supset{+} \\ \hline
X \Rightarrow A \supset B, Y & \mathsf{c}{+}
\end{array}$$

$$\dfrac{\overset{\vdots\;\pi_1}{X, A \subset B, (A \Rightarrow B) \Rightarrow Y}}{X, A \subset B \Rightarrow Y}\;\subset_{\mathsf{L}} \quad\mapsto\quad \begin{array}{cl}
\langle\rangle & \\
\vdots & \pi_1 \\
X, A \subset B, (A \Rightarrow B) \Rightarrow Y & \\ \hline
X, A \subset B, A \subset B \Rightarrow Y & \subset{-} \\ \hline
X, A \subset B \Rightarrow Y & \mathsf{c}{-}
\end{array}$$

$$\dfrac{\overset{\vdots\;\pi_1}{X, A \supset B \Rightarrow A, Y} \quad \overset{\vdots\;\pi_2}{X, A \supset B, B \Rightarrow Y}}{X, A \supset B \Rightarrow Y}\;\supset_{\mathsf{L}} \quad\mapsto\quad \begin{array}{cl}
\langle\rangle & \\ \hline\hline
\langle\,\langle\rangle;\langle\rangle\,\rangle & \mathsf{p} \\
\vdots & \pi_2 \\
\langle\,\langle\rangle; X, A \supset B, B \Rightarrow Y\rangle & \\
\vdots & \pi_1 \\
\langle X, A \supset B \Rightarrow A, Y; X, A \supset B, B \Rightarrow Y\rangle & \\ \hline\hline
X, X, A \supset B, A \supset B\,\langle\, \Rightarrow A; B \Rightarrow\,\rangle\; Y, Y & \mathsf{f}{-}\!\downarrow,\mathsf{f}{+}\!\downarrow \\ \hline\hline
X, A \supset B\,\langle\, \Rightarrow A; B \Rightarrow\,\rangle\; Y & \mathsf{c}{-},\mathsf{c}{+} \\ \hline
X, A \supset B, A \supset B \Rightarrow Y & \supset{-} \\ \hline
X, A \supset B \Rightarrow Y & \mathsf{c}{-}
\end{array}$$

$$
\dfrac{\begin{array}{c}\vdots\;\pi_1\\ X \Rightarrow A, A \subset B, Y\end{array} \quad \begin{array}{c}\vdots\;\pi_2\\ X, B \Rightarrow A \subset B, Y\end{array}}{X \Rightarrow A \subset B, Y}\,\subset_{\mathsf{R}}
\quad\mapsto\quad
\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\dfrac{\langle\rangle}{\langle\,\langle\rangle\,;\,\langle\rangle\,\rangle}\,\mathsf{p}\\ \cdots\cdots\;\pi_2\\ \langle\,\langle\rangle\,;\, X, B \Rightarrow A \subset B, Y\rangle\\ \cdots\cdots\;\pi_1\\ \langle X \Rightarrow A, A \subset B, Y;\, X, B \Rightarrow A \subset B, Y\rangle\end{array}}{X, X\,\langle \Rightarrow A;\, B \Rightarrow\,\rangle\, A \subset B, A \subset B, Y, Y}\,\mathsf{f}{-}{\downarrow},\mathsf{f}{+}{\downarrow}}{X\,\langle \Rightarrow A;\, B \Rightarrow\,\rangle\, A \subset B, Y}\,\mathsf{c}{-},\mathsf{c}{+}}{X \Rightarrow A \subset B, A \subset B, Y}\,\subset{+}}{X \Rightarrow A \subset B, Y}\,\mathsf{c}{+}
$$

□

Lemma A.2.2 (Simulation of G3cp) *If* $\Gamma \mid_{\overline{\text{G3cp}}} \Delta$*, then* $\Gamma \mid_{\overline{\mathsf{B}_{\mathcal{H}} \cup \{\mathsf{f}{+}{+}{\downarrow}\} \setminus \{\mathsf{i}{\uparrow}\}}} \Delta$.

Proof. By induction on the G3cp derivation.

$$
\dfrac{}{a, \Gamma \Rightarrow \Delta, a}
\quad\mapsto\quad
\dfrac{\dfrac{\langle\rangle}{\Gamma\,\langle\rangle\,\Delta}\,\mathsf{w}{-},\mathsf{w}{+}}{a, \Gamma \Rightarrow \Delta, a}\,\mathsf{i}{\downarrow}
$$

$$
\dfrac{}{\bot, \Gamma \Rightarrow \Delta}
\quad\mapsto\quad
\dfrac{\dfrac{\langle\rangle}{\Gamma\,\langle\rangle\,\Delta}\,\mathsf{w}{-},\mathsf{w}{+}}{\bot, \Gamma \Rightarrow \Delta}\,\bot{-}
$$

$$
\dfrac{\begin{array}{c}\vdots\;\pi_1\\ A, B, \Gamma \Rightarrow \Delta\end{array}}{A \wedge B, \Gamma \Rightarrow \Delta}\,L\wedge
\quad\mapsto\quad
\dfrac{\begin{array}{c}\langle\rangle\\ \cdots\cdots\;\pi_1\\ A, B, \Gamma \Rightarrow \Delta\end{array}}{A \wedge B, \Gamma \Rightarrow \Delta}\,\wedge{-}
$$

$$
\dfrac{\begin{array}{c}\vdots\;\pi_1\\ \Gamma \Rightarrow \Delta, A, B\end{array}}{\Gamma \Rightarrow \Delta, A \vee B}\,R\vee
\quad\mapsto\quad
\dfrac{\begin{array}{c}\langle\rangle\\ \cdots\cdots\;\pi_1\\ \Gamma \Rightarrow \Delta, A, B\end{array}}{\Gamma \Rightarrow \Delta, A \vee B}\,\vee{+}
$$

$$
\dfrac{\begin{array}{c}\vdots\;\pi_1\\ \Gamma \Rightarrow \Delta, A\end{array} \quad \begin{array}{c}\vdots\;\pi_2\\ \Gamma \Rightarrow \Delta, B\end{array}}{\Gamma \Rightarrow \Delta, A \wedge B}\,R\wedge
\quad\mapsto\quad
\dfrac{\dfrac{\dfrac{\begin{array}{c}\dfrac{\langle\rangle}{\langle\,\langle\rangle\,;\,\langle\rangle\,\rangle}\,\mathsf{p}\\ \cdots\cdots\;\pi_1, \pi_2\\ \langle \Gamma \Rightarrow \Delta, A;\, \Gamma \Rightarrow \Delta, B\rangle\end{array}}{\Gamma, \Gamma\,\langle \Rightarrow A;\, \Rightarrow B\rangle\, \Delta, \Delta}\,\mathsf{f}{-}{\downarrow},\mathsf{f}{+}{\downarrow}}{\Gamma\,\langle \Rightarrow A;\, \Rightarrow B\rangle\, \Delta}\,\mathsf{c}{-},\mathsf{c}{+}}{\Gamma \Rightarrow \Delta, A \wedge B}\,\wedge{+}
$$

$$
\dfrac{\begin{array}{c}\vdots\ \pi_1\\ A,\Gamma \Rightarrow \Delta\end{array}\quad \begin{array}{c}\vdots\ \pi_2\\ B,\Gamma \Rightarrow \Delta\end{array}}{A \vee B,\Gamma \Rightarrow \Delta}\,L\vee
\quad\mapsto\quad
\begin{array}{cl}
\langle\rangle & \\ \hline\hline
\langle\ \langle\rangle ; \langle\rangle\ \rangle & \mathsf{p} \\
\cdots\cdots\cdots\cdots\cdots\cdots & \pi_1,\pi_2 \\
\langle A,\Gamma \Rightarrow \Delta ;\ B,\Gamma \Rightarrow \Delta\rangle & \\ \hline\hline
\Gamma,\Gamma\ \langle A \Rightarrow ;\ B \Rightarrow\ \rangle\ \Delta,\Delta & \mathsf{f}{-}{\downarrow},\mathsf{f}{+}{\downarrow} \\ \hline\hline
\Gamma\ \langle A \Rightarrow ;\ B \Rightarrow\ \rangle\ \Delta & \mathsf{c}{-},\mathsf{c}{+} \\ \hline
A \vee B,\Gamma \Rightarrow \Delta & \vee{-}
\end{array}
$$

$$
\dfrac{\begin{array}{c}\vdots\ \pi_1\\ \Gamma \Rightarrow \Delta, A\end{array}\quad \begin{array}{c}\vdots\ \pi_2\\ B,\Gamma \Rightarrow \Delta\end{array}}{A \supset B,\Gamma \Rightarrow \Delta}\,L\supset
\quad\mapsto\quad
\begin{array}{cl}
\langle\rangle & \\ \hline\hline
\langle\ \langle\rangle ; \langle\rangle\ \rangle & \mathsf{p} \\
\cdots\cdots\cdots\cdots\cdots\cdots & \pi_1,\pi_2 \\
\langle \Gamma \Rightarrow \Delta, A ;\ B,\Gamma \Rightarrow \Delta\rangle & \\ \hline\hline
\Gamma,\Gamma\ \langle\ \Rightarrow A ;\ B \Rightarrow\ \rangle\ \Delta,\Delta & \mathsf{f}{-}{\downarrow},\mathsf{f}{+}{\downarrow} \\ \hline\hline
\Gamma\ \langle\ \Rightarrow A ;\ B \Rightarrow\ \rangle\ \Delta & \mathsf{c}{-},\mathsf{c}{+} \\ \hline
A \supset B,\Gamma \Rightarrow \Delta & \supset{-}
\end{array}
$$

$$
\dfrac{\begin{array}{c}\vdots\ \pi_1\\ A,\Gamma \Rightarrow \Delta, B\end{array}}{\Gamma \Rightarrow \Delta, A \supset B}\,R\supset
\quad\mapsto\quad
\begin{array}{cl}
\langle\rangle & \\ \hline
\Rightarrow (\langle\rangle) & \mathsf{p}{+} \\
\cdots\cdots\cdots\cdots\cdots & \pi_1 \\
\Rightarrow (A,\Gamma \Rightarrow \Delta, B) & \\ \hline\hline
\Rightarrow \Delta, (A,\Gamma \Rightarrow B) & \mathsf{f}{+}{+}{\downarrow} \\ \hline\hline
\Gamma \Rightarrow \Delta, (A \Rightarrow B) & \mathsf{f}{-}{+}{\downarrow} \\ \hline
\Gamma \Rightarrow \Delta, A \supset B & \supset{+}
\end{array}
$$

$\square$

Bibliography

Here are the references in alphabetical order.

[1] Andreas Abel. ‘Normalization by Evaluation: Dependent Types and Impredicativity’. Habilitationsschrift. Ludwig-Maximilians-University Munich, May 2013 (cited on page 209).

[2] Samson Abramsky and Achim Jung. *Domain Theory.* USA: Oxford University Press, Inc., 1995 (cited on page 191).

[3] Wolfgang Ahrendt and Sarah Grebing. ‘Using the KeY Prover’. en. In: *Deductive Software Verification – The KeY Book.* Ed. by Wolfgang Ahrendt et al. Vol. 10001. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 495–539. DOI: `10.1007/978-3-319-49812-6_15`. (Visited on 11/23/2021) (cited on page 36).

[4] Ahmed Amerkad et al. *Mathematics and Proof Presentation in Pcoq.* Tech. rep. RR-4313. INRIA, Nov. 2001. (Visited on 12/08/2021) (cited on page 35).

[5] Jean-Marc Andreoli. ‘Logic Programming with Focusing Proofs in Linear Logic’. In: *Journal of Logic and Computation* 2.3 (1992), pp. 297–347. (Visited on 06/09/2020) (cited on pages 40, 51, 241).

[6] Andrews. ‘Refutations by Matings’. In: *IEEE Transactions on Computers* C-25.8 (1976), pp. 801–807 (cited on page 58).

[7] Emilio Jesús Gallego Arias, Benoît Pin, and Pierre Jouvelot. ‘jsCoq: Towards Hybrid Theorem Proving Interfaces’. In: *Electronic Proceedings in Theoretical Computer Science* 239 (2017), pp. 15–27. DOI: `10.4204/eptcs.239.2` (cited on page 84).

[8] Federico Aschieri. ‘On Natural Deduction for Herbrand Constructive Logics III: The Strange Case of the Intuitionistic Logic of Constant Domains’. en. In: *Electronic Proceedings in Theoretical Computer Science* 281 (Oct. 2018), pp. 1–9. DOI: `10.4204/EPTCS.281.1`. (Visited on 08/25/2023) (cited on pages 117, 123).

[9] Albert Atkin. ‘Peirce’s Theory of Signs’. In: *The Stanford Encyclopedia of Philosophy.* Ed. by Edward N. Zalta and Uri Nodelman. Spring 2023. Metaphysics Research Lab, Stanford University, 2023. (Visited on 01/09/2024) (cited on page 17).

[10] Serge Autexier. ‘Hierarchical Contextual Reasoning’. PhD thesis. Universität des Saarlandes, 2004 (cited on page 22).

[11] Edward W. Ayers. ‘A Tool for Producing Verified, Explainable Proofs.’ en. PhD thesis. University of Cambridge, 2021 (cited on pages 13, 22, 76, 100, 241).

[12] Edward W. Ayers, Mateja Jamnik, and W. T. Gowers. ‘A Graphical User Interface Framework for Formal Verification’. In: *12th International Conference on Interactive Theorem Proving (ITP 2021).* Ed. by Liron Cohen and Cezary Kaliszyk. Vol. 193. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 4:1–4:16. DOI: `10.4230/LIPIcs.ITP.2021.4` (cited on pages 13, 27, 84, 98).

[13] Stefan Banach and Alfred Tarski. ‘Sur la décomposition des ensembles de points en parties respectivement congruentes’. pl. In: *Fundamenta Mathematicae* 6 (1924). Publisher: Instytut Matematyczny Polskiej Akademii Nauk, pp. 244–277. DOI: `10.4064/fm-6-1-244-277`. (Visited on 01/01/2024) (cited on page 4).

[14] Gilles Barthe et al. ‘EasyCrypt: A Tutorial’. In: *Foundations of Security Analysis and Design VII: FOSAD 2012/2013 Tutorial Lectures.* Ed. by Alessandro Aldini, Javier Lopez, and Fabio Martinelli. Cham: Springer International Publishing, 2014, pp. 146–166. DOI: `10.1007/978-3-319-10082-1_6` (cited on page 83).

[15] Evmorfia-Iro Bartzia et al. ‘Proof assistants for undergraduate mathematics education: elements of an a priori analysis’. working paper or preprint. Apr. 2023 (cited on pages 65, 68, 69, 99).

[16] Andrej Bauer. *Continuous nowhere differentiability and constructive mathematics*. MathOverflow. URL: https://mathoverflow.net/q/439047 (cited on page 4).

[17] Gerard Berry and Gerard Boudol. 'The chemical abstract machine'. In: *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. POPL '90. New York, NY, USA: Association for Computing Machinery, Dec. 1989, pp. 81–94. DOI: 10.1145/96709.96717. (Visited on 05/02/2022) (cited on pages 103, 104).

[18] Yves Bertot, Gilles Kahn, and Laurent Théry. 'Proof by pointing'. In: *Theoretical Aspects of Computer Software*. Ed. by Masami Hagiya and John C. Mitchell. Red. by Gerhard Goos and Juris Hartmanis. Vol. 789. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 141–160. DOI: 10.1007/3-540-57887-0_94. (Visited on 04/21/2020) (cited on pages 16, 30, 62, 103, 126).

[19] Marc Bezem and Thierry Coquand. 'Automating Coherent Logic'. en. In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Ed. by Geoff Sutcliffe and Andrei Voronkov. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 246–260. DOI: 10.1007/11591191_18 (cited on page 193).

[20] W. Bibel and C. Kreitz. 'Connection method'. In: *Scholarpedia* 4.1 (2009). revision #127407, p. 6816. DOI: 10.4249/scholarpedia.6816 (cited on page 242).

[21] François Bobot et al. 'Why3: Shepherd Your Herd of Provers'. In: *Boogie 2011: First International Workshop on Intermediate Verification Languages*. https://hal.inria.fr/hal-00790310. Wrocław, Poland, Aug. 2011, pp. 53–64 (cited on page 83).

[22] Filippo Bonchi, Alessandro Di Giorgio, and Alessio Santamaria. 'Deconstructing the Calculus of Relations with Tape Diagrams'. en. In: *Proceedings of the ACM on Programming Languages* 7.POPL (Jan. 2023), pp. 1864–1894. DOI: 10.1145/3571257. (Visited on 01/21/2024).

[23] Filippo Bonchi et al. *Diagrammatic Algebra of First Order Logic*. arXiv:2401.07055 [cs, math]. Jan. 2024. DOI: 10.48550/arXiv.2401.07055. URL: http://arxiv.org/abs/2401.07055 (visited on 01/21/2024) (cited on page 155).

[24] George Boole. *The Laws of Thought*. London: The Open court publishing company, 1854 (cited on page 2).

[25] Geraldine Brady and Todd H. Trimble. 'A categorical interpretation of C.S. Peirce's propositional logic Alpha'. en. In: *Journal of Pure and Applied Algebra* 149.3 (June 2000), pp. 213–239. DOI: 10.1016/S0022-4049(98)00179-0. (Visited on 02/18/2024) (cited on pages 132, 155, 161).

[26] Geraldine Brady and Todd H. Trimble. *A String Diagram Calculus for Predicate Logic and C. S. Peirce's System Beta*. June 2000. URL: https://ncatlab.org/nlab/files/BradyTrimbleString.pdf (visited on 02/22/2023) (cited on page 155).

[27] Joachim Breitner. 'Visual Theorem Proving with the Incredible Proof Machine'. In: *Interactive Theorem Proving*. Ed. by Jasmin Christian Blanchette and Stephan Merz. Vol. 9807. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 123–139. DOI: 10.1007/978-3-319-43144-4_8. (Visited on 08/09/2020) (cited on page 36).

[28] Taus Brock-Nannestad and Danko Ilik. 'An Intuitionistic Formula Hierarchy Based on High-School Identities'. In: *Mathematical Logic Quarterly* 65.1 (May 2019). arXiv: 1601.04876, pp. 57–79. DOI: 10.1002/malq.201700047. (Visited on 04/06/2022) (cited on page 241).

[29] Kai Brünnler. 'Deep sequent systems for modal logic'. en. In: *Archive for Mathematical Logic* 48.6 (July 2009), pp. 551–577. DOI: 10.1007/s00153-009-0137-3. (Visited on 01/10/2024) (cited on page 19).

[30] Kai Brünnler and Richard McKinley. 'An Algorithmic Interpretation of a Deep Inference System'. en. In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Ed. by Iliano Cervesato, Helmut Veith, and Andrei Voronkov. Vol. 5330. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 482–496. DOI: 10.1007/978-3-540-89439-1_34. (Visited on 05/12/2022) (cited on page 244).

[31] Kai Brünnler and Alwen Fernanto Tiu. 'A Local System for Classical Logic'. en. In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Ed. by Robert Nieuwenhuis and Andrei Voronkov. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2001, pp. 347–361. DOI: 10.1007/3-540-45653-8_24 (cited on pages 156, 169, 175).

[32] Paola Bruscoli and Alessio Guglielmi. 'On Analyticity in Deep Inference'. en. In: *Mathematical Structures in Computer Science* 29.Special Issue 8: A special issue on structural proof theory, automated reasoning and computation in celebration of Dale Miller's 60th birthday (2019). DOI: https://doi.org/10.1017/S0960129519000136 (cited on page 242).

[33] Robert Burch. 'Charles Sanders Peirce'. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Summer 2022. Metaphysics Research Lab, Stanford University, 2022 (cited on pages 165, 176, 180, 184).

[34] Etienne Callies and Olivier Laurent. 'Click and coLLecT An Interactive Linear Logic Prover'. In: *5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021)*. Rome (virtual), Italy, June 2021 (cited on page 36).

[35] Gianluca Caterina and Rocco Gangle. 'A New Syntax for Diagrammatic Logic: A Generic Figures Approach'. en. In: *Numerical Computations: Theory and Algorithms*. Ed. by Yaroslav D. Sergeyev and Dmitri E. Kvasov. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 43–58. DOI: 10.1007/978-3-030-40616-5_4 (cited on page 155).

[36] Luc Chabassier and Bruno Barras. 'A graphical interface for diagrammatic proofs in proof assistants'. 29th International Conference on Types for Proofs and Programs. 2023 (cited on page 99).

[37] Kaustuv Chaudhuri. 'Subformula Linking as an Interaction Method'. In: *Interactive Theorem Proving*. Ed. by Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie. Red. by David Hutchison et al. Vol. 7998. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 386–401. DOI: 10.1007/978-3-642-39634-2_28. (Visited on 04/21/2020) (cited on pages 16, 18, 38, 40, 52, 54, 199, 242).

[38] Kaustuv Chaudhuri. 'Interactive Proof Building with Direct Manipulation for Linear Logic (and Cousins)'. Invited Talk at the *Linearity & TLLA* workshop. 2020.

[39] Kaustuv Chaudhuri. 'Subformula Linking for Intuitionistic Logic with Application to Type Theory'. In: *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*. Ed. by André Platzer and Geoff Sutcliffe. Vol. 12699. Lecture Notes in Computer Science. Springer, 2021, pp. 200–216. DOI: 10.1007/978-3-030-79876-5_12 (cited on pages 52, 58, 97).

[40] Kaustuv Chaudhuri. *Profound–Intuitionistic*. Version 0.9.2. 2022. URL: https://chaudhuri.info/research/profint/ (visited on 08/20/2023) (cited on page 113).

[41] Kaustuv Chaudhuri, Nicolas Guenot, and Lutz Strassburger. 'The Focused Calculus of Structures'. In: *20th EACSL Annual Conference on Computer Science Logic*. Ed. by Marc Bezem. Vol. 12. Leibniz International Proceedings in Informatics (LIPIcs). Bergen, Norway, Sept. 2011, pp. 159–173. DOI: 10.4230/LIPIcs.CSL.2011.159 (cited on page 242).

[42] Kaustuv Chaudhuri, Stefan Hetzl, and Dale Miller. 'A multi-focused proof system isomorphic to expansion proofs'. In: *Journal of Logic and Computation* 26.2 (Apr. 2016), pp. 577–603. DOI: 10.1093/logcom/exu030. (Visited on 06/22/2020).

[43] Kaustuv Chaudhuri, Sonia Marin, and Lutz Straßburger. 'Focused and Synthetic Nested Sequents'. In: *Foundations of Software Science and Computation Structures*. Ed. by Bart Jacobs and Christof Löding. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 390–407 (cited on page 242).

[44] Kaustuv Chaudhuri, Sonia Marin, and Lutz Straßburger. 'Modular Focused Proof Systems for Intuitionistic Modal Logics'. In: *International Conference on Formal Structures for Computation and Deduction*. 2016.

[45] Kaustuv Chaudhuri, Dale Miller, and Alexis Saurin. 'Canonical Sequent Proofs via Multi-Focusing'. In: *Fifth Ifip International Conference On Theoretical Computer Science – Tcs 2008*. Ed. by Giorgio Ausiello et al. Vol. 273. ISSN: 1571-5736 Series Title: IFIP International Federation for Information Processing. Boston, MA: Springer US, 2008, pp. 383–396. DOI: 10.1007/978-0-387-09680-3_26. (Visited on 08/23/2020).

[46] Kaustuv Chaudhuri et al. *Certifying Proof-By-Linking*. en. Sept. 2022. URL: https://inria.hal.science/hal-04317972 (visited on 12/09/2023) (cited on page 97).

[47] Alonzo Church. 'A Set of Postulates for the Foundation of Logic'. In: *Annals of Mathematics* 33.2 (1932), pp. 346–366. (Visited on 01/04/2024) (cited on page 11).

[48] Alonzo Church. 'A Formulation of the Simple Theory of Types'. In: *The Journal of Symbolic Logic* 5.2 (1940), pp. 56–68. (Visited on 01/03/2024) (cited on pages 11, 67).

[49] Ranald Clouston et al. 'Annotation-Free Sequent Calculi for Full Intuitionistic Linear Logic'. In: *Computer Science Logic 2013 (CSL 2013)*. Ed. by Simona Ronchi Della Rocca. Vol. 23. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013, pp. 197–214. DOI: 10.4230/LIPIcs.CSL.2013.197 (cited on pages 120, 121, 124).

[50] Coq Community. *A Visual Studio Code extension for Coq*. https://github.com/coq-community/vscoq. 2023. (Visited on 10/05/2023) (cited on page 86).

[51] Microsoft Corporation. *Official page for Language Server Protocol*. https://microsoft.github.io/language-server-protocol/. 2023. (Visited on 10/05/2023) (cited on page 101).

[52] Louis Couturat. *La Logique de Leibniz*. Ed. by Félix Alcan. Paris: Ancienne librairie Germer Baillère et C[ie], 1901 (cited on page 17).

[53] Tristan Crolard. 'Subtractive logic'. en. In: *Theoretical Computer Science* 254.1-2 (Mar. 2001), pp. 151–185. DOI: 10.1016/S0304-3975(99)00124-3. (Visited on 10/15/2022) (cited on pages 117, 123).

[54] Haskell B. Curry, Robert Feys, and William Craig. 'Combinatory Logic, Volume I'. In: *Philosophical Review* 68.4 (1959), pp. 548–550. DOI: 10.2307/2182503 (cited on page 12).

[55] Evan Czaplicki and Stephen Chong. 'Asynchronous Functional Reactive Programming for GUIs'. In: *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '13. Seattle, Washington, USA: Association for Computing Machinery, 2013, pp. 411–422. DOI: 10.1145/2491956.2462161 (cited on page 231).

[56] Nicolaas Govert de Bruijn. 'A survey of the project Automath'. English. In: *To H.B. Curry : Essays on combinatory logic, lambda calculus and formalism*. Ed. by J.P. Seldin and J.R. Hindley. United States: Academic Press Inc., 1980, pp. 579–606 (cited on page 12).

[57] Nicolaas Govert de Bruijn. 'The Mathematical Vernacular, A Language for Mathematics with Typed Sets'. en. In: *Studies in Logic and the Foundations of Mathematics*. Vol. 133. Elsevier, 1994, pp. 865–935. DOI: 10.1016/S0049-237X(08)70231-3. (Visited on 12/17/2023) (cited on page 232).

[58] Richard Dedekind. *Was sind und was sollen die Zahlen?* 1888. URL: http://www.opera-platonis.de/dedekind/Dedekind_Was_sind_2.pdf (cited on page 7).

[59] Deducteam. *The Dedukti system*. 2013. URL: https://deducteam.github.io/ (visited on 01/04/2024).

[60] Anatoli Degtyarev and Andrei Voronkov. 'The Inverse Method'. In: *Handbook of Automated Reasoning (in 2 volumes)*. Ed. by John Alan Robinson and Andrei Voronkov. Elsevier and MIT Press, 2001, pp. 179–272. DOI: 10.1016/B978-044450813-3/50006-0 (cited on page 181).

[61] David Delahaye. 'A Tactic Language for the System Coq'. en. In: *Logic for Programming and Automated Reasoning*. Ed. by Michel Parigot and Andrei Voronkov. Lecture Notes in Artificial Intelligence. Berlin, Heidelberg: Springer, 2000, pp. 85–95. DOI: 10.1007/3-540-44404-1_7.

[62] Roberto Di Cosmo. *Isomorphisms of types: from λ-calculus to information retrieval and language design*. ISBN-0-8176-3763-X. Birkhauser, 1995 (cited on page 33).

[63] Pablo Donato. *flowers-metatheory*. Nov. 2022. URL: https://github.com/Champitoad/flowers-metatheory (visited on 11/16/2023) (cited on page 209).

[64] Pablo Donato. *flower-auto*. Dec. 2023. URL: https://github.com/Champitoad/flower-auto (visited on 12/08/2023) (cited on page 228).

[65] Pablo Donato. *flower-prover*. Oct. 2023. URL: https://github.com/Champitoad/flower-prover (visited on 12/08/2023) (cited on page 231).

[66] Pablo Donato. 'The Flower Calculus'. Preprint. Feb. 2024 (cited on page 20).

[67] Pablo Donato, Pierre-Yves Strub, and Benjamin Werner. 'A drag-and-drop proof tactic'. In: *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2022. Philadelphia, PA, USA: Association for Computing Machinery, 2022, pp. 197–209. DOI: 10.1145/3497775.3503692 (cited on page 20).

[68] Pablo Donato and Benjamin Werner. *coq-actema*. 2023. URL: https://github.com/Champitoad/coq-actema (visited on 02/24/2024) (cited on pages 20, 82).

[69] Pablo Donato et al. *The Actema prototype, standalone version*. https://actema.xyz. 2021 (cited on pages 25, 82).

[70] Andrei Dorman and Damiano Mazza. 'A Hierarchy of Expressiveness in Concurrent Interaction Nets'. In: *CONCUR 2013 – Concurrency Theory*. Ed. by Pedro R. D'Argenio and Hernán Melgratti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 197–211 (cited on page 184).

[71] Paul Downen et al. 'Sequent calculus as a compiler intermediate language'. en. In: *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*. Nara Japan: ACM, Sept. 2016, pp. 74–88. DOI: 10.1145/2951913.2951931. (Visited on 08/20/2023) (cited on page 110).

[72] A. G. Dragalin and E. Mendelson. 'Mathematical Intuitionism. Introduction to Proof Theory'. In: *Journal of Symbolic Logic* 55.3 (1990). Publisher: Association for Symbolic Logic, pp. 1308–1309 (cited on page 76).

[73] Roy Dyckhoff. 'Contraction-Free Sequent Calculi for Intuitionistic Logic'. In: *Journal of Symbolic Logic* 57.3 (1992). Publisher: Association for Symbolic Logic, pp. 795–807. DOI: 10.2307/2275431 (cited on pages 117, 126).

[74] Roy Dyckhoff and Sara Negri. 'Geometrization of First-Order Logic'. In: *Bulletin of Symbolic Logic* 21.2 (2015), pp. 123–163. DOI: 10.1017/bsl.2015.7 (cited on page 193).

[75] Conal M. Elliott. 'Tangible Functional Programming'. In: *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming*. ICFP '07. Freiburg, Germany: Association for Computing Machinery, 2007, pp. 59–70. DOI: 10.1145/1291151.1291163 (cited on page 59).

[76] Boris Eng. 'An exegesis of transcendental syntax'. en. PhD thesis. Université Sorbonne Paris Nord, June 2023. (Visited on 09/01/2023) (cited on pages 125, 165, 180).

[77] Boris Eng and Thomas Seiller. *Stellar Resolution: Multiplicatives*. arXiv:2007.16077 [cs]. July 2020. DOI: 10.48550/arXiv.2007.16077. URL: http://arxiv.org/abs/2007.16077 (visited on 09/20/2023) (cited on page 158).

[78] Melvin Fitting. 'Nested Sequents for Intuitionistic Logics'. en. In: *Notre Dame Journal of Formal Logic* 55.1 (2014), pp. 41–61. DOI: 10.1215/00294527-2377869. (Visited on 04/30/2020) (cited on pages 118, 121).

[79] Yannick Forster, Dominik Kirst, and Dominik Wehr. 'Completeness Theorems for First-Order Logic Analysed in Constructive Type Theory (Extended Version)'. In: *Journal of Logic and Computation* 31.1 (Jan. 2021). arXiv:2006.04399 [cs, math], pp. 112–151. DOI: 10.1093/logcom/exaa073. (Visited on 09/04/2023) (cited on page 134).

[80] Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle, 1879 (cited on page 2).

[81] Gottlob Frege. *Translations From the Philosophical Writings of Gottlob Frege*. Ed. by P. T. Geach and Max Black. Oxford, England: Blackwell, 1952 (cited on page 17).

[82] Gottlob Frege. *L'idéographie*. Ed. by Vrin. Trans. by Corine Besson. Paris: Vrin, 1999 (cited on page 17).

[83] Dan Frumin. 'Semantic Cut Elimination for the Logic of Bunched Implications, Formalized in Coq'. In: *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2022. Philadelphia, PA, USA: Association for Computing Machinery, 2022, pp. 291–306. DOI: 10.1145/3497775.3503690 (cited on page 210).

[84] Galileo Galilei. *The Assayer*. Trans. by Stillman Drake. 1623 (cited on page 2).

[85] Emilio Jesús Gallego Arias. 'SerAPI: Machine-Friendly, Data-Centric Serialization for COQ'. working paper or preprint. Oct. 2016 (cited on page 84).

[86] Emilio Jesús Gallego Arias. *coq-lsp protocol documentation*. https://github.com/ejgallego/coq-lsp/blob/main/etc/doc/PROTOCOL.md. 2023. (Visited on 10/05/2023) (cited on page 101).

[87] M. Ganesalingam and W. T. Gowers. 'A Fully Automatic Theorem Prover with Human-Style Output'. en. In: *Journal of Automated Reasoning* 58.2 (Feb. 2017), pp. 253–291. DOI: 10.1007/s10817-016-9377-1. (Visited on 12/12/2023) (cited on page 241).

[88] Gerhard Gentzen. 'Untersuchungen über das logische Schließen. I'. de. In: *Mathematische Zeitschrift* 39.1 (Dec. 1935), pp. 176–210. DOI: 10.1007/BF01201353. (Visited on 12/10/2023) (cited on pages 5, 6, 223).

[89] Gerhard Gentzen. 'Die Widerspruchsfreiheit der reinen Zahlentheorie'. de. In: *Mathematische Annalen* 112.1 (Dec. 1936), pp. 493–565. DOI: 10.1007/BF01565428. (Visited on 01/02/2024) (cited on page 9).

[90] Herman Geuvers. 'Proof assistants: History, ideas and future'. en. In: *Sadhana* 34.1 (Feb. 2009), pp. 3–25. DOI: 10.1007/s12046-009-0001-5. (Visited on 04/03/2023) (cited on pages 11, 12, 14).

[91] Stéphane Gimenez and Georg Moser. 'The Structure of Interaction'. en. In: *DROPS-IDN/v2/document/10.4230/LIPIcs.CSL.2013.316*. Schloss-Dagstuhl - Leibniz Zentrum für Informatik, 2013. DOI: 10.4230/LIPIcs.CSL.2013.316. (Visited on 12/12/2023) (cited on page 244).

[92] Jean-Yves Girard. 'Linear logic'. In: *Theoretical Computer Science* 50.1 (1987), pp. 1–101. DOI: https://doi.org/10.1016/0304-3975(87)90045-4 (cited on pages 58, 78, 155, 184, 210).

[93] Jean-Yves Girard. 'Locus Solum: From the rules of logic to the logic of rules'. en. In: *Mathematical Structures in Computer Science* 11.3 (June 2001), pp. 301–506. DOI: 10.1017/S096012950100336X. (Visited on 09/01/2023) (cited on pages 125, 164, 165).

[94] Jean-Yves Girard. *The blind spot*. European Mathematical Society (EMS), Sept. 2011, 550 pages (cited on pages 78, 106, 131, 179).

[95] Jean-Yves Girard. 'Transcendental syntax I: deterministic case'. In: *Mathematical Structures in Computer Science* 27.5 (2017), pp. 827–849. DOI: 10.1017/S0960129515000407 (cited on page 242).

[96] Jean-Yves Girard. *Schrödinger's cut III. Les univers parallèles*. Online Notes. Dec. 2022. URL: https://girard.perso.math.cnrs.fr/chat3.pdf (visited on 11/16/2023) (cited on page 209).

[97] Marianna Girlando et al. 'Intuitionistic S4 is decidable'. en. In: *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. arXiv:2304.12094 [cs]. June 2023, pp. 1–13. DOI: 10.1109/LICS56636.2023.10175684. (Visited on 12/10/2023) (cited on page 223).

[98] GeoGebra GmbH. *GeoGebra Geometry*. http://www.geogebra.org/geometry. 2023 (cited on page 27).

[99] Adele Goldberg and Alan Kay. *SMALLTALK-72 INSTRUCTION MANUAL*. en. Xerox Palo Alto Research Center: Xerox Corporation, Mar. 1976 (cited on page 235).

[100] Georges Gonthier, Assia Mahboubi, and Enrico Tassi. *A Small Scale Reflection Extension for the Coq system*. Research Report RR-6455. Inria Saclay Ile de France, 2016 (cited on pages 77, 96).

[101] R. Goré and Ian Shillito. 'Bi-Intuitionistic Logics: A New Instance of an Old Problem'. In: 2020. (Visited on 08/25/2023) (cited on page 147).

[102] Rajeev Goré. 'Dual Intuitionistic Logic Revisited'. en. In: *Automated Reasoning with Analytic Tableaux and Related Methods*. Ed. by Roy Dyckhoff. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2000, pp. 252–267. DOI: 10.1007/10722086_21 (cited on page 123).

[103] Rajeev Goré, Linda Postniece, and Alwen Tiu. 'Cut-elimination and proof-search for bi-intuitionistic logic using nested sequents'. In: *Advances in Modal Logic 7, papers from the seventh conference on "Advances in Modal Logic," held in Nancy, France, 9-12 September 2008*. Ed. by Carlos Areces and Robert Goldblatt. College Publications, 2008, pp. 43–66.

[104] Ivor Grattan-Guinness. *The Search for Mathematical Roots, 1870-1940: Logics, Set Theories and the Foundations of Mathematics from Cantor through Russell to Godel*. Princeton University Press, 2000. (Visited on 11/04/2023) (cited on page 184).

[105] J. Grundy. 'Window Inference In The HOL System'. In: *1991 International Workshop on the HOL Theorem Proving System and Its Applications*. Aug. 1991, pp. 177–189. DOI: 10.1109/HOL.1991.596285 (cited on page 35).

[106] Jim Grundy. 'A Window Inference Tool for Refinement'. In: *5th Refinement Workshop*. Ed. by Cliff B. Jones, Roger C. Shaw, and Tim Denvir. London: Springer London, 1992, pp. 230–254 (cited on page 35).

[107] Nicolas Guenot. 'Nested Proof Search as Reduction in the Lambda-Calculus'. In: *Proceedings of the 13th International ACM SIGPLAN Symposium on Principles and Practices of Declarative Programming*. PPDP '11. Odense, Denmark: Association for Computing Machinery, 2011, pp. 183–194. DOI: 10.1145/2003476.2003501 (cited on page 242).

[108] Nicolas Guenot. 'Nested Deduction in Logical Foundations for Computation'. en. PhD thesis. Ecole Polytechnique X, Apr. 2013. (Visited on 03/05/2021) (cited on pages 110, 117, 118, 150, 154, 244).

[109] Giulio Guerrieri, Willem B. Heijltjes, and Joseph W. N. Paulus. 'A Deep Quantitative Type System'. In: *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*. Ed. by Christel Baier and Jean Goubault-Larrecq. Vol. 183. Leibniz International Proceedings in Informatics (LIPIcs). ISSN: 1868-8969. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021, 24:1–24:24. DOI: 10.4230/LIPIcs.CSL.2021.24. (Visited on 05/12/2022) (cited on page 245).

[110] Alessio Guglielmi. *A Calculus of Order and Interaction*. Tech. rep. Technische Universität Dresden, 1999 (cited on pages 9, 22, 108, 224).

[111] Alessio Guglielmi. *Deep Inference*. Online Notes. 2014. URL: https://people.bath.ac.uk/ag248/p/DI.pdf (visited on 01/02/2024) (cited on page 126).

[112] Alessio Guglielmi. *All about Proofs, Proofs for All*. Ed. by Bruno Woltzenlogel Paleo and David Delahaye. College Publications, 2015. Chap. Deep Inference.

[113] Tom Gundersen, Willem Heijltjes, and Michel Parigot. 'Atomic Lambda Calculus: A Typed Lambda-Calculus with Explicit Sharing'. In: *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*. ISSN: 1043-6871. June 2013, pp. 311–320. DOI: 10.1109/LICS.2013.37 (cited on page 244).

[114] Nathan Haydon. *C.S Peirce's Early Developments in Linear Logic*. Draft. 2023 (cited on page 244).

[115] Nathan Haydon and Paweł Sobociński. 'Compositional Diagrammatic First-Order Logic'. en. In: *Diagrammatic Representation and Inference*. Ed. by Ahti-Veikko Pietarinen et al. Vol. 12169. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 402–418. DOI: 10.1007/978-3-030-54249-8_32. (Visited on 02/05/2023) (cited on pages 155, 178, 181).

[116] Fanny He. 'The Atomic Lambda-Mu Calculus'. PhD thesis. University of Bath, Jan. 2018 (cited on page 245).

[117] Willem B. Heijltjes, Dominic J. D. Hughes, and Lutz StraBburger. 'Intuitionistic proofs without syntax'. en. In: *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. Vancouver, BC, Canada: IEEE, June 2019, pp. 1–13. DOI: 10.1109/LICS.2019.8785827. (Visited on 08/23/2020) (cited on pages 58, 125).

[118] Hugo Herbelin. *Duality of computation and sequent calculus : a few more remarks*. Online Notes. 2008. URL: http://pauillac.inria.fr/~herbelin/publis/full-dual-lk.pdf (cited on page 116).

[119] Hugo Herbelin and Gyesik Lee. 'Forcing-Based Cut-Elimination for Gentzen-Style Intuitionistic Sequent Calculus'. In: *Logic, Language, Information and Computation*. Ed. by Hiroakira Ono, Makoto Kanazawa, and Ruy de Queiroz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 209–217 (cited on page 210).

[120] Olivier Hermant. ‘Semantic Cut Elimination in the Intuitionistic Sequent Calculus’. en. In: *Typed Lambda Calculi and Applications*. Ed. by David Hutchison et al. Vol. 3461. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 221–233. DOI: 10.1007/11417170_17. (Visited on 11/22/2022) (cited on pages 210, 217, 219).

[121] Jaakko Hintikka. *Logic, Language-Games and Information: Kantian Themes in the Philosophy of Logic*. Oxford, Clarendon Press, 1973 (cited on page 159).

[122] Christopher Hookway. *Peirce*. Ed. by Ted Honderich. New York: Routledge, 1985 (cited on page 17).

[123] William Alvin Howard. ‘The Formulae-as-Types Notion of Construction’. In: *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Ed. by Haskell Curry et al. Academic Press, 1980 (cited on page 12).

[124] John Howse, Gem Stapleton, and John Taylor. ‘Spider Diagrams’. en. In: *LMS Journal of Computation and Mathematics* 8 (Jan. 2005). Publisher: Cambridge University Press, pp. 145–194. DOI: 10.1112/S1461157000000942. (Visited on 02/12/2024).

[125] Xu Huang. *Synthetic Tait Computability the Hard Way*. 2023. URL: http://arxiv.org/abs/2310.02051 (cited on page 242).

[126] Gerard P. Huet. ‘The undecidability of unification in third order logic’. In: *Information and Control* 22.3 (1973), pp. 257–267. DOI: https://doi.org/10.1016/S0019-9958(73)90301-X (cited on page 98).

[127] Dominic Hughes. ‘Proofs without syntax’. In: *Annals of Mathematics* 164.3 (Nov. 2006), pp. 1065–1076. DOI: 10.4007/annals.2006.164.1065 (cited on pages 54, 58).

[128] Dominic J. D. Hughes. *First-order proofs without syntax*. 2019. URL: https://arxiv.org/abs/1906.11236 (cited on page 59).

[129] Danko Ilik. ‘Constructive Completeness Proofs and Delimited Control’. PhD thesis. Ecole Polytechnique X, Oct. 2010 (cited on page 210).

[130] Martin Irvine. ‘Semiotics in Computing and Information Systems’. en. In: *Semiotics in the Natural and Technical Sciences*. Ed. by Jamin Pelkey and Stéphanie Walsh Matthews. Vol. 2. Bloomsbury Semiotics. Bloomsbury Publishing, Dec. 2022, p. 34 (cited on page 159).

[131] Predrag Janičić and Julien Narboux. ‘Automated generation of illustrated proofs in geometry and beyond’. en. In: *Annals of Mathematics and Artificial Intelligence* (July 2023). DOI: 10.1007/s10472-023-09857-y. (Visited on 11/12/2023) (cited on page 193).

[132] Stanisław Jaśkowski. ‘On the Rules of Suppositions in Formal Logic’. In: Oxford at the Clarendon Press, 1934 (cited on page 5).

[133] Phillip E. Johnson. ‘The Genesis and Development of Set Theory’. In: *The Two-Year College Mathematics Journal* 3.1 (1972), pp. 55–62. (Visited on 12/29/2023) (cited on page 2).

[134] Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*. Vol. 2. Oxford Logic Guides. Oxford, England: Clarendon Press, Sept. 2002.

[135] Jean-Baptiste Joinet. ‘Completeness of MLL proof-nets w.r.t. weak distributivity’. In: *The Journal of Symbolic Logic* 72.1 (Mar. 2007). Publisher: Association for Symbolic Logic, pp. 159–170. (Visited on 05/22/2022) (cited on page 109).

[136] Ralf Jung et al. ‘Iris from the ground up: A modular foundation for higher-order concurrent separation logic’. en. In: *Journal of Functional Programming* 28 (2018), e20. DOI: 10.1017/S0956796818000151. (Visited on 01/22/2024) (cited on page 58).

[137] Ozan Kahramanogullari. ‘Interaction and Depth against Nondeterminism in Proof Search’. In: *Logical Methods in Computer Science* Volume 10, Issue 2 (May 2014). DOI: 10.2168/LMCS-10(2:5)2014 (cited on pages 223, 224, 242).

[138] Ozan Kahramanogullari, Pierre-Etienne Moreau, and Antoine Reilles. ‘Implementing Deep Inference in Tom’. en. In: May 2005, p. 158. (Visited on 12/10/2023) (cited on page 97).

[139] Ozan Kahramanoğulları. 'Reducing Nondeterminism in the Calculus of Structures'. en. In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Ed. by Miki Hermann and Andrei Voronkov. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 272–286. DOI: 10.1007/11916277_19 (cited on pages 223, 242).

[140] Ozan Kahramanoğulları. 'Maude as a Platform for Designing and Implementing Deep Inference Systems'. en. In: *Electronic Notes in Theoretical Computer Science* 219 (Nov. 2008), pp. 35–50. DOI: 10.1016/j.entcs.2008.10.033. (Visited on 12/10/2023) (cited on page 97).

[141] Alfred Bray Kempe. 'A memoir on the theory of mathematical form'. In: *Philosophical Transactions of the Royal Society of London* 177 (Jan. 1886). Publisher: Royal Society, pp. 1–70. DOI: 10.1098/rstl.1886.0002. (Visited on 11/05/2023) (cited on page 184).

[142] Kevin Klement. 'Russell's Logical Atomism'. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Spring 2020. Metaphysics Research Lab, Stanford University, 2020. (Visited on 08/14/2023) (cited on page 104).

[143] Roman Kuznets and Lutz Straßburger. 'Maehara-style modal nested calculi'. en. In: *Archive for Mathematical Logic* 58.3-4 (May 2019), pp. 359–385. DOI: 10.1007/s00153-018-0636-1. (Visited on 02/14/2024).

[144] Marie La Palme Reyes, Gonzalo E. Reyes, and Houman Zolfaghari. *Generic figures and their glueings. A constructive approach to functor categories*. Polimetrica, International Scientific Publisher, 2008.

[145] Ambroise Lafont. 'A diagram editor to mechanise categorical proofs'. In: *35es Journées Francophones des Langages Applicatifs (JFLA 2024)*. Saint-Jacut-de-la-Mer, France, Jan. 2024 (cited on page 27).

[146] Thomas Långbacka, Rimvydas Rukšėnas, and Joakim von Wright. 'TkWinHOL: A tool for Window Inference in HOL'. en. In: *Higher Order Logic Theorem Proving and Its Applications*. Ed. by E. Thomas Schubert, Philip J. Windley, and James Alves-Foss. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1995, pp. 245–260. DOI: 10.1007/3-540-60275-5_69 (cited on page 35).

[147] F. William Lawvere. 'Continuously Variable Sets; Algebraic Geometry = Geometric Logic'. In: *Logic Colloquium '73*. Ed. by H.E. Rose and J.C. Shepherdson. Vol. 80. Studies in Logic and the Foundations of Mathematics. Elsevier, 1975, pp. 135–156. DOI: https://doi.org/10.1016/S0049-237X(08)71947-5 (cited on page 193).

[148] Francis William Lawvere. 'Quantifiers and Sheaves'. In: *Actes du Congrès International des Mathématiciens*. Vol. 1. Paris: Gauthier-Villars, 1970, pp. 329–334 (cited on page 193).

[149] Catherine Legg. 'The Problem of the Essential Icon'. In: *American Philosophical Quarterly* 45.3 (2008). Publisher: [North American Philosophical Publications, University of Illinois Press], pp. 207–232. (Visited on 01/09/2024) (cited on page 17).

[150] Sorin Lerner, Stephen R. Foster, and William G. Griswold. 'Polymorphic Blocks: Formalism-Inspired UI for Structured Connectors'. In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*. Ed. by Bo Begole et al. ACM, 2015, pp. 3063–3072. DOI: 10.1145/2702123.2702302 (cited on page 36).

[151] C. I. Lewis. 'A Survey of Symbolic Logic'. In: *Journal of Philosophy, Psychology and Scientific Methods* 17.3 (1920), pp. 78–79. DOI: 10.2307/2940631 (cited on page 189).

[152] Chuck Liang and Dale Miller. 'Focusing and polarization in linear, intuitionistic, and classical logics'. In: *Theoretical Computer Science* 410.46 (Nov. 2009), pp. 4747–4768. DOI: 10.1016/j.tcs.2009.07.041. (Visited on 08/12/2020).

[153] Sven Linker, Jim Burton, and Mateja Jamnik. 'Tactical Diagrammatic Reasoning'. en. In: *Electronic Proceedings in Theoretical Computer Science* 239 (Jan. 2017), pp. 29–42. DOI: 10.4204/EPTCS.239.3. (Visited on 02/12/2024).

[154] Christian van der Loo. *Cross-platform React app for solving existential graph-based proofs*. Jan. 2022. URL: https://github.com/chrisvander/existential-graphs (visited on 12/10/2023) (cited on page 231).

[155] Christoph Lüth and Burkhart Wolff. 'TAS — A Generic Window Inference System'. en. In: *Theorem Proving in Higher Order Logics*. Ed. by Gerhard Goos et al. Vol. 1869. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 406–423. DOI: 10.1007/3-540-44659-1_25. (Visited on 12/08/2021) (cited on pages 35, 36).

[156] Tim Lyon. 'Refining Labelled Systems for Modal and Constructive Logics with Applications'. PhD thesis. July 2021. DOI: 10.48550/arXiv.2107.14487. (Visited on 06/02/2022) (cited on page 223).

[157] Tim S. Lyon. 'Nested Sequents for Intuitionistic Modal Logics via Structural Refinement'. en. In: *Automated Reasoning with Analytic Tableaux and Related Methods*. Ed. by Anupam Das and Sara Negri. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 409–427. DOI: 10.1007/978-3-030-86059-2_24.

[158] Minghui Ma and Ahti-Veikko Pietarinen. 'Proof Analysis of Peirce's Alpha System of Graphs'. en. In: *Studia Logica* 105.3 (June 2017), pp. 625–647. DOI: 10.1007/s11225-016-9703-y. (Visited on 01/21/2024) (cited on page 169).

[159] Minghui Ma and Ahti-Veikko Pietarinen. 'A graphical deep inference system for intuitionistic logic'. In: *Logique et Analyse* 245 (Jan. 2019), pp. 73–114. DOI: 10.2143/LEA.245.0.3285706. (Visited on 12/16/2021) (cited on pages 158, 166, 171, 190, 191, 200, 240).

[160] Sonia Marin et al. 'From axioms to synthetic inference rules via focusing'. en. In: *Annals of Pure and Applied Logic* 173.5 (May 2022), p. 103091. DOI: 10.1016/j.apal.2022.103091. (Visited on 11/07/2023) (cited on page 241).

[161] Alberto Martelli and Ugo Montanari. 'An Efficient Unification Algorithm'. In: *ACM Trans. Program. Lang. Syst.* 4.2 (Apr. 1982), pp. 258–282. DOI: 10.1145/357162.357169 (cited on page 43).

[162] Martin Jambon. *OCaml support – atdgen*. https://atd.readthedocs.io/en/latest/atdgen.html. 2023. (Visited on 07/05/2023) (cited on page 93).

[163] Martin Jambon. *The ATD Language*. https://atd.readthedocs.io/en/latest/atd-language.html. 2023. (Visited on 07/05/2023) (cited on page 91).

[164] Per Martin-Löf. 'On the Meanings of the Logical Constants and the Justifications of the Logical Laws'. In: *Nordic Journal of Philosophical Logic* 1.1 (1996), pp. 11–60.

[165] Guido Martínez et al. *Meta-F*: Proof Automation with SMT, Tactics, and Metaprograms*. 2019 (cited on page 83).

[166] Patrick Massot. *Demo of automatically generated proof text in natural language*. https://www.imo.universite-paris-saclay.fr/~patrick.massot/Examples/ContinuousFrom.html. 2023. (Visited on 09/05/2023) (cited on pages 14, 100).

[167] Conor McBride. 'Dependently Typed Functional Programs and their Proofs'. en. PhD thesis. University of Edinburgh, July 2000. (Visited on 12/12/2023) (cited on page 241).

[168] Paul-André Melliès. 'A micrological study of negation'. en. In: *Annals of Pure and Applied Logic* 168.2 (Feb. 2017), pp. 321–372. DOI: 10.1016/j.apal.2016.10.008. (Visited on 09/20/2023) (cited on page 158).

[169] Paul-André Melliès. 'Une étude micrologique de la négation'. Habilitation à Diriger des Recherches. Université Paris Diderot, 2017 (cited on page 76).

[170] Paul-André Melliès and Noam Zeilberger. 'A bifibrational reconstruction of Lawvere's presheaf hyperdoctrine'. In: *arXiv:1601.06098 [cs, math]* (Aug. 2016). arXiv: 1601.06098. (Visited on 02/07/2022) (cited on page 155).

[171] Merriam-Webster Dictionary. *Dictionary entry for the noun "corollary"*. Online. 2024. URL: https://www.merriam-webster.com/dictionary/corollary (cited on page 195).

[172] Dale Miller. 'A compact representation of proofs'. In: *Studia Logica* 46 (1987), pp. 347–370.

[173] Dale Miller. 'PROOFCERT – Broad Spectrum Proof Certificates – ERC'. In: *Impact* 2017 (Mar. 2017), pp. 68–70. DOI: 10.21820/23987073.2017.3.68.

[174] Dale Miller. 'A Survey of the Proof-Theoretic Foundations of Logic Programming'. en. In: *Theory and Practice of Logic Programming* 22.6 (Nov. 2022). Publisher: Cambridge University Press, pp. 859–904. DOI: 10.1017/S1471068421000533. (Visited on 08/18/2023) (cited on page 109).

[175] Robin Milner. 'The use of machines to assist in rigorous proof'. In: *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences* 312.1522 (1984), pp. 411–422. DOI: 10.1098/rsta.1984.0067 (cited on page 13).

[176] Alexandre Miquel. 'Implicative algebras: a new foundation for realizability and forcing'. In: *Mathematical Structures in Computer Science* 30.5 (May 2020). arXiv:1802.00528 [math], pp. 458–510. DOI: 10.1017/S0960129520000079. (Visited on 05/22/2023) (cited on page 244).

[177] Alexandre Miquel. *Implicative algebras: a new foundation for realizability and forcing*. Topos Institute Colloquium. 2022. URL: https://www.youtube.com/watch?v=MDLpG7_7LhE (cited on page 244).

[178] Leonardo de Moura and Sebastian Ullrich. 'The Lean 4 Theorem Prover and Programming Language'. In: *Automated Deduction – CADE 28*. Ed. by André Platzer and Geoff Sutcliffe. Cham: Springer International Publishing, 2021, pp. 625–635 (cited on page 12).

[179] Ike Mulder and Robbert Krebbers. 'Unification for Subformula Linking under Quantifiers'. In: *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2024. New York, NY, USA: Association for Computing Machinery, Jan. 2024, pp. 75–88. DOI: 10.1145/3636501.3636950. (Visited on 01/21/2024) (cited on page 58).

[180] Carl Mummert. *Intuitionistic Banach-Tarski Paradox*. Mathematics Stack Exchange. URL: https://math.stackexchange.com/q/175699 (cited on page 4).

[181] Wojciech Nawrocki. *User interfaces for computer-assisted mathematics*. Master's Thesis. Oct. 2023.

[182] Wojciech Nawrocki, Edward W. Ayers, and Gabriel Ebner. 'An Extensible User Interface for Lean 4'. In: *14th International Conference on Interactive Theorem Proving (ITP 2023)*. Ed. by Adam Naumowicz and René Thiemann. Vol. 268. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 24:1–24:20. DOI: 10.4230/LIPIcs.ITP.2023.24.

[183] Sara Negri, Jan von Plato, and Aarne Ranta. *Structural Proof Theory*. 1st ed. Cambridge University Press, June 2001. (Visited on 06/08/2023) (cited on pages 54, 78, 81, 125, 147).

[184] N Nikolai. 'Vorob'ev. The derivability problem in the constructive propositional calculus with strong negation'. In: *Doklady Akademii Nauk SSSR* 85 (1952), pp. 689–692.

[185] Tobias Nipkow. 'Structured Proofs in Isar/HOL'. In: *Types for Proofs and Programs, Second International Workshop, TYPES 2002, Berg en Dal, The Netherlands, April 24-28, 2002, Selected Papers*. Ed. by Herman Geuvers and Freek Wiedijk. Vol. 2646. Lecture Notes in Computer Science. Springer, 2002, pp. 259–278. DOI: 10.1007/3-540-39185-1_15 (cited on pages 14, 36).

[186] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*. Vol. 2283. Springer Science & Business Media, 2002.

[187] Winfried Nöth. 'Peircean Semiotics in the Study of Iconicity in Language'. In: *Transactions of the Charles S. Peirce Society* 35.3 (1999). Publisher: Indiana University Press, pp. 613–619. (Visited on 09/16/2023).

[188] Mitsuhiro Okada. 'A uniform semantic proof for cut-elimination and completeness of various first and higher order logics'. In: *Theoretical Computer Science* 281.1 (2002). Selected Papers in honour of Maurice Nivat, pp. 471–498. DOI: https://doi.org/10.1016/S0304-3975(02)00024-5 (cited on page 210).

[189] Cyrus Omar et al. 'Filling typed holes with live GUIs'. In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. PLDI 2021. New York, NY, USA: Association for Computing Machinery, June 2021, pp. 511–525. DOI: 10.1145/3453483.3454059. (Visited on 12/10/2021) (cited on page 13).

[190] Arnold Oostra. *Los gráficos Alfa de Peirce aplicados a la lógica intuicionista*. es. Cuadernos de Sistemática Peirceana 2. Centro de Sistemática Peirceana, 2010 (cited on pages 191, 240).

[191] Arnold Oostra. *Gráficos existenciales Beta intuicionistas*. es. Cuadernos de Sistemática Peirceana 3. Centro de Sistemática Peirceana, 2011 (cited on page 240).

[192] Arnold Oostra. ‘Equivalence Proof for Intuitionistic Existential Alpha Graphs’. In: *Diagrammatic Representation and Inference: 12th International Conference, Diagrams 2021, Virtual, September 28–30, 2021, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2021, pp. 188–195. DOI: 10.1007/978-3-030-86062-2_16 (cited on page 240).

[193] Arnold Oostra. ‘Advances in Peircean Mathematics: The Colombian School’. In: ed. by Fernando Zalamea. De Gruyter, 2022. Chap. Intuitionistic and Geometrical Extensions of Peirce’s Existential Graphs, pp. 105–180 (cited on pages 192, 200, 243).

[194] OpenJS Foundation and Electron contributors. *Process Model*. Version 19. 2022. URL: https://www.electronjs.org/docs/latest/tutorial/process-model (visited on 08/05/2023) (cited on page 86).

[195] OpenJS Foundation and Electron contributors. *What is Electron?* Version 19. 2022. URL: https://www.electronjs.org/docs/latest/ (visited on 08/05/2023) (cited on page 86).

[196] OpenJS Foundation and Node.js contributors. *Node.js HTTP library documentation*. Version 14. 2020. URL: https://nodejs.org/docs/latest-v14.x/api/http.html (visited on 08/05/2023) (cited on page 86).

[197] Jens Otten and Christoph Kreitz. ‘A connection based proof method for intuitionistic logic’. In: *Theorem Proving with Analytic Tableaux and Related Methods*. Ed. by Peter Baumgartner, Reiner Hähnle, and Joachim Possega. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 122–137 (cited on page 242).

[198] Charles Sanders Peirce. *On the Logic of Number*. eng. American Journal of Mathematics, Jan. 1881. (Visited on 01/02/2024) (cited on page 7).

[199] Charles Sanders Peirce. ‘On the Algebra of Logic: A Contribution to the Philosophy of Notation [Continued in Vol. 7, no. 3]’. In: *American Journal of Mathematics* 7.2 (1885). Publisher: The Johns Hopkins University Press, pp. 180–196. DOI: 10.2307/2369451. (Visited on 03/05/2023) (cited on pages 2, 155).

[200] Charles Sanders Peirce. ‘The Logic of Relatives’. In: *The Monist* 7.2 (1897). Publisher: Oxford University Press, pp. 161–217. (Visited on 03/05/2023).

[201] Charles Sanders Peirce. ‘[Lecture I] : autograph manuscript notebook’. In: *Charles S. Peirce papers*. MS Am 1632, (450). Houghton Library, 1903.

[202] Charles Sanders Peirce. ‘Prolegomena to an Apology for Pragmaticism’. In: *The Monist* 16.4 (1906). Publisher: Oxford University Press, pp. 492–546. (Visited on 10/06/2023) (cited on pages 156, 161, 163–165, 183, 188–191).

[203] Ahti-Veikko Pietarinen. ‘The Endoporeutic Method’. In: *The Commens Encyclopedia: The Digital Encyclopedia of Peirce Studies*. Ed. by M Bergman and J Queiroz. New Edition. Pub. 131013-2050a. 2004. (Visited on 10/31/2023). published (cited on page 178).

[204] Luís Pinto and Tarmo Uustalu. ‘Proof Search and Counter-Model Construction for Bi-intuitionistic Propositional Logic with Labelled Sequents’. en. In: *Automated Reasoning with Analytic Tableaux and Related Methods*. Ed. by David Hutchison et al. Vol. 5607. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 295–309. DOI: 10.1007/978-3-642-02716-1_22. (Visited on 08/26/2023) (cited on page 132).

[205] Luís Pinto and Tarmo Uustalu. ‘Relating Sequent Calculi for Bi-intuitionistic Propositional Logic’. en. In: *Electronic Proceedings in Theoretical Computer Science* 47 (Jan. 2011), pp. 57–72. DOI: 10.4204/EPTCS.47.7. (Visited on 03/31/2022) (cited on page 123).

[206] Linda Postniece. ‘Deep Inference in Bi-intuitionistic Logic’. en. In: *Logic, Language, Information and Computation*. Ed. by Hiroakira Ono, Makoto Kanazawa, and Ruy de Queiroz. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 320–334. DOI: 10.1007/978-3-642-02261-6_26 (cited on pages 116, 118, 143, 144, 146).

[207] Linda Postniece. ‘Proof theory and proof search of bi-intuitionistic and tense logic’. en. In: Artwork Size: vii, 228 leaves. The Australian National University, 2010, vii, 228 leaves. DOI: 10.25911/5D5FCC3A4DB33. (Visited on 08/26/2023) (cited on pages 123, 144, 148, 151).

[208] Thomas Raths, Jens Otten, and Christoph Kreitz. ‘The ILTP Problem Library for Intuitionistic Logic: Release v1.1’. en. In: *Journal of Automated Reasoning* 38.1-3 (Feb. 2007), pp. 261–271. DOI: 10.1007/s10817-006-9060-z. (Visited on 12/09/2023) (cited on page 228).

[209] Cecylia Rauszer. ‘A Formalization of the Propositional Calculus of H-B Logic’. In: *Studia Logica: An International Journal for Symbolic Logic* 33.1 (1974). Publisher: Springer, pp. 23–34. (Visited on 08/25/2023) (cited on pages 123, 132, 147).

[210] Cecylia Rauszer. ‘Semi-Boolean Algebras and Their Applications to Intuitionistic Logic with Dual Operations’. In: *Fundamenta Mathematicae* 83 (1974), pp. 219–249 (cited on page 123).

[211] Cecylia Rauszer. ‘Applications of Kripke Models to Heyting-Brouwer Logic’. In: *Studia Logica: An International Journal for Symbolic Logic* 36.1/2 (1977). Publisher: Springer, pp. 61–71. (Visited on 08/25/2023) (cited on page 123).

[212] Greg Restall. *Extending Intuitionistic Logic with Subtraction*. Online Notes. 1997. URL: https://consequently.org/papers/extendingj.pdf (visited on 08/25/2023) (cited on page 135).

[213] Brian Ritchie. ‘The Design and Implementation of an Interactive Proof Editor’. en. Accepted: 2013-04-02T15:18:04Z Publisher: The University of Edinburgh. PhD thesis. 1988. (Visited on 02/12/2024).

[214] Don D. Roberts. *The Existential Graphs of Charles S. Peirce*. Berlin, Boston: De Gruyter Mouton, 1973 (cited on pages 104, 114, 155, 156, 169, 179–181, 184, 245).

[215] Don D. Roberts. ‘The existential graphs’. en. In: *Computers & Mathematics with Applications* 23.6-9 (Mar. 1992), pp. 639–663. DOI: 10.1016/0898-1221(92)90127-4. (Visited on 09/19/2023) (cited on page 161).

[216] Peter J. Robinson and John Staples. ‘Formalizing a Hierarchical Structure of Practical Mathematical Reasoning’. In: *Journal of Logic and Computation* 3.1 (Feb. 1993), pp. 47–61. DOI: 10.1093/logcom/3.1.47. (Visited on 12/08/2021) (cited on pages 22, 35).

[217] Benoit Rognier and Guillaume Duhamel. ‘Présentation de la plateforme edukera’. In: *Vingt-septièmes Journées Francophones des Langages Applicatifs (JFLA 2016)*. 2016 (cited on pages 61, 228).

[218] Kurt Schütte. *Proof Theory*. en. Ed. by S. S. Chern et al. Vol. 225. Grundlehren der mathematischen Wissenschaften. Berlin, Heidelberg: Springer, 1977. (Visited on 12/28/2023) (cited on page 9).

[219] Jan Sebestik. *Logique et mathématiques chez Bernard Bolzano*. Ed. by Vrin. Vrin, 1992 (cited on page 9).

[220] David Sherratt et al. ‘Spinal Atomic Lambda-Calculus’. en. In: *Foundations of Software Science and Computation Structures*. Ed. by Jean Goubault-Larrecq and Barbara König. Vol. 12077. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 582–601. DOI: 10.1007/978-3-030-45231-5_30. (Visited on 05/12/2022) (cited on page 245).

[221] Sun-Joo Shin. *The Iconic Logic of Peirce’s Graphs*. The MIT Press, May 2002 (cited on pages 155, 157).

[222] Shneiderman. ‘Direct Manipulation: A Step Beyond Programming Languages’. en. In: *Computer* 16.8 (Aug. 1983), pp. 57–69. DOI: 10.1109/MC.1983.1654471. (Visited on 01/08/2024) (cited on pages 15, 16).

[223] Thoralf Skolem. ‘Logisch-Kombinatorische Untersuchungen Über Die Erfüllbarkeit Oder Bewiesbarkeit Mathematischer Sätze Nebst Einem Theorem Über Dichte Mengen’. In: *Selected Works in Logic*. Ed. by Thoralf Skolem. Universitetsforlaget, 1920 (cited on page 193).

[224] Robin Smith. ‘Aristotle’s Logic’. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta and Uri Nodelman. Winter 2022. Metaphysics Research Lab, Stanford University, 2022. (Visited on 10/25/2023) (cited on page 177).

[225] John Sowa. ‘Conceptual Graphs for a Data Base Interface’. In: *IBM Journal of Research and Development* 20 (July 1976), p. 336. DOI: 10.1147/rd.204.0336 (cited on page 155).

[226] John Sowa. 'Peirce's Tutorial on Existential Graphs'. In: *Semiotica* 186 (2011), pp. 345–394. DOI: 10.1515/semi.2011.060 (cited on pages 158, 159, 191, 241).

[227] Matthieu Sozeau. *Coq's Reference Manual*. Chap. Generalized Rewriting (cited on page 51).

[228] Matthieu Sozeau et al. 'The MetaCoq Project'. In: *Journal of Automated Reasoning* (Feb. 2020). DOI: 10.1007/s10817-019-09540-0 (cited on pages 87, 96).

[229] Richard Statman. 'Intuitionistic propositional logic is polynomial-space complete'. In: *Theoretical Computer Science* 9.1 (1979), pp. 67–72.

[230] Jonathan Sterling. 'First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory'. In: (Apr. 2022). DOI: 10.1184/R1/19632681.v1 (cited on pages 210, 242).

[231] Jonathan Sterling and Robert Harper. *Algebraic Foundations of Proof Refinement*. en. Number: arXiv:1703.05215 arXiv:1703.05215 [cs]. Mar. 2017. URL: http://arxiv.org/abs/1703.05215 (visited on 09/10/2022) (cited on page 242).

[232] M. H. Stone. 'Topological representations of distributive lattices and Brouwerian logics'. cs. In: *Časopis pro pěstování matematiky a fysiky*. Vol. 067. ISSN: 1802-114X Issue: 1 Journal Abbreviation: Časopis Pěst. Mat. Fys. 1938, pp. 1–25. DOI: 10.21136/CPMF.1938.124080. (Visited on 11/08/2023) (cited on page 191).

[233] Lutz Straßburger. 'The problem of proof identity, and why computer scientists should care about Hilbert's 24th problem'. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 377.2140 (Mar. 11, 2019), p. 20180038. DOI: 10.1098/rsta.2018.0038. (Visited on 08/08/2020) (cited on pages 58, 99, 109).

[234] Ivan E. Sutherland. 'Sketchpad: A Man-machine Graphical Communication System'. In: *Proceedings of the SHARE Design Automation Workshop*. DAC '64. New York, NY, USA: Association for Computing Machinery, 1964, pp. 6.329–6.346. DOI: 10.1145/800265.810742 (cited on page 235).

[235] Enrico Tassi. *Coq-Elpi GitHub repository*. 2023. URL: https://github.com/LPCIC/coq-elpi (visited on 08/05/2023) (cited on page 87).

[236] Agda team. *The Agda Wiki*. 2023. URL: https://wiki.portal.chalmers.se/agda/Main/HomePage.

[237] The Coq Development Team. *coq-core OCaml API documentation*. Version 8.15. 2022. URL: https://coq.github.io/doc/v8.15/api/coq-core/index.html (visited on 08/05/2023) (cited on page 87).

[238] The Coq Development Team. *The Coq Proof Assistant*. Version 8.16. 2022. DOI: 10.5281/zenodo.7313584 (cited on page 12).

[239] The Coq Development Team. *Ltac2*. https://coq.inria.fr/refman/proof-engine/ltac2.html. 2023. (Visited on 09/05/2023) (cited on page 96).

[240] The Vue.js team. *Vue.js*. Version 2. 2022. URL: https://v2.vuejs.org/v2/guide/ (visited on 08/03/2023) (cited on page 83).

[241] Alwen Tiu. 'A Local System for Intuitionistic Logic'. en. In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Ed. by Miki Hermann and Andrei Voronkov. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 242–256. DOI: 10.1007/11916277_17 (cited on pages 125, 127, 158, 160, 171, 209, 243).

[242] Andrea Aler Tubella and Lutz Straßburger. 'Introduction to Deep Inference'. École thématique. Lecture. Riga, Latvia, Aug. 2019 (cited on pages 59, 125, 170, 172).

[243] Sebastian Ullrich and Leonardo de Moura. 'Beyond Notations: Hygienic Macro Expansion for Theorem Proving Languages'. In: *arXiv:2001.10490 [cs]* (Apr. 2022). arXiv: 2001.10490. (Visited on 04/14/2022) (cited on page 13).

[244] Unknown author. *Visual Logic: Peirce's Existential Graphs*. Minds and Machines. Oct. 2001. URL: https://www.youtube.com/watch?v=j2s5CVxQu7I (visited on 12/10/2023) (cited on page 231).

[245] Igor Urbas. 'Dual-Intuitionistic Logic'. In: *Notre Dame Journal of Formal Logic* 37.3 (July 1996). Publisher: Duke University Press, pp. 440–451. DOI: 10.1305/ndjfl/1039886520. (Visited on 08/25/2023) (cited on page 123).

[246] Jérôme Vouillon and Vincent Balat. 'From bytecode to JavaScript: the Js_of_ocaml compiler'. en. In: *Software: Practice and Experience* 44.8 (Aug. 2014), pp. 951–972. DOI: 10.1002/spe.2187. (Visited on 01/06/2023) (cited on page 83).

[247] Rolin Wavre. 'Y a-t-il une crise des mathématiques? À propos de la notion d'existence et d'une application suspecte du principe du tiers exclu'. In: *Revue de Métaphysique et de Morale* 31.3 (1924), pp. 435–470. (Visited on 01/01/2024).

[248] Karl Weierstraß. 'Über continuirliche functionen eines reellen arguments, die für keinen werth des letzteren einen bestimmten differentialquotienten besitzen'. gelesen in der Königl. Akademie der Wissenschaften am 18 Juli 1872. In: Berlin: Mayer & Müller, 1895, pp. 71–74 (cited on page 4).

[249] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, 1925 (cited on page 2).

[250] Freek Wiedijk. *The De Bruijn Factor*. Online Notes. 2000. URL: http://www.cs.ru.nl/F.Wiedijk/factor/factor.pdf (cited on page 12).

[251] Wikipedia contributors. *Alfred Kempe*. en. Page Version ID: 1178258486. Oct. 2023. URL: https://en.wikipedia.org/w/index.php?title=Alfred_Kempe&oldid=1178258486 (visited on 10/23/2023) (cited on page 184).

[252] Wikipedia contributors. *Four color theorem*. en. Page Version ID: 1178219709. Oct. 2023. URL: https://en.wikipedia.org/w/index.php?title=Four_color_theorem&oldid=1178219709 (visited on 10/23/2023) (cited on page 184).

[253] Wikipedia contributors. *Node graph architecture — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Node_graph_architecture&oldid=1192030043. [Online; accessed 30-January-2024]. 2023.

[254] Wikipedia contributors. *Sequence diagram*. https://en.wikipedia.org/w/index.php?title=Sequence_diagram&oldid=1153944336. 2023. (Visited on 07/05/2023) (cited on page 87).

[255] Ludwig Wittgenstein. *Tractatus Logico-Philosophicus*. Ed. by Project Gutenberg. Project Gutenberg, 2010. (Visited on 08/14/2023) (cited on page 104).

[256] Yongwei Yuan et al. 'Live Pattern Matching with Typed Holes'. en. In: *Proceedings of the ACM on Programming Languages* 7.OOPSLA1 (Apr. 2023), pp. 609–635. DOI: 10.1145/3586048. (Visited on 11/14/2023) (cited on page 201).

[257] Fernando Zalamea. *Lógica topológica. Una introducción a los gráficos existenciales de Peirce*. Memorias del XIV Coloquio Distrital de Matemáticas y Estadística. 1997 (cited on page 192).

[258] Fernando Zalamea. *Pragmaticismo, gráficos y continuidad. Hacia el lugar de C. S. Peirce en la historia de la lógica*. Mathesis 13, pp. 147–156. 1997 (cited on page 192).

[259] Fernando Zalamea. 'Peirce's logic of continuity: Existential graphs and non-Cantorian continuum'. In: *Review of Modern Logic* 9.1-2 (Jan. 2003). Publisher: The Review of Modern Logic, pp. 115–162. (Visited on 12/16/2021) (cited on page 192).

[260] Fernando Zalamea. 'Advances in Peircean Mathematics: The Colombian School'. en. In: *Advances in Peircean Mathematics*. De Gruyter, Nov. 2022. DOI: 10.1515/9783110717631. (Visited on 11/10/2023).

[261] J.J. Zeman. 'The Graphical Logic of C. S. Peirce'. PhD thesis. University of Chicago, 1964 (cited on page 156).

[262] Bohua Zhan et al. 'Design of Point-and-Click User Interfaces for Proof Assistants'. In: *Formal Methods and Software Engineering: 21st International Conference on Formal Engineering Methods, ICFEM 2019, Shenzhen, China, November 5–9, 2019, Proceedings*. Shenzhen, China: Springer-Verlag, 2019, pp. 86–103. DOI: 10.1007/978-3-030-32409-4_6 (cited on page 36).

Concept Index

Here are the introduced concepts in alphabetical order.

λ-calculus, 11, 12, 67, 110, 194, 233, 243–245
λ-term, 11, 12, 59, 67, 244
n-ary scroll, vii, 191–194, 197, 209, 231, 241

accessibility, 209, 210, 221, 223
action, viii, 15, 16, 18, 19, 21, 25–33, 35, 36, 38–41, 44, 45, 52–54, 60–63, 65–69, 71–73, 75, 77, 78, 81, 83, 84, 86, 87, 91, 93–96, 98, 100, 101, 103, 104, 108, 109, 113–115, 117, 159, 180, 188, 232–239, 242, 243, 245
    Auto, 235, 236
    Case, 233, 236, 238, 239
    Compute, 68
    contextual action, 66–68, 71, 93, 95, 99, 113
    Crop, 238
    Delete, 238
    EFQ, 233, 238
    Fence, 234, 245
    Glue, 238
    Grow, 238
    Import, 233–239, 243
    Induction, 71, 95, 99
    Insert, 238
    Justify, 233–235, 238, 239
    Pull, 238
    QED, 233, 238, 239
    Redo, 232
    Search, 67, 72
    Simplify, 68, 72, 73, 95
    Undo, 232, 234, 235
    Unfold, 66–68, 93, 95
    Unlock, 233, 238, 239
admissible, 9, 20, 54, 117, 144, 146, 150, 156, 175, 181, 200, 201, 205, 234, 241
analytic, vi, 9, 20, 54, 146, 156, 160, 163, 169, 176, 181, 187, 188, 201, 209, 217, 219, 222, 223, 231, 232, 241, 242, 245
anchor, 224, 225
argument, 224, 225
axiom, 2, 3, 5, 8, 12, 30, 66, 68, 72, 169, 178, 181

backward, 5, 9, 14, 19, 27, 30, 31, 33, 37, 39, 43–45, 48–51, 53, 58, 69, 72, 78–80, 96, 104, 116, 119, 125, 191, 199
Barendregt convention, 26
behavioral, 66, 68, 72
bi-intuitionistic, 19, 116, 117, 123, 124, 126, 131–135, 143, 144, 147, 148, 151, 154, 209
binder, 178–181, 183, 184, 186, 192, 193, 195, 197, 199, 201, 214, 241
body, 65, 66
Boolean algebra, 134, 135, 139, 140, 143, 249, 252
bound variable, 197
bounded lattice, 133–135
bouquet, viii, 195–201, 203–205, 211, 214, 218, 219, 224, 225, 228, 232, 234
branching operator, 119, 124, 127, 138, 139
Brouwer algebra, 134, 135, 140
bubble, vi, vii, 19, 103–105, 113–116, 118, 119, 121–127, 130–132, 152, 154, 161, 209, 226
bubble calculus, vi, vii, 19, 21, 103, 106–108, 111, 113, 116–118, 123, 127, 130, 144, 146, 148, 149, 159, 178, 242

calculus of inductive constructions, 12, 84
calculus of structures, 9, 10, 22, 59, 97, 125, 155, 156, 166, 169–171, 242–244
calculus ratiocinator, 2
capture-avoiding, 198, 212, 215
category theory, 8, 17, 99, 155, 178, 242
characteristica universalis, 2, 17
chemical abstract machine, 103
classical, vii, 4, 6–8, 19, 42, 51, 58, 59, 75–81, 116, 118, 120–123, 126, 131, 133, 134, 140, 143, 147, 151, 155–157, 160, 167, 169, 178, 189, 190, 192, 194, 199, 200, 209, 217, 221, 224, 228, 230, 245
co-exponential, 134
coherent formula, 187, 193, 241
compatible, 225
computational, 68, 72
computational reflection, 96
consistency, 7, 96
context, 8–10, 38–40, 42, 45–49, 51–53, 66–69, 71, 79, 96, 103, 108, 109, 114, 115, 118, 121, 125, 126, 136, 140, 141, 144, 146, 150, 152, 154, 158, 159, 161, 162, 165–168, 170–173, 176, 180, 186, 196, 198–202, 204, 205, 213, 214, 220, 223–226, 233, 235, 236, 241, 250

contraction, 8, 53, 58, 108, 113, 126, 130, 148, 150, 151, 153, 154, 159, 174, 241, 243
corolla, 195, 196, 200, 201, 234
curl, vii, 191
cut, 157–162, 164–166, 169, 173–176, 178–180, 183, 184, 186, 188, 190–192, 194, 199, 209
cut-elimination, 9, 132, 144, 209, 217, 243

declarative, 14, 100, 236
decomposition phase, 226, 227, 235
deep embedding, 96, 100
deep inference, 9, 10, 16, 18, 19, 22, 38, 54, 82, 96–98, 100, 109, 116, 123, 125, 126, 144, 150, 155, 158, 160, 169, 187, 199, 209, 212, 223, 224, 241–244
depth, 45, 138, 141, 158, 159, 162, 164, 166, 170, 173, 196, 198, 235
diagram, 17, 18, 87, 99, 103, 104, 133, 155, 156, 190, 194
direct manipulation, 15, 16, 18, 19, 22, 77, 116, 126, 146, 183, 187, 194, 231, 232
double-negation elimination law, viii, 146, 228
drag-and-drop, 18, 25, 28–31, 33, 36, 60, 62, 79, 199, 242
dual-intuitionistic, 19, 116, 123, 124, 126, 131, 133, 134, 143, 146, 151

elaboration, 12, 13, 19
endoporeutic, 178, 186
entitative graph, 9, 155, 158, 188
exchange, 8, 53, 54
exclusion, vii, 123, 124, 134, 140, 143, 146, 148, 150
existential graph, 18, 19, 21, 104, 114, 132, 155, 156, 187, 188, 192
exponential, 134

first-order, vi, 11, 19, 26, 27, 35, 40, 59, 65, 66, 84, 92, 94, 96, 176, 178, 180, 193, 195, 204, 218, 241, 243, 245
first-order signature, 195
first-order structure, 210, 221
flower calculus, viii, 20, 155, 187, 188, 198, 199, 201–205, 208, 209, 211, 217, 218, 222, 223, 231, 232, 241, 242, 245
focusing, 38, 40, 51–53, 81, 125, 241
forcing, 210, 211, 218, 221, 244
forward, 5, 19, 27, 28, 31–33, 37, 39, 43–45, 48–51, 53, 61–63, 67, 69, 78, 79, 81, 96, 104, 114, 116, 119, 121, 125, 190, 191, 199
free variable, 196, 197, 210

garden, 184–186
geometric logic, 193
goal, vi, 5, 10, 13–16, 18–20, 22, 26–29, 31, 33, 34, 40, 41, 44, 51–53, 61, 66–68, 75–78, 82–84, 87, 91, 93, 94, 96–100, 103, 105, 108, 114, 115, 118, 125, 126, 132, 152, 153, 156, 160, 162, 187, 191, 193, 199, 203, 225, 226, 231–237, 241, 242, 245
graph, 157–168, 170, 172, 173, 176–181, 183–186, 240
graphical user interface, 16, 35
Grishin (a), vii, 120–122
Grishin (b), vii, 120–122

head, 65, 66
Heyting algebra, 79, 134, 135, 140, 143, 210
Heyting-Brouwer algebra, 134, 135, 140, 143, 145, 148
higher-order, 11, 65–67, 82, 91, 97, 98, 156, 204
higher-order logic, 67
Hilbert system, 2, 3, 5–8, 12, 155
hole, 9, 38, 39, 42, 45, 51, 108, 136, 162, 165, 170, 172, 198
hook, 177–180, 183, 184

icon, 17–20, 22, 116, 117, 125, 127, 154, 155, 157, 158, 177–179, 187–189, 191, 193, 194, 231, 242
illative atomicity, 156, 163–166, 169, 190, 240
illative transformation, 156, 158, 164, 176, 183, 188, 191, 245
  Blank Antecedant, vii, 189–191, 199
  Deiteration, 159, 160, 164, 182, 185, 186
  Deletion, 156, 160, 181–183, 186, 200
  Double-cut, 156, 160, 164, 165, 182, 190, 191, 193
  Insertion, 156, 159, 160, 180–182, 186, 190, 200
  Iteration, 156, 159, 160, 164–166, 182, 185
imperative, 13, 14, 16, 18, 100, 236
inference rule, vii, 3, 5, 9, 14, 16, 19, 26, 29, 36, 51, 76, 103, 105, 106, 108, 110, 116–118, 125, 126, 144, 150, 156, 158, 159, 162, 163, 170, 171, 175, 187, 198, 241, 246
  Flow, 107, 108, 111, 113, 121, 123, 126–132, 145, 148–150, 152–154, 159
  Heating, 106, 107, 109, 111, 113, 114, 123, 128–130, 148–151, 153, 154
  Identity, 106, 107, 111, 126, 128–130, 137,

149
MEMBRANE, 107, 108, 111, 113, 128–130, 132, 137, 145, 149
RESOURCE, 106, 107, 111, 126, 128–130
axiom, 8, 27, 28, 78, 108, 117, 159
cultural, 20, 201, 217
cut, 8, 54, 108, 114, 146, 157, 159, 160, 217, 234
elimination rule, 6, 8, 29, 126, 200
identity rule, 8, 28, 106, 241
introduction rule, vi, vii, 5, 6, 8, 29, 54, 62, 75–78, 103, 104, 106, 108, 116–118, 126, 146, 150, 151, 153
left introduction rule, vii, 8, 29, 54, 78, 104, 108, 117, 154, 200
natural, viii, 20, 201, 204, 208, 215, 216, 240, 241
propagation rule, 144–146, 150
structural rule, 8, 53, 54, 59, 106
switch rule, 9, 10, 48, 126, 174
inloop, 191–194
integrated development environment, 84
interaction operator, 19, 27, 40, 45–47, 49, 75, 78, 79, 199
interpretation, 210, 221
asymmetric interpretation, 106, 118, 119
negative interpretation, 124, 140, 142, 143
positive interpretation, 124, 140
symmetric interpretation, 120, 121, 124, 125
intuitionistic, vi–viii, 4, 6–9, 12, 18–20, 25, 29, 42, 51–54, 58, 59, 75–79, 81, 103, 106, 113, 116–118, 120–124, 126, 131, 133, 134, 143, 146, 147, 151, 154, 160, 166, 171, 179, 187, 190–194, 199, 201, 204, 209, 210, 219, 223, 224, 228, 229, 231, 241–244
intuitionistic existential graph, 187
intuitionistic first-order logic, 82
invertible, vii, 10, 19, 20, 46, 51, 52, 76, 78, 81, 116–118, 121, 125, 130, 148–151, 167, 187, 188, 200, 201, 204, 223, 234, 235, 240, 241, 245
ion, 104, 105, 111, 124, 161
isotropy, 158, 161, 175
item, 26–29, 31, 34, 39–41, 44, 45, 54, 62, 67, 71, 75, 78, 93, 94, 103, 104, 108, 113–115, 121–124, 130–132, 136, 144, 151

judgment, 11, 12, 58, 66, 68, 119, 204, 242
justification, 224, 225
juxtaposition, 157, 158, 180, 185, 188, 240

kernel, 12, 82, 84, 87, 91
Kripke structure, 209–211, 213–215, 218, 223

law of excluded middle, 3, 4, 6, 75, 116, 120, 224
law of non-contradiction, vi, 3, 6–8
line of identity, vii, 156, 165, 177, 179, 184, 199, 246
linkage, vi, vii, 18, 38–44, 47, 48, 50, 53, 54, 58, 59, 72, 79–81, 84, 93, 94, 96, 104, 113, 114
logical linkage, 40–42, 44, 47, 51, 80
rewrite linkage, 40–44, 80
Martin-Löf type theory, 12, 66, 179
mathematical logic, 2, 3, 125
metaphor, 17–20, 41, 103–105, 113–116, 118, 121, 127, 160, 179, 180, 184, 187, 194, 199, 226
mix operator, 120
model theory, 3, 123, 159, 181
modus ponens, viii, 5, 8, 217, 227

natural deduction, vi, 5–8, 10, 12, 14, 26, 51, 77, 110, 125, 126, 134, 157, 209, 217
negative, 39–42, 47, 53, 75, 79, 80, 104, 105, 123, 124, 130–132, 146, 151, 153, 154, 157–160, 162, 166, 168, 173, 178, 180, 193, 194, 199, 203, 204, 212, 223–226, 228, 234, 235, 238
nested sequent, vii, 19, 78, 103, 105, 117, 118, 121, 124, 143, 145, 146, 154, 187, 244, 245
neutral, 123, 124, 127, 130–132, 144, 148, 150, 151
node, 161, 185, 186, 198
nominal, 65, 66, 68

occurrence, 224, 234, 242, 244, 245
outloop, 191–194

parallel, vi, 19, 79–81
path, 39, 52, 58, 94–96, 100, 224
petal, viii, 195, 197–201, 209, 219–221, 226, 233–235, 237, 240, 241, 244
pistil, 195, 197, 199–201, 219–221, 233, 234, 237, 240, 243
polarity, 19, 38–42, 46, 53, 79–81, 116, 123, 124, 130–132, 144, 150, 154, 158–162, 166, 169–171, 173, 192, 194, 198, 203, 234

pollination, vii, 150, 199, 200, 205, 209, 213, 223, 233, 237, 242
cross-pollination, 199, 205, 214
self-pollination, 199, 205, 214
pollination phase, 225, 227–229, 235, 242
positive, 10, 39–41, 47, 53, 71, 77, 79–81, 104, 105, 110, 111, 122–124, 130–132, 151, 153, 154, 159, 160, 162, 166, 168, 170, 172, 173, 178, 180, 181, 193, 198, 199, 203, 212, 223–225, 234, 235, 237, 238
predicate logic, 2, 9–11, 25, 29, 54, 65, 66, 77, 78, 97, 104, 126, 134, 165, 176, 180, 181, 183, 187, 201, 204, 241
productivity, 38, 40, 50, 51, 58
proof assistant, 1, 3, 10–15, 20, 22, 23, 25, 30, 35, 36, 38, 41, 51, 62, 63, 65–68, 71, 75, 76, 78, 82, 97, 98, 100, 103, 115, 156, 181, 201, 231, 232, 236, 237, 244
proof canvas, 27, 113, 115
proof evolution, 17, 19, 83, 100, 245
proof formalism, 1, 3, 4, 9–11, 18, 19, 22, 25, 65, 76, 77, 103, 106, 109, 116, 117, 126, 155, 158, 163, 169, 187, 209, 217, 223, 242
proof script, vi, 13, 60, 62, 64, 73, 82, 84, 86, 94, 97, 99–101, 236
proof state, 13, 14, 22, 26, 36, 68, 76, 83, 84, 103, 105, 108, 110, 113, 114, 118, 125
proof term, 12, 19, 83, 84, 237, 242, 245
proof theory, 3, 5, 7–11, 14, 18, 20–22, 25, 35, 38, 40, 60, 82, 99, 109, 114, 123, 125–127, 158, 187, 194, 235
proof view, 13, 14, 19, 82, 84, 86, 156, 232, 236
proof-by-action, 16, 25, 60, 82, 103, 109, 113, 146, 150, 188, 231
proof-by-linking, 16
proof-by-pointing, 16, 25, 30, 35, 103, 126
proofs-as-programs, 12, 15, 33, 53, 99, 110, 123, 144, 201, 237, 243, 244
propositions-as-types, 12

reproduction phase, 226, 227, 235
rewriting rule, 1, 9, 16, 28, 40, 43–48, 50–53, 59, 78, 79, 96, 103, 109, 110, 114, 117, 132, 158, 161, 199
rewriting system, 9, 10, 19, 38, 45, 97, 109, 156, 187, 199

saturated, 116, 119–121, 123, 124, 127, 130, 132, 144, 148, 150–152, 156
scroll, vii, 187, 189–191, 219, 240
semiotics, 17
sequence diagram, 87
sequent, 8–10, 13, 18, 19, 25, 26, 53, 54, 75–78, 80, 103–105, 117, 121, 125, 126, 134, 144, 195, 204
sequent calculus, vi, 7–10, 14, 16, 19, 26, 29, 36, 53–55, 59, 63, 75–78, 103, 104, 106, 109, 110, 116–118, 121, 125, 126, 130–132, 135, 138, 146–148, 155, 157, 159, 160, 163, 166, 200, 204, 205, 209, 217, 218, 234, 241, 243
set theory, 2, 4, 7, 12, 65, 66, 123
shallow, 202, 215
sheet of assertion, 127, 156, 188
single-conclusion, 105, 106, 116, 118, 121
solution, 103, 105, 106, 108, 109, 113, 114, 116, 118–121, 123–127, 130–132, 136, 139, 144, 146, 148, 150, 151, 154, 156, 157, 161
spot, 177, 178, 180, 181, 184
sprinkler, 184, 195, 196, 200, 201, 223, 231, 243, 245
structural proof theory, 3, 12
structure, 144, 170, 172, 175
subformula linking, 16, 18, 21, 38, 40, 51–55, 58, 59, 95–97, 100, 103, 108, 113, 152, 199, 242, 245
subformula property, 9, 54, 146, 163, 169, 176, 223
subgoal, 5, 19, 27, 29, 31, 62, 63, 71, 72, 76, 83, 86, 87, 91, 93–95, 103–106, 108, 113–115, 118–120, 125, 127, 130–132, 148, 150–152, 233–237
subgraph, 158, 159, 172, 173, 176
subsolution, 109, 110, 124, 130, 144, 151–153
substitution, 38, 42–44, 47, 48, 58, 196–198, 200, 202, 210, 215, 220–222, 246
support, 197
symbol, 1, 3, 4, 7, 13, 15, 17–22, 27, 66, 104, 123, 125, 127, 154–157, 176, 178, 179, 183, 184, 187, 189–191, 201, 209, 228, 237, 242

tactic, 13, 14, 16, 18, 30, 31, 51, 61–63, 68, 69, 73, 76, 77, 83, 84, 86, 87, 91, 93–97, 100, 101, 236, 241, 242
teridentity, vii, 178–181, 183, 184
term, 11–13, 67, 84, 96, 98, 99,

243, 244
theory, 193, 195, 196, 218–222
type, 11, 12, 33, 67, 71, 96, 164, 179, 244, 245
type theory, 11–13, 26, 34, 67, 68, 96, 164, 195, 204, 237, 241, 242, 244, 245

unifiable, 18, 35, 40, 42–44, 58, 84, 96, 98, 100, 125, 181
unifiable variable, 42, 43
universal Kripke structure, 221
unsaturated, 116, 119, 121, 123, 124, 127, 130, 132, 144, 148, 151, 153
update, 197
user experience, 38

valid, 18, 40, 41, 43, 44, 47, 48, 50, 54, 58, 59, 72, 79, 80, 84, 93, 188, 211, 219, 220, 228
    classically valid, 80
valuation, 167–169, 172
vehicle, 224, 225, 242

weakening, 8, 53, 58, 108, 113, 130, 148, 159, 160, 174, 241
window inference, 22, 35, 36
wire, vii, 179–181, 183, 184
world, 209–211, 213–215, 218, 221, 223

zoomable user interface, 114, 235

Acronym Index

Here are the introduced acronyms in alphabetical order.

CHAM, 103, 104, 109

CoS, 9, 16, 18, 59, 108, 109, 126, 130, 170, 172

DnD, vi, 18, 21, 25, 27, 29, 31, 34, 36, 38–41, 44, 51–54, 59–63, 67, 69, 72, 75, 77, 78, 81, 84, 93–97, 100, 103, 104, 113, 114, 152, 199, 233, 235, 238, 243

DNE, vii, 146, 147

EG, vii, 18–20, 155, 156, 158–160, 164–166, 169, 176, 178, 179, 181, 183, 184, 187–189, 191, 194, 195, 198–201, 203, 209, 215, 231, 240, 241, 243–246

FOL, 11, 12, 16, 25, 26, 52, 91, 96, 100, 105, 117, 178, 193

GUI, 16, 18–20, 97, 103, 109, 113, 151, 187, 188, 231, 235

HOL, 35, 67

IDE, 84, 86, 101, 236

IEG, vii, 187, 191, 193, 200, 240

iFOL, 82–84

ITPs, 1, 13–16, 19, 20, 22, 82–84, 98, 99, 187

LEM, 3

LoI, 177–181, 183–186, 199, 200, 215, 240, 241

PbA, 16–18, 20, 26, 60, 63, 65, 66, 68, 71, 72, 82, 98, 100

PbL, 16, 18

PbP, 16, 18, 30

SFL, 16, 96, 114, 181

UX, 38, 97, 103, 126, 183

ZUI, 114

Proof System Index

Here are the introduced proof systems in alphabetical order.

Alpha, vii, 156–163, 165–167, 169–173, 175–181, 186, 190, 191, 198, 200, 240
Alphaa, vii, 165, 166

Beta, vii, 156, 165, 176–186, 192, 195, 198–200, 240
Binv, vii, 148–151, 154, 159, 225
BJ, vi, vii, 19, 106–114, 117, 121, 125–127, 130–132, 163
BV, 22, 224

cISgq, 126
CoIC, 12, 96

DBiInt, vii, 116, 143–148, 150, 151, 153, 255
dualKS, 175, 176

FILL, 120, 124

G3c, 78, 81
G3cp, 147, 258
G3i, 54, 55
G3IntQ, 223
G4ip, 241
Gamma, 156
GHCP, 76, 77

HS, 241

IMLL, 120
ISgq, 126
ISp, 158, 160, 243

JN, vii, 154

KS, 175, 176

LBiInt, 143, 144
LJ, vi, 7–9, 14, 19, 29, 75, 77, 106, 113, 116
LJT, 117, 126, 241
LK, vi, 8, 75–77, 116, 126
LL, 78

MLL, 109, 120
MLTT, 12, 68

NJ, vi, 6, 12, 29
NK, 6

sBIL, 147, 148
SKS, vii, 10, 156, 166, 169–172, 174, 175, 243
SKSg, 169
sysBB, 140, 143
sysBH, 140, 143, 144, 146, 147, 258
sysBHB, 140, 143–146, 255
system B, vii, 19, 116, 117, 127–133, 135, 136, 139, 140, 143, 144, 146–148, 150, 151, 154, 159, 160, 250
system L, 116

wBIL, 147, 148

ZF, 2, 65, 66
ZFC, 2

Software System Index

Here are the introduced software systems in alphabetical order.

SSReflect, 77, 96, 100

Actema, vi, 18, 19, 25–27, 29, 30, 39, 40, 52–54, 58, 60–63, 65, 67, 69, 71, 75, 77, 78, 82–87, 91, 93, 94, 96–100, 103, 104, 114, 115, 231–233, 237
Agda, 12, 14, 201, 237, 244
Alf, 12
ATD, vi, 91–93, 95, 98
atdgen, 93
Automath, 11, 12

Base64, 87
Biniou, 93

Chromium, 86
Coq, vi, 12, 14, 19, 35, 51, 58, 60, 62–64, 66–74, 77, 82–87, 91, 94–101, 209, 232, 235–237, 244
coq-actema, vi, 19, 20, 82–90, 97–101
coq-core, 87, 98
Coq-Elpi, 87
coq-lsp, 101
CSS, 83

DOM, 93

EasyCrypt, 83
Edukera, vi, viii, 61, 64, 228, 230
Electron, 86
Elm, 231

F*, 83
Flower Prover, viii, 20, 187, 188, 231–233, 235–239, 242–244

HOL Light, 77
HTML, 25, 83, 93, 98, 99, 231
HTTP, 85–87, 91, 93, 99

Iris, 58
Isabelle, 14, 68, 83, 97, 232
Isar, 14, 36, 60, 68, 236

JavaScript, 25, 83, 99, 231
js_of_ocaml, 83, 84
jsCoq, 84

Larus, 193
Lean, 12–14, 22, 68, 71, 77, 83, 84, 97–100, 232, 236, 244
LSP, 101
Ltac, 96
Ltac2, 96

Meta-F*, 83
MetaCoq, 87, 96
Mizar, 12, 14

Node.js, 86
NuPrl, 12

OCaml, 83, 87, 91, 93, 98, 228

Profint, 58, 97, 113
ProofWidgets, 13, 22, 27, 84, 98, 99

SerAPI, 84
Sketchpad, 235
Smalltalk, 235

UML, 87

VsCoq, 86, 101
Vue.js, 83

Why3, 83

XML, 84