PHILOSOPHICAL TRANSACTIONS A

rsta.royalsocietypublishing.org

Research

Article submitted to journal

Subject Areas:

xxxxx, xxxxx, xxxx

Keywords:

xxxx, xxxx, xxxx

Author for correspondence:

Lutz Straßburger

The Problem of Proof Identity, and why computer scientists should care about Hilbert's 24th problem

Lutz Straßburger

Inria

In this short overview article I will discuss the problem of proof identity and explain how it is related to Hilbert's 24th problem. I will also argue that not knowing when two proofs are "the same" has embarrassing consequences not only for proof theory but also for certain areas of computer science where formal proofs play a fundamental role, in particular the formal verification of software. Then I will formulate a set of 4 objectives that a satisfactory notion of proof identity should obey. And finally, I discuss Hughes' *combinatorial proofs* and argue that they can be seen as a first step towards a possible solution to the problem of proof identity.

1. Introduction

(a) From Hilbert's 24th problem to the identity of proofs

When David Hilbert prepared his lecture [Hil00] in 1900 in which he presented his now famous 23 problems, he also considered a 24th problem [Thi03], as he wrote in his notebook [Hil]:

> As 24th problem in my Paris lecture, I wanted to ask the question: Find criteria of simplicity or rather prove the greatest simplicity of given proofs. More generally develop a theory of proof methods in mathematics. Under given conditions there can be only one simplest proof. And if one has 2 proofs for a given theorem, then one must not rest before one has reduced one to the other or discovered which different premises (and auxiliary means) have been used in the proofs: When one has two routes then one must not just go these routes or find new routes, but the whole area lying between these two routes must be investigated…[1]

[1]Translation by the author.

Fast forward 118 years: Proof theory[1] is now a well-established mathematical discipline. However, Hilbert's original problem, as formulated in his notebook, is still a mystery. We have no criteria for "simplicity" of a proof. We have no methods to compare proofs, in fact, we do not even know when two proofs are *the same*.[2] In that respect, proof theory is very different from most other mathematical fields which usually can identify the objects of their interest. For example, in group theory, two groups are *the same* if they are isomorphic, and in topology, two spaces are *the same* if they are homeomorphic.

This is clearly embarrassing for proof theory as a mathematical field; and only a satisfactory notion of proof identity can eventually lead to a satisfactory answer to Hilbert's 24th problem. Furthermore, the lack of understanding of what actually constitutes a mathematical proof has consequences for many other areas in which proof theory is applied, in particular, the foundations of computer science. In fact, the problem of proof identity is no longer some abstract "ivory tower problem", but it has serious implications for our daily life, as I will argue below.

(b) The relevance for computer science

Many aspects of our modern life depend on computer systems, and the correct functioning of the software running on them should be of utmost importance. Yet we do not trust software and expect it to break, we are used to turning off and back on again our personal computers if they show unexpected behavior, and we just shrug at news about the latest "software bug" that has been exploited by criminals or has caused some major damage. A few decades ago such a "computer bug" might have just been a little annoyance, and even today, a crash of a word processor or a media player is not a life-threatening problem. But now essentially the same devices that run our email-client are built into cars, airplanes, hospital equipment, nuclear power plants, etc. All of the sudden our life depends on software that we do not trust. We connect computers with faulty software to the Internet and send confidential messages over channels where the slightest leak will immediately be exploited. And the situation will only get worse. Soon we connect our heating, our oven, and our fridge to the Internet, to be accessed from distance via software on our phone that we do not trust. What could possibly go wrong?

Maybe the most embarrassing fact about *computer science* is that software companies do not have to take legal liability for the software they ship. This is very different from car manufacturers who have to take legal liability for the cars they make, or building companies who have to take legal liability for the buildings and bridges they build. Is it really so much harder to make correctly functioning software than to make correctly functioning cars? Both are engineering problems, but in the case of cars, we have a clear understanding of the underlying scientific principles, we know the laws of classical mechanics and thermodynamics. But in the case of software, we only have very rudimentary knowledge of the underlying science. A computer program is built from algorithms, but we understand what an algorithm is as much as we understand what a proof is. We know one when we see one, but we have no idea of what it means for two algorithms to be *the same*. Hilbert's question can be reformulated in terms of algorithms: What is the *simplest* algorithm for a given problem.

At the current state of the art, the only way to guarantee that the software at hand is indeed without errors is to use verification tools to obtain a *formal proof*. In fact, according to the ISO/IEC Common Criteria[3], the highest level of assurance (EAL7) for software is achieved by formal mathematical proofs of the correctness.

These proofs can be obtained using automated or interactive theorem provers (or a combination of both). Recently there have been some breathtaking successes in the use of interactive theorem proving. This concerns formalized mathematics, where interactive theorem proving has been used for large proofs of highly computational nature: the Kepler

[1]In this article, the term *proof theory* always refers to the subfield of *structural proof theory*. The areas of *ordinal analysis* and *reverse mathematics* are not discussed here.
[2]Proof normalization will be discussed in Section 2.(a) below.
[3]`http://www.commoncriteriaportal.org/cc/`

Conjecture [Hal05], the 4-Color Theorem [Gon07], and the Odd-Order Theorem [GAA+13], and it also concerns computer science, where interactive theorem proving has been used for formally verifying large components of software, for example, the semantic correctness of a C compiler [Ler09], the functional correctness of the seL4 micro-kernel [KEH+09], and the security properties of the Java Card[4].

However, in order to trust any of these formal proofs, we have to trust the correctness of the tool (i.e., the automated or interactive theorem prover) that has been used to produce the proof. This is because every proof tool comes equipped with a formal proof language that is unique to this tool and cannot be understood by another tool. Furthermore, not only the language, but also the internal representation of proofs differs from one proof tool to another. For example the *Abella* proof system [BCG+14] uses *sequent calculus*, and the *Coq* proof system [BC04,DFH+93] uses *natural deduction*.

A possible approach to overcome this problem is to define richer proof languages that can be used to encode proofs from various different tools, in order to check these proofs independently. An example is the *Dedukti* proof checker [Ded13] which has a rich proof language based on the $\lambda\Pi$-calculus modulo [BCH12,Sai15]. An alternative approach is the use of *proof certificates* [Mil11] that are essentially logic programs encoding a *focused sequent calculus*, and that can be used as a meta-language to describe proofs produced by tools based on the sequent calculus. However, the problem of proof identity remains untouched since these approaches only translate one syntactic representation of a proof into another, and these translations are neither unique nor canonical.

2. Objectives for a universal notion of proof identity

The lack of understanding of the nature of proofs has its cause in the very fundamentals of proof theory: we study formal proofs as syntactic objects that are represented in some *formal proof system*, and we cannot consider the proof independently from that system. In that respect, we can say that *current proof theory is not a theory of proofs, but a theory of proof systems.* In fact, most of the important theorems of proof theory, like soundness, completeness, cut admissibility, focusing, or proof complexity results are not about proofs but about proof systems. All notions of proof identity that exist are between proofs within the same proof system, as it is not clear what it means to compare two formal proofs that are given in two different proof systems.

(a) Existing notions of proof identity

Up to now, there exist only four notions of proof identity, and two of them are trivial: (1) two proofs are the same if they prove the same theorem, and (2) two proofs are the same if they are syntactically equal. They have already been ruled out by Hilbert, and there is no need to discuss them further here. The two non-trivial existing notions of proof identity are based on *normalization* and *generality*, respectively [Doš03].

Normalization can be seen as the standard proof theoretical answer to the problem of proof identity [Pra65]: Two proofs are the same if they have the same normal form. In some respect, this certainly makes sense: Under the Curry-Howard-correspondence [How80], a (natural deduction) proof corresponds to a λ-term, and the normalization of a proof corresponds to the evaluation of a λ-term, and two λ-terms are equal if they evaluate to the same term. This is the basis of the functional programming paradigm. But from the viewpoint of proof search and proof presentation, this notion of equality makes only little sense, since most formal proofs that are considered in formal verification, automated reasoning, and interactive theorem proving are already in normal form in the sense above. Another notion of proof normalization is *cut elimination*, where the *cuts* in a (sequent calculus) proof can be seen as auxiliary lemmas used in that proof. Therefore, these cuts are crucial information and should not be eliminated. An instructive example is Fürstenberg's proof of the infinity of primes, which uses

[4]http://www.gemalto.com/techno/javacard/

a topological argument. When we eliminate the cuts from that proof, we obtain Euclid's original proof [BHL+08].

Generality [Lam68,Lam69] identifies proofs based on coherence theorems for categories seen as deductive systems: two proofs are the same, if they constitute the same morphism in the category. Of course, the problem here is to find the right axiomatisation of the category in question. For intuitionistic propositional logic the notions of coherence (via Cartesian closed categories) and normalization (via β-reduction) yield equivalent notions of proof identity [LS86]. Also for linear logic, the notions of coherence and normalization make the same proof identifications [See89, Blu93,LS06,HS16]. However, for classical logic, the logic of our every-day-reasoning, neither notion has a broadly accepted definition [Par92,BHRU06,Hyl02,FP04,McK06,DP04,Str07b,Str11].

But more importantly, neither notion speaks about the feasibility of deciding proof identity. For practical purposes, however, it is important that it is feasible to check whether two proofs are the same, even if these proofs are huge objects. This leads to our first objective for a universal notion of proof identity:

Objective 1: Proof identity must be decidable in polynomial time in the size of the proofs.

(b) The problem of proof size

A formal proof is usually given within some proof system consisting of axioms and inference rules that allow to construct new proofs from given ones. Here are two examples of such inference rules:

$$\mathsf{mp}\ \frac{A \qquad A \to B}{B} \qquad\qquad \wedge\ \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \tag{2.1}$$

The first says that if we have a proof of A and a proof of $A \to B$ (read "A implies B"), then we can obtain a proof of B. The second says that if we have a proof of A from premises Γ and a proof of B from premises Γ, then we can also prove $A \wedge B$ (read "A and B") from premises Γ.

If we want to automatize the (bottom-up) proof search process, our proof system should have the *subformula property*, which says that every formula that is encountered during the proof search is already present as a subformula in the formula to be proven. For example, the right rule in (2.1) has this property, but the left rule does not. In order to prove B using the rule mp (called *modus ponens*), we have to invent the formula A out of "thin air".

The subformula property of a proof system is usually obtained through the *cut elimination property* (pioneered by Gentzen in his seminal work [Gen35]), which says that the use of auxiliary lemmas can be eliminated from a proof by plugging the proof of the lemma in the place where the lemma was used. Most of the ingenuity of research in proof theory in the last eighty years went into the design of proof systems that do not need a *cut-rule* (which is always some variant of the rule mp above) for completeness.

This marvelous property comes at a price: the transformation of a proof with cuts into a proof without cuts can cause an explosion of the size of the proof. One can construct examples of proofs with cuts that fit on a page, such that eliminating the cuts makes them bigger than the size of the universe [Boo84]. Because of this, the cut is also called a mechanism of *proof compression*. Other such mechanisms are, for example, *extension* and *substitution* [CR79], that allow us to use abbreviations inside a proof, and *co-contraction* (the dual of the usual contraction rule) which allows to share subproofs. Discussing the relation between the various proof compression mechanisms [CR79,KP89,BG09,Str12,NS15,Jeř09,BGGP10] would go beyond the scope of this paper. However, the point I want to stress here is that any reasonable notion of proof identity should take the size of a proof into account:

Objective 2: Proof identity must respect the size of proofs.

In more technical terms this means that whenever two proofs are identified, their size must only differ by a polynomial factor. For propositional classical logic, this is an important issue because

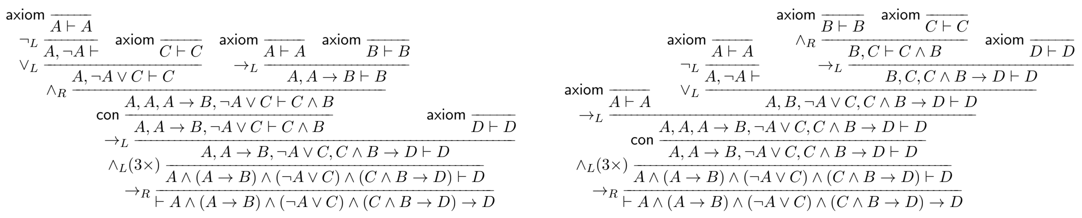


Figure 1. Two sequent calculus proofs of the formula in (2.2)

the question of whether there is a proof system yielding a polynomial size proof of every Boolean tautology is equivalent to the question of whether NP = coNP [CR79].

(c) Partial proofs

For many real-world applications it is not enough to run some automated theorem prover to obtain a proof. Proving a property of a piece of software usually demands interaction of the user who has some knowledge about the system that says why this property should be true. This is where interactive proof assistants, like *Coq* [BC04,DFH$^+$93] or *Isabelle/HOL* [NPW02], are employed which can guide the user to a proof of a given obligation. Obtaining such a proof is often a huge effort, involving many people producing various pieces of the proof. These pieces are represented in form of *proof scripts* that tell the proof assistant how to obtain a proof. Such scripts can in principle also represent incomplete or *partial proofs*. This can be because there is still an open proof obligation for which the proof might be found by using a different tool, or because the missing step is a pure computation step that is redone every time the proof script is checked by the prover. A reasonable notion of proof identity should be able to abstract away from such computation steps, since for the proof it is irrelevant how this computation is done [DHK03]. This leads to:

Objective 3: Proof identity must be able to handle partial proofs.

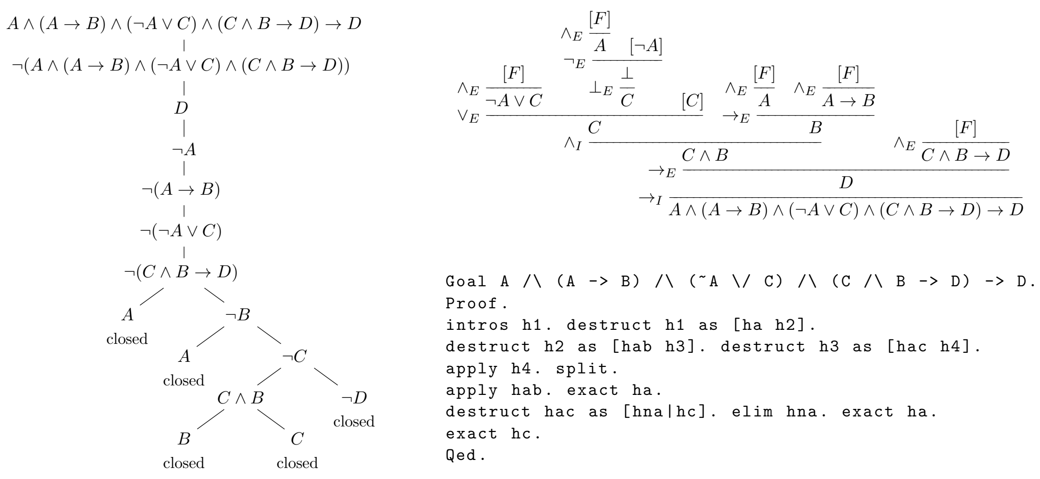
(d) Proofs versus proof representations

We have seen above that the objects of study in proof theory are not proofs but proof systems, and that syntactic proofs are intrinsically tied to the proof systems in which they are carried out. Here I want to argue that the syntactic proofs are just representations of the actual proofs objects. For this, consider as a very simple example the following formula

$$(A \wedge (A \to B) \wedge (\neg A \vee C) \wedge (C \wedge B \to D)) \to D \tag{2.2}$$

Figure 1 shows two (syntactically) different proofs of that formula in Gentzen's sequent calculus LK [Gen35,TS00]. The attentive reader might immediately observe that the two proofs only differ in the order of the application of inference rules and that one can be transformed into the other by a series of simple rule permutation steps. In fact, sequent calculus proofs are often considered equal iff they are equivalent modulo rule permutation. However, recently it has been shown that equivalence of sequent calculus proofs modulo rule permutation can be PSPACE-complete [HH14], which violates Objective 1 stated above and renders it unfeasible to base proof identity on rule permutations (unless P = PSPACE).

To make the situation worse, consider Figure 2, which shows a proof of our formula (2.2) as an analytic tableau [Smu68] on the left, a natural deduction proof [Gen35,Pra65] on the top right, and as a proof script in the Coq system [DFH$^+$93] on the bottom right. A priori, there is no obvious relation between all these proofs, except for the fact that they prove the same formula. However, the natural deduction proof and the Coq proof script are just two different notations of the same



```
Goal A /\ (A -> B) /\ (~A \/ C) /\ (C /\ B -> D) -> D.
Proof.
intros h1. destruct h1 as [ha h2].
destruct h2 as [hab h3]. destruct h3 as [hac h4].
apply h4. split.
apply hab. exact ha.
destruct hac as [hna|hc]. elim hna. exact ha.
exact hc.
Qed.
```

Figure 2. A tableau proof, a natural deduction proof, and a Coq proof of the formula in (2.2)

object. But they are not comparable to the tableau proof on the left or the two sequent proofs in Figure 1 above.[5]

Nonetheless, even though syntactically all these proofs look very different from each other, in principle they all do the same thing: they use A twice to justify B (via the subformula $A \to B$) and C (via the subformula $\neg A \vee C$), which in turn are then used to justify D (via the subformula $C \wedge B \to D$).

So, why not extend the equivalence between the natural deduction proof and the Coq proof script (being just two different notations of the same object) to all proofs in Figures 1 and 2? Why not consider them all simply as different syntactic representations of *the same proof*? The problem here is that the syntactic peculiarities of the chosen formal proof system or proof language hide the essence of the proof, and that it is not clear at all how to give a mathematical description of that proof without relying on the syntax of a proof system. Note that our example only speaks about propositional logic. The situation only gets worse when variable binding, quantifiers and fixpoints enter the scene.

Objective 4: Proof identity must be independent from the syntax of proof systems.

3. From Proof theory to a theory of proofs

In the previous two sections I argued that Hilbert's 24th problem, or the problem of proof identity, is highly relevant for modern computer science, and I discussed some properties that a reasonable notion of proof identity should have, in order to be useful for certain applications. But so far, it is not clear how such a notion could be found and whether it is possible to meet the four objectives discussed above. One might even be inclined to think that the non-existence of our desired notion of proof identity is inherently caused by the very nature of proof theory.

In this section I will argue that the problem is a mathematical problem that can be solved by mathematical means. In fact, there exist already two fundamentally different approaches for freeing proofs from the syntactic yoke of proof systems, and both have the potential to enable us eventually to speak of a theory of proofs, instead of a theory of proof systems.

In the first approach we define the properties of proofs and their interactions via axioms, in a similar way as it is done in abstract algebra. In most cases this leads to some kind of category of proofs [Lam68,Lam69]. Let us call this first approach the *axiomatic approach*. The second approach

[5]Although in propositional logic a tableau proof can be seen as an "upside-down" sequent calculus proof, this is no longer the case for more expressive logics [Gor99]. It seems that we need more sophisticated versions of sequent calculi and tableau systems, in order to recover this upside-down correspondence.

tries to find concrete combinatorial objects (e.g. graphs) that carry the meaning of proofs. Let us call this second approach the *combinatorial approach*.

(a) Towards an axiomatic theory of proofs

As observed by Lambek [Lam68], an algebraic treatment of proofs can be provided by category theory. For this, it is necessary to accept the following postulates about proofs:

- for every proof f of conclusion B from hypothesis A (denoted by $f\colon A \to B$) and every proof g of conclusion C from hypothesis B (denoted by $g\colon B \to C$) there is a uniquely defined composite proof $g \circ f$ of conclusion C from hypothesis A (denoted by $g \circ f\colon A \to C$),
- this composition of proofs is associative,
- for each formula A there is an identity proof $\mathsf{id}_A\colon A \to A$ such that for $f\colon A \to B$ we have $f \circ \mathsf{id}_A = f = \mathsf{id}_B \circ f$.

Under these assumptions the proofs are the arrows in a category whose objects are the formulas of the logic. What remains is to provide the right axioms for the “category of proofs”.

It seems that finding these axioms is particularly difficult for the case of classical logic. Linear logic can be axiomatised by various variants of star-autonomous categories [See89,Blu93,LS06, HS16]. For intuitionistic logic, a reasonable notion of proof identity is given by the axioms of a Cartesian closed category [LS86]. In fact, one can say that the proofs of intuitionistic logic are the arrows in the free Cartesian closed category generated by the set of propositional variables.

A naive approach for extending this result to classical logic would be to add an involutive negation to a Cartesian closed category, i.e., a natural isomorphism between A and the double-negation of A. But this immediately gives us a collapse into a Boolean algebra, i.e., any two proofs $f, g\colon A \to B$ are identified [LS86,Str11]. To overcome this problem, we clearly have to drop some of the equations that one would like to hold between proofs in classical logic. But which ones should go? There are now several different approaches, and all have their advantages and disadvantages.

(i) The first says that the axioms of Cartesian closed categories are essential and cannot be dispensed with. Instead, one sacrifices the duality between $\wedge$ and $\vee$. The motivation for this approach is that a proof system for classical logic can now be seen as an extension of the λ-calculus and the notion of normalization does not change. One has term calculi for proofs, namely Parigot’s $\lambda\mu$-calculus [Par92] and its many variants (e.g., [CH00]), and an important aspect is the computational meaning in terms of continuations [Thi97,SR98] in functional programming languages. For this approach, the category theoretical axiomatisation is well understood [Sel01], and there is a well-behaved theory of proof nets [Lau03]. However, in this setting, proof identity is based on normalization, which does not solve Hilbert’s problem, as discussed in Section 2.(a) above.

(ii) The second approach considers the perfect symmetry between $\wedge$ and $\vee$ to be an essential facet of Boolean logic, that cannot be dispensed with. Consequently, the axioms of Cartesian closed categories and the close relation to the λ-calculus have to be sacrificed. More precisely, the conjunction $\wedge$ is no longer a Cartesian product, but merely a tensor-product. Thus, the Cartesian closed structure is replaced by a star-autonomous structure, as it is known from linear logic. This approach [FP06,LS05a,McK05,Str07b, Lam07,Str11] is much less investigated than the first one above, since there is no immediate correspondence to functional programming constructs. However, it seems to be better suited for Hilbert’s 24th problem.

(iii) The third approach [DP04,CS09] keeps the perfect symmetry between $\wedge$ and $\vee$, as well as the Cartesian product property for $\wedge$. What has to be dropped is the property of being closed, i.e., there is no longer a bijection between the proofs of $A \vdash B \to C$ and the proofs

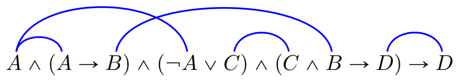

$A \wedge (A \to B) \wedge (\neg A \vee C) \wedge (C \wedge B \to D) \to D$

Figure 3. The matings for the two sequent proofs in Figure 1

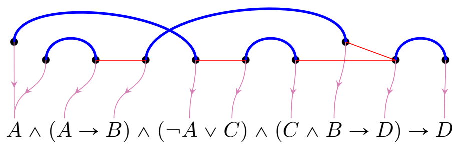

$A \wedge (A \to B) \wedge (\neg A \vee C) \wedge (C \wedge B \to D) \to D$

Figure 4. A combinatorial proof of the formula in (2.2)

of $A \wedge B \vdash C$. This means we lose currying and the categorical version of the deduction theorem.

(iv) Of course, one can also drop even more equations, and choose alternative routes to go, as for example in [Hyl02,Hyl04,BHRU06].

In any case, the fundamental problem with the axiomatic approach is that it always "comes second". We first need a clear understanding of the objects we are investigating before we can find the correct axiomatisations. So did mathematicians settle on the modern axioms for a group only after they had been studying permutations in algebra and symmetries in geometry for decades, and they had for long been investigating various forms of "spaces" before the modern notion of "topology" emerged. For proofs, we are in a similar situation. Only when we fully understand what we mean by "proof", we can start to agree on an axiomatisation.

(b) From syntactic proofs to combinatorial proofs

For this reason, the combinatorial approach seems to be more promising. The first attempt in that direction was through Andrews' *matings* [And76], which kept only the *axiom links* of a proof. Figure 3 shows the matings for the two sequent proofs in Figure 1—both sequent proofs have identical matings. In fact, Andrews considered these matings to be the "essence" of a proof, and thus provided a proposal for a nontrivial notion of proof identity. The more recent work in [LS05b] provided a notion of proof composition for matings.

However, checking correctness of matings takes exponential time[6]. This means that checking correctness of a proof object for a formula is as expensive as proving the formula from scratch. What we need are canonical proof objects whose correctness can be checked in polynomial time.

For (multiplicative) linear logic, this goal was achieved by Girard's *proof nets* [Gir87], where the basic idea is essentially the same as for matings—keep the information about which atom pairs meet in an axiom link. But because of linearity, checking correctness can be done in polynomial time. In order to obtain this property for classical logic, we have to keep track of the places in the proof where duplication and erasure of atoms can happen. The first attempt to do this was through the notion of *atomic flow* [GG08,GGS10], but unfortunately no polynomial-time checking algorithm has been found so far, and it is very likely that it cannot exist [Das13].

An alternative approach is to completely separate in a proof the purely linear part and the duplication/erasure part, as it is done in Hughes' *combinatorial proofs* [Hug06a]. In fact, for the very restrictive case of classical proposition logic, combinatorial proofs meet all four objectives discussed in Section 2. For this reason I will give here more technical details.[7]

[6]The problem is co-NP complete.

[7]In this article, I will follow the presentation in [Str17b,Str17a], and I refer the reader to these papers for more details.

The example in Figure 4 shows the combinatorial proof corresponding to the syntactic proofs in Figures 1 and 2. In other words, all these proofs are identified, as demanded in Section 2.(d).

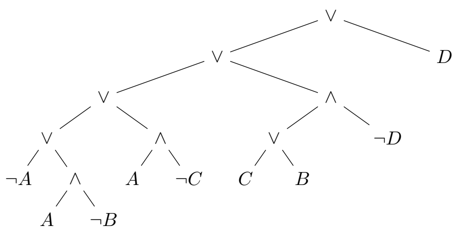
A *combinatorial (pre-)proof* π of a formula F consists of three parts:

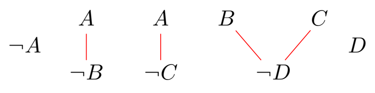
(i) an undirected graph $\mathcal{C} = \langle V, R\rangle$ with vertices V and edges R
(depicted in black dots (V) and red/regular thick edges (R) in Figure 4),

(ii) a perfect matching B on the vertices V of the graph $\mathcal{C}$
(depicted by bold/blue edges in Figure 4),[8] and

(iii) a mapping f from the vertices V of $\mathcal{C}$ to the literal occurrences in F
(depicted by purple arrows in Figure 4).

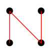
We say that F is the *conclusion* of π. Before we can say when such a π is *correct*, i.e., under which conditions it represents a *proof*, we need the notion of the *formula graph* $\mathcal{G}(F)$ of a formula F. If F is in negation normal form (nnf)[9], we define $\mathcal{G}(F) = \langle V_F, R_F\rangle$ such that V_F is the set of literal occurrences in F, and there is an edge between two literal occurrences iff their first common ancestor in the formula tree is an $\wedge$-connective. To give an example, below on the left is the formula tree of the nnf of the formula in (2.2), and on the right is the formula graph:



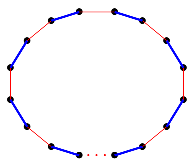


The formula graph of a formula F that is not in nnf is the formula graph of the nnf of F. We can neglect the bracketing of the connectives since two formulas have the same formula graph if and only if they are equivalent modulo associativity and commuatativity of $\wedge$ and $\vee$.[10]

We say that π is *correct*, i.e., it represents a *proof* iff:

(i) The graph $\mathcal{C}$ is a *cograph*, i.e., it does not contain a configuration  as induced subgraph,

(ii) The perfect matching B on the graph $\mathcal{C}$ is such that there is no configuration of the shape:[11]



(iii) the mapping f from $\mathcal{C} = \langle V, R\rangle$ to $\mathcal{G}(F) = \langle V_F, R_F\rangle$ is

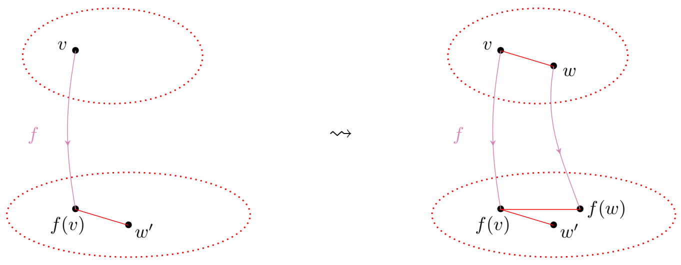
(a) a *graph homomorphism*, i.e., if there is an edge between $u, v \in V$ in R, then there is an edge between $f(u)$ and $f(v)$ in R_F,

(b) *axiom preserving*, i.e., vertices that are connected by a B-edge in $\mathcal{C}$ are mapped to dual literals in $\mathcal{G}(F)$, and

(c) a *skew fibration*, i.e., for all $v \in V$ and $w' \in V_F$, if there is an edge between $f(v)$ and w' in R_F, as depicted on the left below, then there is a $w \in V$ such that there is no edge in V_F between w' and $f(w)$, as depicted on the right below (but we can have

[8]A matching in a graph is a set of pairwise non-adjacent edges. A matching is perfect if it matches all vertices of the graph.
[9]A formula is in nnf if negation occurs only in front of atoms, and the only binary connectives are $\wedge$ (and) and $\vee$ (or).
[10]This can be shown by a simple induction on the number of associativity/commuativity steps.
[11]Formally, the condition is that there is no chordless æ-cycle. See [Ret03] for details.

that $f(w) = w'$):



Condition (i) says that $\mathcal{C}$ is the formula graph of some formula $F_\mathcal{C}$. Condition (ii) ensures that the linear part of the proof is correct. More precisely, the B-edges describe a proof net of linear logic for the formula $F_\mathcal{C}$ (in the sense of Retoré [Ret03]). Condition (iii) ensures that erasure and duplication of atomic formulas follows the rules of classical logic. More precisely, there is a skew fibration from $\mathcal{C}$ to $\mathcal{G}(F)$ iff there is derivation from $F_\mathcal{C}$ to F using only contraction and weakening, i.e., duplication and erasure of subformulas (see [Hug06b,Str07a] for details).

The important observation here is that all three conditions are purely combinatorial properties that do not refer to the syntax of a formal deductive proof system. Furthermore, all these conditions can be checked in polynomial time [Hug06a]. Hughes also has shown soundness and completeness: Every Boolean tautology has a combinatorial proof and every combinatorial proof has a Boolean tautology as conclusion. In [Hug06b,Str17b,Str17a], combinatorial proofs have been extended to also cover cuts, substitution, and extension, and can therefore handle all known notions of proof compression in classical propositional logic. This motivates the following thesis:

$\boxed{\text{Two proofs are \textit{the same} iff they have identical combinatorial proofs.}}$

We consider combinatorial proofs as *canonical proof representations*, and it has already been shown in [Hug06b,Str17b] how syntactic proofs in the sequent calculus can be translated into combinatorial proofs, in such a way that the B-edges in the combinatorial proofs are in one-to-one correspondence to the instances of the axiom rule in the syntactic proof. Similar results have been obtained in [AS18] for analytic tableaux, and in [Str17b,Str17a] for deep inference proofs [GS01,BT01]. This ensures that proof identity via combinatorial proofs respects the size of proofs as demanded by Objective 2.

4. Conclusion

The results mentioned in the previous section suggest that combinatorial proofs satisfy all four objectives that have been postulated in Section 2. One can therefore argue that combinatorial proofs solve the problem of proof identity for the special case of classical propositional logic. However, the problem remains wide open when we go beyond that:

- The most important question is what happens in the presence of quantifiers. Can we have combinatorial proofs for first-order logic? What about higher-order logic? And how should we deal with non-logical axioms?
- Can we also have combinatorial proofs for intuitionistic logic, and if yes, what is the relation to λ-calculus? And what about intermediate logics?
- We can say that proof nets are for linear logic what combinatorial proofs are for classical logic. This raises the question of whether we can have a similar notion of "combinatorial proof" for other substructural logics.
- Another question is whether we can extend the data structure of combinatorial proofs to modalities. And if this is the case, is there a general scheme, or are there modal logics which are more "friendly" towards combinatorial proofs, i.e., is the situation similar to the (cut-free) sequent calculus, which can capture some modal logics very easily (like K and S4) and has notorious difficulties with others (like KB and S5)?

Finally, there are two issues that have deliberately been neglected in this article because a satisfactory treatment would have gone beyond its scope.

The first one comes from our original motivation in computer science: the formal verification of software and hardware. The most successful method so far is *model checking* [CGP99], which essentially consists in the exploration of a mathematical model to check if a certain property is fulfilled. If a given model checker comes back with a positive answer then this is considered to be a proof of the given property (modulo the correct implementation of the model checker in use). However, the output we get is not a formal proof in the sense of traditional proof theory.[12] For proof theory, this is an unsatisfactory situation, in particular, since the structure of the algorithms searching the model to check the validity of the formula in question is, on an abstract level, similar to the structure of the algorithms that are employed in proof search [DM08]. From a practical point of view, it would therefore be desirable to have a notion of proof identity that is able to handle proofs that are obtained via model checking.

The second issue neglected in this article is of a more philosophical nature: can a combinatorial proof, or more precisely, a first-order version of a combinatorial proof, capture the mathematical idea behind a proof, and what does that mean?

Acknowledgements. The author would like to thank the anonymous referees for their helpful comments for improving the readability of this article.

Ethics. No animals were harmed for this research.

Data Accessibility. There is no supporting data for this research.

Authors' Contributions. The author declares that he has written this article all by himself.

Competing Interests. The author declares that he has no competing interests.

References

And76 Peter B. Andrews. Refutations by matings. *IEEE Transactions on Computers*, C-25(8):801–807, 1976.

AS18 Matteo Acclavio and Lutz Straßburger. From syntactic proofs to combinatorial proofs. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900, pages 481–497. Springer, 2018.

BC04 Yves Bertot and Pierre Casteréran. *Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

BCG$^+$14 David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu, and Yuting Wang. Abella: A system for reasoning about relational specifications. *Journal of Formalized Reasoning*, 7(2), 2014.

BCH12 Mathieu Boespflug, Quentin Carbonneaux, and Olivier Hermant. The lambda-Pi-calculus modulo as a universal proof language. In *Proceedings of the Second International Workshop on Proof Exchange for Theorem Proving*, volume 878 of *CEUR Workshop Proceedings*, pages 28–43, Manchester, UK, 2012.

BG09 Paola Bruscoli and Alessio Guglielmi. On the proof complexity of deep inference. *ACM Transactions on Computational Logic*, 10(2):1–34, 2009. Article 14.

BGGP10 Paola Bruscoli, Alessio Guglielmi, Tom Gundersen, and Michel Parigot. A quasipolynomial cut-elimination procedure in deep inference via atomic flows and threshold formulae. In *LPAR-16*, volume 6355 of *LNCS*, pages 136–153. Springer-Verlag, 2010.

BHL$^+$08 Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Ceres: An analysis of Fürstenberg's proof of the infinity of primes. *Theoretical Computer Science*, 403(2):160–175, 2008.

[12] As a matter of fact, one of the original motivations for the development of model checking was to avoid interactive theorem provers [Eme08].

BHRU06 Gianluigi Bellin, Martin Hyland, Edmund Robinson, and Christian Urban. Categorical proof theory of classical propositional calculus. *Theoretical Computer Science*, 364(2):146–165, 2006.

Blu93 Richard Blute. Linear logic, coherence and dinaturality. *Theoretical Computer Science*, 115:3–41, 1993.

Boo84 George Boolos. Don't eliminate cut. *Journal of Philosophical Logic*, 13:373–378, 1984.

BT01 Kai Brünnler and Alwen Fernanto Tiu. A local system for classical logic. In R. Nieuwenhuis and A. Voronkov, editors, *LPAR 2001*, volume 2250 of *LNAI*, pages 347–361. Springer, 2001.

CGP99 Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

CH00 Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *ICFP*, pages 233–243, 2000.

CR79 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979.

CS09 J. Robin B. Cockett and Luigi Santocanale. On the word problem for $\Sigma\Pi$-categories, and the properties of two-way communication. In *Computer Science Logic, CSL'09*, volume 5771 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2009.

Das13 Anupam Das. Rewriting with linear inferences in propositional logic. In Femke van Raamsdonk, editor, *24th International Conference on Rewriting Techniques and Applications (RTA)*, volume 21 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 158–173. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2013.

Ded13 The Dedukti system. http://dedukti.gforge.inria.fr/, 2013.

DFH$^+$93 Gilles Dowek, Amy Felty, Gèrard Huet, Hugo Herbelin, Chet Murthy, Catherine Parent, Christine Paulin-Mohring, and Benjamin Werner. The Coq proof assistant user's guide. Technical report, INRIA, Rocquencourt, France, 1993.

DHK03 Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. *J. of Automated Reasoning*, 31(1):31–72, 2003.

DM08 Olivier Delande and Dale Miller. A neutral approach to proof and refutation in MALL. In F. Pfenning, editor, *23th IEEE Symp. on Logic in Computer Science (LICS 2008)*, pages 498–508. IEEE Computer Society Press, 2008.

Doš03 Kosta Došen. Identity of proofs based on normalization and generality. *The Bulletin of Symbolic Logic*, 9:477–503, 2003.

DP04 Kosta Došen and Zoran Petrić. *Proof-Theoretical Coherence*. KCL Publ., London, 2004.

Eme08 E. Allen Emerson. The beginning of model checking: A personal perspective. In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *Lecture Notes in Computer Science*, pages 27–45. Springer, 2008.

FP04 Carsten Führmann and David Pym. On the geometry of interaction for classical logic (extended abstract). In *19th IEEE Symposium on Logic in Computer Science (LICS 2004)*, pages 211–220, 2004.

FP06 Carsten Führmann and David Pym. Order-enriched categorical models of the classical sequent calculus. *Journal of Pure and Applied Algebra*, 204(1):21–78, 2006.

GAA$^+$13 Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A machine-checked proof of the odd order theorem. In *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, volume 7998 of *Lecture Notes in Computer Science*, pages 163–179. Springer, 2013.

Gen35 Gerhard Gentzen. Untersuchungen über das logische Schließen. I. *Mathematische Zeitschrift*, 39:176–210, 1935.

GG08 Alessio Guglielmi and Tom Gundersen. Normalisation control in deep inference via atomic flows. *Logical Methods in Computer Science*, 4(1:9):1–36, 2008.

GGS10 Alessio Guglielmi, Tom Gundersen, and Lutz Straßburger. Breaking paths in atomic flows for classical logic. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 284–293. IEEE Computer Society, 2010.

Gir87 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

Gon07 Georges Gonthier. The four colour theorem: Engineering of a formal proof. In Deepak Kapur, editor, *8th Asian Symposium on Computer Mathematics*, volume 5081 of *Lecture Notes in Computer Science*, page 333. Springer, 2007.

Gor99 Rajeev Goré. Tableau methods for modal and temporal logics. In Marcello D'Agostino, Dov M. Gabbay, Reiner Hähnle, and Joachim Posegga, editors, *Handbook of Tableau Methods*, pages 297–396. Springer Netherlands, 1999.

GS01 Alessio Guglielmi and Lutz Straßburger. Non-commutativity and MELL in the calculus of structures. In Laurent Fribourg, editor, *Computer Science Logic, CSL 2001*, volume 2142 of *LNCS*, pages 54–68. Springer-Verlag, 2001.

Hal05 Thomas C. Hales. Introduction to the Flyspeck Project. In Thierry Coquand, Henri Lombardi, and Marie-Françoise Roy, editors, *Mathematics, Algorithms, Proofs*, volume 05021 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

HH14 Willem Heijltjes and Robin Houston. No proof nets for MLL with units: proof equivalence in MLL is PSPACE-complete. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 50:1–50:10. ACM, 2014.

Hil David Hilbert. Mathematische Notizbücher. Niedersächsische Staats- und Universitätbibliothek, Cod. Ms. D. Hilbert 600:3, S.25.

Hil00 David Hilbert. Mathematische Probleme. *Nachrichten der Königlichen Gesellschaft der Wissenschaften zu Göttingen, mathematisch-physikalische Klasse*, 3:253–297, 1900.

How80 W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.

HS16 Willem Heijltjes and Lutz Straßburger. Proof nets and semi-star-autonomous categories. *Mathematical Structures in Computer Science*, 26(5):789–828, 2016.

Hug06a Dominic Hughes. Proofs Without Syntax. *Annals of Mathematics*, 164(3):1065–1076, 2006.

Hug06b Dominic Hughes. Towards Hilbert's 24^{th} problem: Combinatorial proof invariants: (preliminary version). *Electr. Notes Theor. Comput. Sci.*, 165:37–63, 2006.

Hyl02 J. M. E. Hyland. Proof theory in the abstract. *Annals of Pure and Applied Logic*, 114(1–3):43–78, 2002.

Hyl04 J. Martin E. Hyland. Abstract interpretation of proofs: Classical propositional calculus. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic, CSL 2004*, volume 3210 of *LNCS*, pages 6–21. Springer-Verlag, 2004.

Jeř09 Emil Jeřábek. Proof complexity of the cut-free calculus of structures. *Journal of Logic and Computation*, 19(2):323–339, 2009.

KEH^{+}09 Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd Symposium on Operating Systems Principles (22nd SOSP'09), Operating Systems Review (OSR)*, pages 207–220, Big Sky, MT, October 2009. ACM SIGOPS.

KP89 Jan Krajíček and Pavel Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *The Journal of Symbolic Logic*, 54(3):1063–1079, 1989.

Lam68 Joachim Lambek. Deductive systems and categories. I: Syntactic calculus and residuated categories. *Math. Systems Theory*, 2:287–318, 1968.

Lam69 Joachim Lambek. Deductive systems and categories. II. standard constructions and closed categories. In P. Hilton, editor, *Category Theory, Homology Theory and Applications*, volume 86 of *Lecture Notes in Mathematics*, pages 76–122. Springer, 1969.

Lam07 François Lamarche. Exploring the gap between linear and classical logic. *Theory and Applications of Categories*, 18(18):473–535, 2007.

Lau03 Olivier Laurent. Polarized proof-nets and $\lambda\mu$-calculus. *Theoretical Computer Science*, 290(1):161–188, 2003.

Ler09 Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, 2009.

LS86 Joachim Lambek and Phil J. Scott. *Introduction to higher order categorical logic*, volume 7 of *Cambridge studies in advanced mathematics*. Cambridge University Press, 1986.

LS05a François Lamarche and Lutz Straßburger. Constructing free Boolean categories. In *LICS'05*, pages 209–218, 2005.

LS05b François Lamarche and Lutz Straßburger. Naming proofs in classical propositional logic. In Paweł Urzyczyn, editor, *TLCA'05*, volume 3461 of *LNCS*, pages 246–261. Springer, 2005.

LS06 François Lamarche and Lutz Straßburger. From proof nets to the free *-autonomous category. *Logical Methods in Computer Science*, 2(4:3):1–44, 2006.

McK05 Richard McKinley. Classical categories and deep inference. In *Structures and Deduction 2005 (Satellite Workshop of ICALP'05)*, 2005.

McK06 Richard McKinley. *Categorical Models of First Order Classical Proofs*. PhD thesis, University of Bath, 2006.

Mil11 Dale Miller. A proposal for broad spectrum proof certificates. In J.-P. Jouannaud and Z. Shao, editors, *CPP: First International Conference on Certified Programs and Proofs*, volume 7086 of *Lecture Notes in Computer Science*, pages 54–69, 2011.

NPW02 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Number 2283 in Lecture Notes in Computer Science. Springer, 2002.

NS15 Novak Novakovic and Lutz Straßburger. On the power of substitution in the calculus of structures. *ACM Trans. Comput. Log.*, 16(3):19, 2015.

Par92 Michel Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *LPAR 1992*, volume 624 of *LNAI*, pages 190–201. Springer-Verlag, 1992.

Pra65 Dag Prawitz. *Natural Deduction, A Proof-Theoretical Study*. Almquist and Wiksell, 1965.

Ret03 Christian Retoré. Handsome proof-nets: perfect matchings and cographs. *Theoretical Computer Science*, 294(3):473–488, 2003.

Sai15 Ronan Saillard. Rewriting modulo β in the $\lambda\Pi$-calculus modulo. In Iliano Cervesato and Kaustuv Chaudhuri, editors, *Proceedings Tenth International Workshop on Logical Frameworks and Meta Languages: Theory and Practice, LFMTP 2015, Berlin, Germany, 1 August 2015.*, volume 185 of *EPTCS*, pages 87–101, 2015.

See89 R.A.G. Seely. Linear logic, *-autonomous categories and cofree coalgebras. *Contemporary Mathematics*, 92, 1989.

Sel01 Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Math. Structures in Comp. Science*, 11:207–260, 2001.

Smu68 Raymond M. Smullyan. *First-Order Logic*. Springer-Verlag, Berlin, 1968.

SR98 Thomas Streicher and Bernhard Reus. Classical logic, continuation semantics and abstract machines. *J. of Functional Programming*, 8(6):543–572, 1998.

Str07a Lutz Straßburger. A characterisation of medial as rewriting rule. In Franz Baader, editor, *Term Rewriting and Applications, RTA'07*, volume 4533 of *LNCS*, pages 344–358. Springer, 2007.

Str07b Lutz Straßburger. On the axiomatisation of Boolean categories with and without medial. *Theory and Applications of Categories*, 18(18):536–601, 2007.

Str11 Lutz Straßburger. *Towards a Theory of Proofs of Classical Logic*. Habilitation à diriger des recherches, Université Paris VII, 2011.

Str12 Lutz Straßburger. Extension without cut. *Annals of Pure and Applied Logic*, 163(12):1995–2007, 2012.

Str17a Lutz Straßburger. Combinatorial Flows and Proof Compression. Research Report RR-9048, Inria Saclay, 2017.

Str17b Lutz Straßburger. Combinatorial flows and their normalisation. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*, volume 84 of *LIPIcs*, pages 31:1–31:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

Thi97 Hayo Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, 1997.

Thi03 Rüdiger Thiele. Hilbert's twenty-fourth problem. *American Mathematical Monthly*, 110:1–24, 2003.

TS00 Anne Sjerp Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, second edition, 2000.