Normalization Without Syntax

Willem B. Heijltjes
Department of Computer Science, University of Bath, UK

Dominic J. D. Hughes
Logic Group, University of California Berkeley, CA, USA

Lutz Straßburger
Equipe Partout, Inria Saclay, Palaiseau, France

Abstract

We present normalization for intuitionistic combinatorial proofs (ICPs) and relate it to the simply-typed lambda-calculus. We prove confluence and strong normalization. Combinatorial proofs, or "proofs without syntax", form a graphical semantics of proof in various logics that is canonical yet complexity-aware: they are a polynomial-sized representation of sequent proofs that factors out exactly the non-duplicating permutations. Our approach to normalization aligns with these characteristics: it is canonical (free of permutations) and generic (readily applied to other logics). Our reduction mechanism is a canonical representation of reduction in sequent calculus with closed cuts (no abstraction is allowed below a cut), and relates to closed reduction in lambda-calculus and supercombinators. While we will use ICPs concretely, the notion of reduction is completely abstract, and can be specialized to give a reduction mechanism for any representation of typed normal forms.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Lambda calculus

Keywords and phrases combinatorial proofs, intuitionistic logic, lambda-calculus, Curry–Howard, proof nets

Digital Object Identifier 10.4230/LIPIcs.FSCD.2022.19

Related Version *Technical report*: `https://hal.inria.fr/hal-03654060` [17]

Funding Supported by EPSRC Grant EP/R029121/1 *Typed Lambda-Calculi with Sharing and Unsharing* and Inria Associated Team *Combinatorial Proof Normalization (COMPRONOM)*.

Acknowledgements Dominic Hughes would like to thank Wes Holliday, his host at U.C. Berkeley. We thank the referees for their diligence.

1 Introduction

The sequent calculus was introduced by Gentzen [9] as a meta-calculus, to describe the construction of proofs in natural deduction, the object-calculus. The sequent calculus has good proof-theoretic properties, such as isolating the cut-rule as the distinction between normal and non-normal proofs and avoiding the ad-hoc construction of open and closed assumptions. However, it features many permutations, that relate different ways of constructing the same natural deduction proof. This is a problem for proof normalization in particular, since permutations come to dominate the cut-elimination process.

When Girard introduced Linear Logic [10], it was naturally expressed in sequent calculus, which defined clear and natural meta-level operations for proof construction. But there was no object-level calculus to which these applied, and which might capture its computational content. Constructing one became the project of *proof nets* [10, 12, 20, 15], with the aim of *canonicity*: proof nets aim to represent sequent proofs canonically, modulo permutations.

Combinatorial proofs, first developed for classical propositional logic by Hughes [18], continue the tradition of proof nets with a refined aim, called *local canonicity* [19]. The issue is that permutations may *duplicate* subproofs; to factor them out then generally causes an exponential blowup of the representation. Figure 1 illustrates such a permutation. The idea of *local canonicity* is to give a complexity-sensitive, polynomial representation of sequent proofs, modulo the non-duplicating permutations. This is achieved in combinatorial proofs by a clean separation of the logical content (the logical rules of a sequent proof) and the structural content (the structural rules, contraction and weakening), each captured in a distinct part of a combinatorial proof. Sequent calculi are generally unable to stratify proofs in this way, but it is a natural form of decomposition in deep inference [30]. Beyond classical propositional logic, combinatorial proofs have been given for intuitionistic propositional logic [16], first-order classical logic [21, 22], relevance logics [2], and modal logics [3].

The problem of exponential duplication appears also at the level of formula isomorphisms [6, 8], and is usefully illustrated there. The formula-isomorphisms of symmetry, associativity, and currying, below, do not affect the size of the formula.

$$A \wedge B \ \sim\ B \wedge A \qquad A \wedge (B \wedge C) \ \sim\ (A \wedge B) \wedge C \qquad (A \wedge B) \Rightarrow C \ \sim\ A \Rightarrow (B \Rightarrow C)$$

But the distributivity isomorphism, below, duplicates the antecedent of an implication, and its repeated application may cause exponential growth. Combinatorial proofs, as a complexity-aware graphical formalism, factor out the former three, but not the latter.

$$A \Rightarrow (B \wedge C) \ \sim\ (A \Rightarrow B) \wedge (A \Rightarrow C)$$

We are interested in the question: what is a natural and general notion of composition for combinatorial proofs? In this paper we consider the intuitionistic case – *Intuitionistic Combinatorial Proofs* (ICPs) [16] – where the question is particularly pertinent due to the Curry–Howard correspondence with typed lambda-calculi.

Our aim has been twofold: 1) to implement sequent-calculus reduction canonically (i.e. without permutations), and 2) to ensure our notion of reduction is sufficiently abstract that it will (plausibly) generalize to combinatorial proofs more widely.

Our solution is a notion of composition in conjunction-implication intuitionistic logic that is locally canonical for sequent calculus normalization, in the sense that non-duplicating permutations on cuts are factored out. Reduction operates on trees of normal forms, where edges represent cuts, giving a simple and natural structure that may easily generalize to other logics. A reduction step on a given edge is determined by how the attached nodes may sequentialize, not by their internal structure. Consequently, the reduction mechanism is *abstract* in the sense that it is agnostic about the actual contents of nodes, which can be any representation of normal forms. Beyond the scope of this paper, the mechanism generalizes straightforwardly to classical logic, which we will briefly expand on in the conclusion.

Proofs are omitted; a version with all proofs in an appendix is on the HAL archive [17].

1.1 Composition

Composition of proofs in intuitionistic sequent calculus is by the following cut-rule, followed by cut-elimination. We would like to transport this operation to combinatorial proofs.

$$\frac{\Gamma \vdash A \qquad A, \Delta \vdash B}{\Gamma, \Delta \vdash B}\,\mathsf{cut}$$

We identify two prominent approaches for similar composition operations in the literature (our classification is not intended to be comprehensive, only helpful in setting out similarities):

$$\dfrac{\Gamma \vdash A \qquad \dfrac{B, B, \Delta \vdash C}{B, \Delta \vdash C}\,\mathsf{c}}{\Gamma, A \Rightarrow B, \Delta \vdash C}\,{\Rightarrow}\mathsf{L} \quad \approx \quad \dfrac{\dfrac{\Gamma \vdash A \qquad \dfrac{\Gamma \vdash A \quad B, B, \Delta \vdash C}{B, \Gamma, A \Rightarrow B, \Delta \vdash C}\,{\Rightarrow}\mathsf{L}}{\Gamma, A \Rightarrow B, \Gamma, A \Rightarrow B, \Delta \vdash C}\,{\Rightarrow}\mathsf{L}}{\Gamma, A \Rightarrow B, \Delta \vdash C}\,\mathsf{c}$$

Figure 1 A duplicating permutation. Intuitionistic sequent calculus, as we will use it, has exactly one duplicating permutation, illustrated here. Permuting the contraction rule c and the implication-left rule $\Rightarrow\mathsf{L}$ duplicates the subproof on the left. Iterating the permutation gives exponential growth. It is instructive to consider the translation to natural deduction, which unfolds along this permutation and does indeed grow exponentially.

Internal rewriting. An object-calculus may support non-normal forms and rewriting internally. In the λ-calculus, composition creates a redex, which is then beta-reduced. Likewise, many notions of proof net admit an explicit notion of cut, as a node or as a *cut-link* connecting dual formulae, that is eliminated by rewriting [12, 19], giving rise to the interaction nets paradigm [27].

Direct composition. For an object calculus that admits only normal forms, composition may be computed by a single-shot operation. Examples are the Geometry of Interaction, which computes a normal form via the execution formula [11]; game semantics, which composes strategies by *interaction + hiding* [1, 25]; evaluation of cut-nets in ludics [13]; and various notions of proof net where composition is a form of path composition over links [20, 15, 23].

Observe that object-level proofs become an invariant for sequent-calculus cut-elimination. Based on prior art, one may readily imagine what either approach would involve for ICPs. For internal rewriting, an ICP may be constructed over a sequent that includes internal cut-formulas as special antecedents $A \Rightarrow A$ (marked below by underlining), introduced by a cut as analogous to a $\Rightarrow\mathsf{L}$ rule, and eliminated by rewriting. One may transport sequent-calculus cut-elimination to this setting by identifying sub-proofs of ICPs, via *kingdoms* [4].

$$\Gamma, \underline{A_1 \Rightarrow A_1}, \ldots, \underline{A_n \Rightarrow A_n} \vdash B$$

For direct composition, ICPs may be interpreted as games with *sharing* [16], for which the *interaction + hiding* approach can be explored. Both these approaches are interesting and deserve to be investigated, and we may do so in future. However, they will inevitably require some intricate combinatorics, and are not likely to generalize across combinatorial proofs.

Here, we describe a normalization method for ICPs that is simple, natural, and achieves both our main objectives: 1) it is effectively a permutation-free implementation of sequent calculus cut-elimination, and 2) it is sufficiently abstract that it is likely to generalize well. Technically, ICPs will form the nodes of a *combinatorial tree*, connected by edges that represent cuts. Combinatorial trees are then reduced by cut-elimination, following the reduction in sequent calculus. Interestingly, this approach fits neither of the above categories well, and instead suggests to identify a third category:

External rewriting. An object calculus without internal composition may be extended by a secondary structure, which is then evaluated by rewriting. The prime example is *supercombinators* [24, 29], where normalization takes place on a tree of normal-form λ-terms (restricted to having no abstractions inside applications).

We explore the parallels between our combinatorial trees and supercombinators in Section 7. In addition, we connect ICP normalization to *closed reduction* in λ-calculus [7] in Section 8, via a novel explicit-substitution calculus, the *combinatory λ-calculus*, in Section 6.

2 Intuitionistic Combinatorial Proofs

We give a concise inductive definition of ICPs; see [16] for a full treatment including an informal introduction and a geometric definition. For the purposes of this paper, it would also be sufficient to view ICPs as sequent proofs modulo permutations.

We work in conjunction–implication intuitionistic logic. ***Formulas*** A, B, C are given by the grammar below, where P, Q are propositional atoms. A ***context*** Γ, Δ is a multiset of formulas and a ***sequent*** $\Gamma \vdash A$ is a context with a formula.

$$A, B, C ::= P \mid A \wedge B \mid A \Rightarrow B$$

An ICP for a formula A will be a graph homomorphism $f : \mathcal{G} \to [\![A]\!]$ consisting of:

- an ***arena*** $[\![A]\!]$, a graph representing the formula A modulo the non-duplicating isomorphisms of symmetry, associativity, and currying;
- a ***linked arena*** $\mathcal{G}$, a proof net in IMLL (intuitionistic multiplicative linear logic) over an arena rather than a formula, to represent the *logical* rules of the sequent calculus;
- a ***skew fibration*** f, a graph homomorphism from $\mathcal{G}$ to $[\![A]\!]$ representing the *structural* rules of contraction and weakening.

We define each component inductively. An arena will be a DAG (directed acyclic graph) $\mathcal{G} = (V_\mathcal{G}, \to_\mathcal{G})$ with vertices $V_\mathcal{G}$ and edges $\to_\mathcal{G} \subseteq V_\mathcal{G} \times V_\mathcal{G}$. We indicate the ***root vertices*** of $\mathcal{G}$ (those without outgoing edges) by $R_\mathcal{G}$. Consider the following two operations: a ***sum*** of two graphs $\mathcal{G} + \mathcal{H}$ is their disjoint union, and a ***subjunction*** $\mathcal{G} \triangleright \mathcal{H}$ is a disjoint union that in addition connects all the roots of $\mathcal{G}$ to the roots of $\mathcal{H}$.

$$\begin{array}{llcl} \text{sum:} & \mathcal{G} + \mathcal{H} & = & (V_\mathcal{G} \uplus V_\mathcal{H}, \to_\mathcal{G} \uplus \to_\mathcal{H}) \\ \text{subjunction:} & \mathcal{G} \triangleright \mathcal{H} & = & (V_\mathcal{G} \uplus V_\mathcal{H}, \to_\mathcal{G} \uplus \to_\mathcal{H} \uplus (R_\mathcal{G} \times R_\mathcal{H})) \end{array}$$

▶ **Definition 1.** *An* ***arena*** *is a graph $\mathcal{G}$ constructed from single vertices by $(+)$ and $(\triangleright)$, with an L-**labelling** $\ell_\mathcal{G} : V_\mathcal{G} \to L$ assigning each vertex a label from a set L. The arena $[\![A]\!]$ of a formula A is given inductively as follows: $[\![P]\!]$ is a single vertex labelled P, and*

$$[\![A \wedge B]\!] = [\![A]\!] + [\![B]\!] \qquad \textit{and} \qquad [\![A \Rightarrow B]\!] = [\![A]\!] \triangleright [\![B]\!] .$$

Note that arenas are linear in the size of formulas, and while they factor out symmetry, associativity, and currying, they do not factor out distributivity.

$$[\![A \Rightarrow (B \wedge C)]\!] \neq [\![(A \Rightarrow B) \wedge (A \Rightarrow C)]\!]$$

An ICP will be an *arena morphism*: a map $f : \mathcal{G} \to [\![A]\!]$ given by an underlying function on vertices $f : V_\mathcal{G} \to V_{[\![A]\!]}$ that preserves edges, roots, and the equivalence given by labelling, i.e. if $\ell_\mathcal{G}(v) = \ell_\mathcal{G}(w)$ then $\ell_{[\![A]\!]}(f(v)) = \ell_{[\![A]\!]}(f(w))$. We will construct arena morphisms inductively, which guarantees these conditions. For $g : \mathcal{G} \to [\![A]\!]$ and $h : \mathcal{H} \to [\![B]\!]$ we have the operations

$$\begin{array}{lrcll} \text{implication:} & g \triangleright h & : & \mathcal{G} \triangleright \mathcal{H} \to [\![A]\!] \triangleright [\![B]\!] & \\ \text{sum:} & g + h & : & \mathcal{G} + \mathcal{H} \to [\![A]\!] + [\![B]\!] & \\ \text{contraction:} & [g, h] & : & \mathcal{G} + \mathcal{H} \to [\![A]\!] & \qquad (\text{where } [\![A]\!] = [\![B]\!]) \end{array}$$

where each case is given by the union of the underlying functions on vertex sets: for implication and sum, $g \cup h : (V_\mathcal{G} \uplus V_\mathcal{H}) \to (V_{[\![A]\!]} \uplus V_{[\![B]\!]})$, and for contraction $g \cup h : (V_\mathcal{G} \uplus V_\mathcal{H}) \to V_{[\![A]\!]}$. In addition, we use the following constructions, where $\varnothing$ is the empty graph.

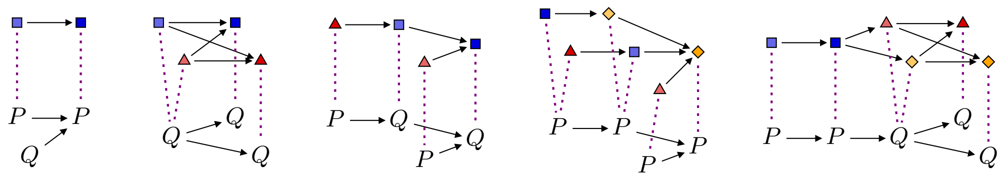
$$\begin{array}{lrcl} \text{axiom:} & 1_{P,Q} & : & [\![P]\!] \to [\![Q]\!] \\ \text{weakening:} & \varnothing_A & : & \varnothing \to [\![A]\!] \end{array}$$

$$\frac{}{1 :: P \vdash 1 :: P}\,\mathsf{ax}^* \qquad \frac{\varphi :: \Gamma \vdash f :: B}{\varphi :: \Gamma, \varnothing :: A \vdash f :: B}\,\mathsf{w} \qquad \frac{\varphi :: \Gamma, k :: A, l :: A \vdash f :: B}{\varphi :: \Gamma, [k,l] :: A \vdash f :: B}\,\mathsf{c}^\dagger$$

$$\frac{\varphi :: \Gamma, k :: A, l :: B \vdash f :: C}{\varphi :: \Gamma, k{+}l :: A \wedge B \vdash f :: C}\,\wedge\mathsf{L} \qquad \frac{\varphi :: \Gamma \vdash f :: A \qquad \psi :: \Delta \vdash g :: B}{\varphi :: \Gamma, \psi :: \Delta \vdash f{+}g :: A \wedge B}\,\wedge\mathsf{R}$$

$$\frac{\varphi :: \Gamma, k :: A \vdash f :: B}{\varphi :: \Gamma \vdash k \triangleright f :: A \Rightarrow B}\,\Rightarrow\mathsf{R} \qquad \frac{\varphi :: \Gamma \vdash f :: A \qquad k :: B, \psi :: \Delta \vdash g :: C}{\varphi :: \Gamma, f \triangleright k :: A \Rightarrow B, \psi :: \Delta \vdash g :: C}\,\Rightarrow\mathsf{L}^\ddagger$$

Figure 2 Inductive construction of ICPs. ($*$) Each instance of ax is given a distinct label in the source arena. (†) For c we require $k, l \neq \varnothing$. (‡) For $\Rightarrow\mathsf{L}$ we require $k \neq \varnothing$.



$P \Rightarrow Q \Rightarrow P$ $\quad$ $Q \Rightarrow (Q \wedge Q)$ $\quad$ $(P \Rightarrow Q) \Rightarrow P \Rightarrow Q$ $\quad$ $(P \Rightarrow P) \Rightarrow P \Rightarrow P$ $\quad$ $((P \Rightarrow P) \Rightarrow Q) \Rightarrow (Q \wedge Q)$

$\lambda x.\lambda y.x$ $\quad$ $\lambda x.\langle x, x\rangle$ $\quad$ $\lambda f.\lambda x.fx$ $\quad$ $\lambda f.\lambda x.f(fx)$ $\quad$ $\lambda f.\langle f(\lambda x.x), f(\lambda x.x)\rangle$

Figure 3 Examples of ICPs with corresponding λ-terms. The source arena is at the top, with its labelling given by coloured shapes. The target arena is at the bottom, labelled with propositional atoms, and the arena morphism is given by dotted (purple) lines.

The axiom is the trivial map from one singleton arena (with vertex labelled P) to another (with vertex labelled Q). Weakening is the empty morphism. Note that because arenas are non-empty, weakening in isolation is not an arena morphism, but we will use it only in the context of an implication, sum, or contraction, so that this is not an issue.

We write $f :: A$ for $f : \mathcal{G} \to [\![A]\!]$. To construct ICPs from sequent proofs we use ***sequents*** of arena morphisms (and weakenings), that represent a single arena morphism as follows.

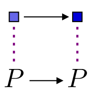
$$k_1 :: A_1, \ldots, k_n :: A_n \vdash f :: B \quad \Longleftrightarrow \quad (k_1 + \ldots + k_n) \triangleright f \ :: \ (A_1 \wedge \ldots \wedge A_n) \Rightarrow B$$

We refer to f and the k_i as ***ports***, where k_i is an ***antecedent*** and f the ***consequent***, and we write $\varphi :: \Gamma$ for the ***context*** $k_1 :: A_1, \ldots, k_n :: A_n$.

▶ **Definition 2.** *An **intuitionistic combinatorial proof (ICP)** of a formula A is an arena morphism $f :: A$ constructed by the sequent calculus of Figure 2.*

Figure 3 gives examples of ICPs, with corresponding types and λ-terms (the translation will be made formal in Section 8). Figure 4 gives non-examples of ICPs.

For clarity, an axiom ax generates the ICP below.

$1 :: P \vdash 1 :: P \quad =$ 

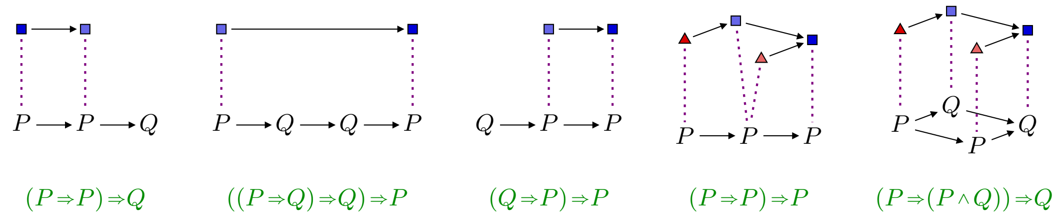


Figure 4 Non-examples of ICPs. They cannot be constructed with the sequent calculus in Figure 2.

a) $$\frac{\Gamma\vdash A \qquad A,\Delta\vdash B}{\Gamma,\Delta\vdash B}\,\mathsf{cut}$$

b) $$\varphi::\Gamma\vdash f::A \;=\; \frac{\varphi::\Gamma}{f::A}$$

$$k::A,\psi::\Delta\vdash g::B \;=\; \frac{k::A\quad \psi::\Delta}{g::B}$$

c) $$\begin{array}{c}\dfrac{\varphi::\Gamma}{f::A}\\ \hdashline \dfrac{k::A\quad\psi::\Delta}{g::B}\end{array}$$

Figure 5 Composition of combinatorial proofs into combinatorial trees. **a)** The sequent calculus cut-rule. **b)** Presenting ICP sequents as nodes of a tree, with antecedent ports above and consequent port below a central line. **c)** Connecting both nodes by an edge, represented by a dashed line, to form a tree.

We call the subgraph $\square\rightarrow\blacksquare$ a ***link***, where the side condition ($\star$) in Figure 2 requires that every link receives a different label $\blacksquare$, $\blacktriangle$, $\lozenge$, etc. Vertices are ***equivalent*** if they have the same label, and ICPs as arena morphisms preserve equivalence by construction.

To *decompose* an ICP, the unary rules $\wedge\mathsf{L},\Rightarrow\!\mathsf{R},\mathsf{c},\mathsf{w}$ apply whenever the given port is of the right kind, respectively $k+l$, $k\triangleright f$, $[k,l]$, and $\varnothing$. The binary rules $\wedge\mathsf{R},\Rightarrow\!\mathsf{L}$ apply only when the ICP can be split into two without breaking up any links in the source graph. We write $\varphi\mid\mid\psi$ when the sources of φ and ψ do not share any labels; then the rules $\wedge\mathsf{R},\Rightarrow\!\mathsf{L}$ as given in Figure 2 apply in reverse exactly when respectively $\varphi,f\mid\mid\psi,g$ and $\varphi,f\mid\mid k,\psi,g$. We call a port ***open*** if the ICP can be decomposed along it, and ***closed*** otherwise.

We refer to [16] for a *geometric* definition of ICPs, where the equivalence with the *inductive* definition given here is a theorem. We recall the following from [16].

▶ **Theorem 3** (Local canonicity). *Two sequent proofs construct the same ICP if and only if they are equivalent modulo non-duplicating rule permutations and formula-isomorphisms.*

3 Composition of combinatorial proofs

Combinatorial proofs represent normal forms: the sequent calculus for constructing them, in Figure 2, does not have a cut-rule (Figure 5a). What is expected is a notion of composition, of an ICP for $\Gamma\vdash A$ and one for $A,\Delta\vdash B$ into one for $\Gamma,\Delta\vdash B$.

We give a direct interpretation of composition by taking ICPs as the nodes of a tree, connected by cuts as edges; see Figure 5, where solid lines represent the nodes in the tree and the dashed lines the edges. We formalize this construction as a notion of ***combinatorial tree***, which we will then proceed to reduce. The nature of reduction will make it desirable to have constants available.

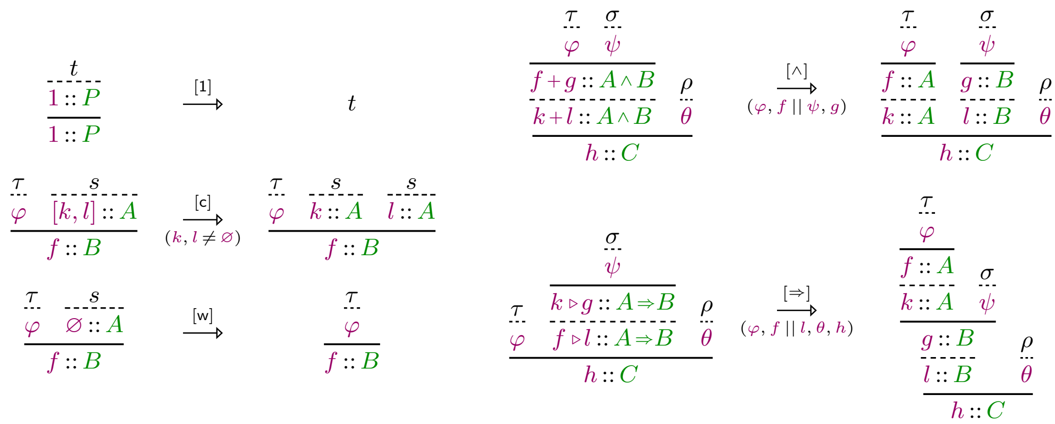


Figure 6 Reduction rules.

▶ **Definition 4** (Combinatorial tree). *A* ***combinatorial tree*** $t :: C$ *with* ***conclusion*** *formula* C *is an inductive tree consisting of either:*

- *a* ***premiss*** $\star :: C$*, representing (the arena of)* C*, or*
- *a* ***constant*** $c :: C$ *where* $C = P_1 \Rightarrow \ldots \Rightarrow P_n \Rightarrow P$ $(n \geq 0)$*, or*
- *a* ***node*** $k_1 :: A_1, \ldots, k_n :: A_n \vdash f :: C$ *with a sequence of* ***subtrees*** $t_1 :: A_1 \ldots t_n :: A_n$*,*

$$\text{written:}\quad \frac{\dfrac{t_1 :: A_1}{k_1 :: A_1} \;\cdots\; \dfrac{t_n :: A_n}{k_n :: A_n}}{f :: C}$$

For a concrete example, Figure 7 gives a reduction featuring various combinatorial trees. We abbreviate $t :: C$ to t, and write $\tau :: \Gamma$ for a ***forest*** $t_1 :: A_1 \ldots t_n :: A_n$ (where $\Gamma = A_1, \ldots, A_n$). Edges connecting τ to antecedents $\varphi = k_1, \ldots, k_n$ are drawn like a single dashed edge, rendering the above tree as (a) below. We indicate a forest of premisses by $\star :: \Gamma$, as in (b), and denote the premisses of a tree t by $\star t$. A tree ***for*** the sequent $\Gamma \vdash A$ is one $t :: A$ with $\star t = \Gamma$. We visually identify the premisses of a tree by a double dashed edge, as in (c) below for s with $\star s = A, \Delta$. Then (d) is the result of replacing $\star :: A$ in s by a tree t for $\Gamma \vdash A$, imitating the cut rule of Figure 5a.

$$(a)\;\frac{\dfrac{\tau :: \Gamma}{\varphi :: \Gamma}}{f :: C} \qquad (b)\;\frac{\dfrac{\star :: \Gamma}{\varphi :: \Gamma}}{f :: C} \qquad (c)\;\frac{\star :: A \quad \star :: \Delta}{s :: B} \qquad (d)\;\frac{\dfrac{\star :: \Gamma}{t :: A} \quad \star :: \Delta}{s :: B}$$

▶ **Definition 5** (Reduction). *Reduction of combinatorial trees is by the rules in Figure 6.*

The reduction rules are essentially those of the sequent calculus, but in a setting that is free of permutations. Observe that while combinatorial trees involve a good amount of notation, the notion of a tree of normal forms is in fact highly conceptual. For reduction, the particular use of ICPs is secondary, and any representation of normal forms would do: the reduction rules are determined entirely by the *sequentialization* or *decomposition* of nodes.

We will assume that constants represent primitives of base type, such as integers and booleans, and functions over base types, such as addition. We extend the reduction rule $[\Rightarrow]$ to the latter case as below; an example instance would be where c is the integer 7 and c' is a squaring function, with the resulting constant c'' the integer 49.

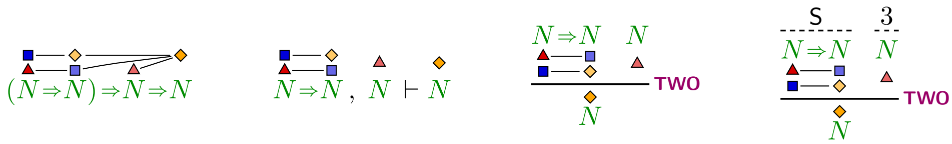
$$\frac{\dfrac{c}{1 :: P} \quad \dfrac{c'}{1 \triangleright k :: P \Rightarrow A} \quad \dfrac{\tau}{\varphi}}{f :: B} \quad \xrightarrow[(1,1\ ||\ k,\varphi,f)]{[\Rightarrow]} \quad \frac{\dfrac{c''}{k :: A} \quad \dfrac{\tau}{\varphi}}{f :: B}$$

3.1 Reduction examples

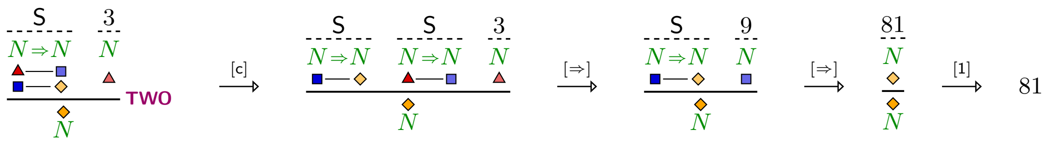
We illustrate reduction with an example analogous to the following lambda calculus reduction, applying the Church numeral two $\lambda f.\lambda x.f(fx) : (N \Rightarrow N) \Rightarrow N \Rightarrow N$ to the squaring function constant $\mathsf{S} : N \Rightarrow N$ and the integer constant $3 : N$.

$$(\lambda f.\lambda x.f(fx))\,\mathsf{S}\,3 \;\rightarrow\; (\lambda x.\mathsf{S}(\mathsf{S}x))\,3 \;\rightarrow\; \mathsf{S}(\mathsf{S}3) \;\rightarrow\; \mathsf{S}9 \;\rightarrow\; 81$$

The combinatorial proof **TWO** corresponding to the Church numeral is the penultimate one displayed in Figure 3. Below, from left to right, we have: numeral two in compact form; two in sequent form; two as a node in a combinatorial tree; and the combinatorial tree representing $(\lambda f.\lambda x.f(fx))\,\mathsf{S}\,3$.



The reduction sequence is as follows:



For a richer example we consider the ICP version of the Church successor $\lambda n.\lambda f.\lambda x.f(nfx)$ applied to Church zero $\lambda f.\lambda x.x$, the squaring function $\mathsf{S} : N \Rightarrow N$ and 4, to yield 16.

$$(\lambda n.\lambda f.\lambda x.f(nfx))\;\;(\lambda f.\lambda x.x)\;\mathsf{S}\;4 \qquad \twoheadrightarrow \qquad 16$$

The ICP reduction is shown in Figure 7.

4 Strong Reduction

The reduction rules $[\wedge], [\Rightarrow]$ apply only when the two ports involved are both open (this is what the side-conditions on the reduction rules entail). We briefly show that this does not lead to a deadlock. In a combinatorial tree, a port is ***extremal*** if it is connected to a premiss or the consequent of the root node, otherwise ***internal***.

▶ **Lemma 6** (Progress). *For a combinatorial tree t with at least one edge, if no extremal port is open, then a reduction step applies.*

The progress lemma illustrates a limitation of the normalization process: reduction may become deadlocked if an extremal port remains open. This is closely related to *weak* reduction in the λ-calculus, which does not reduce under an abstraction, though note it is not the same: internal reduction in a combinatorial tree is allowed, and may still be possible, when the root node is an abstraction. As with weak reduction, this is no limitation in practice: we expect a real program to be of base type, and without free variables (the premisses of a combinatorial tree). In that case the progress lemma guarantees we will not reach a deadlock. This explains also the reason to include constants: without them it is impossible to create a combinatorial tree of base type with no premisses, as it would be logically unsound.

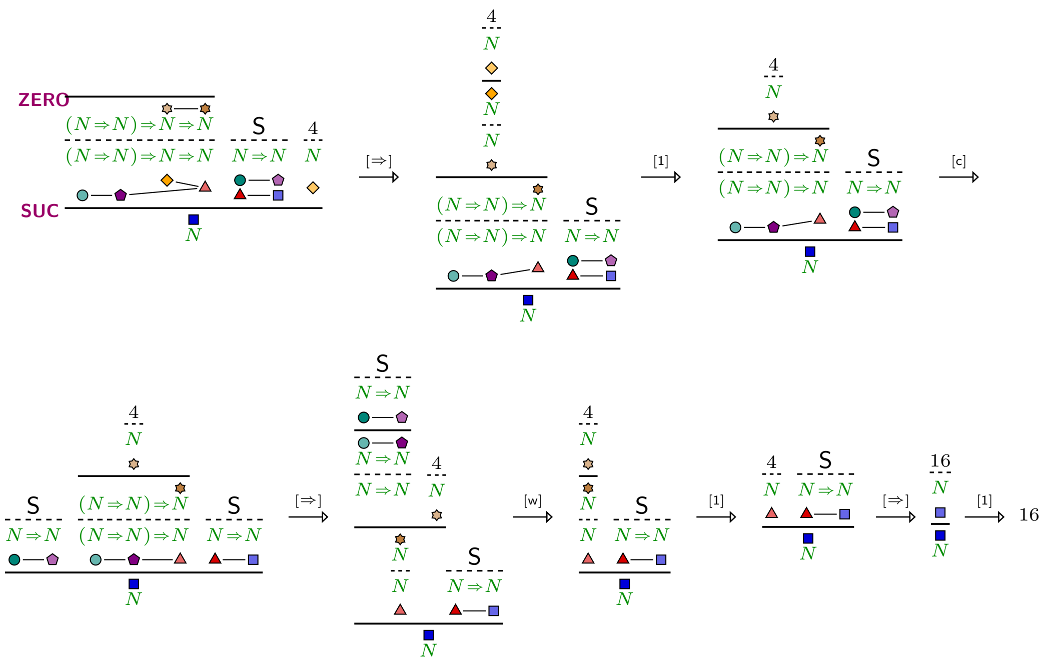


Figure 7 Example of ICP normalization corresponding to the lambda calculus normalization of the Church successor function applied to Church zero, the squaring function constant S, and the constant 4: $(\lambda n.\lambda f.\lambda x.f(nfx))\ (\lambda f.\lambda x.x)\ \mathsf{S}\ 4\ \to^* 16$.

To reduce any combinatorial tree, we combine reduction with sequentialization. We may then reduce open extremal ports by interpreting them as sequent rules. We add a special axiom (icp), given below, to the cut-free sequent calculus. It incorporates a combinatorial tree t for $\Gamma \vdash A$ as a sub-proof of $\Gamma \vdash A$. A proof in this calculus is a ***hybrid proof***.

$$\frac{\boxed{t :: A}}{\star t \vdash A}\ (\mathsf{icp})$$

The reduction rules [1], [∧], and [⇒] apply directly to hybrid proofs, since they preserve the premisses and conclusion of a combinatorial tree. The rules [c] and [w] duplicate or delete premisses; to accommodate this in hybrid proofs, contraction or weakening rules are added. The resulting rules are the last two in Figure 8, which gives the rules for strong reduction.

▶ **Definition 7** (Hybrid reduction). ***Hybrid proof reduction*** *is the rewrite relation on hybrid proofs generated by the rules* [1], [∧], [⇒] *in Figure 6 plus the rules in Figure 8.*

Progress (Lemma 6) gives the following.

▶ **Lemma 8** (Hybrid progress). *If a hybrid proof contains an* (icp) *axiom, a hybrid reduction step applies.*

A normal form of a hybrid proof is then a regular, cut-free sequent proof. This may directly be used to construct an ICP, to obtain fully general ICP normalization. The effect of embedding a combinatorial tree in a hybrid proof is akin to *normalization-by-evaluation* [5]: it provides an environment that supplies sufficient arguments to any function (it is an *applicative context*), and other similar services, to ensure continued reduction.

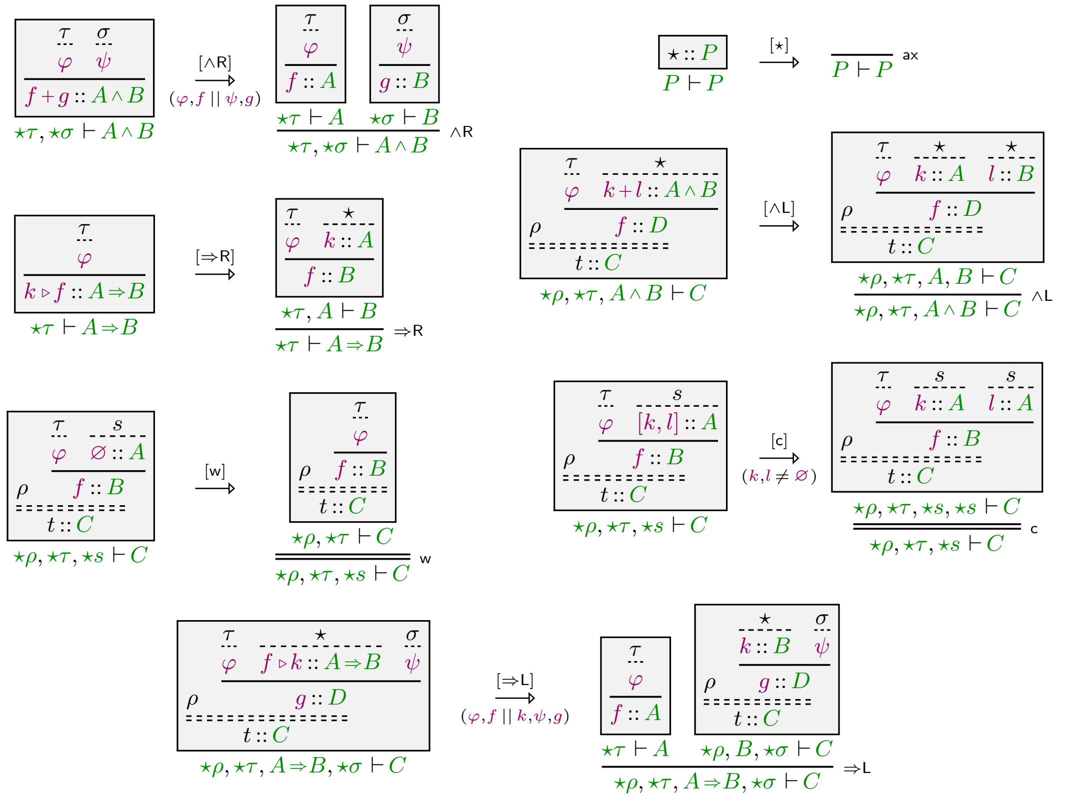


Figure 8 Hybrid sequentialization and reduction rules.

5 Confluence and strong normalization

Combinatorial-tree reduction is confluent and strongly normalizing. In this section we will consider only *local confluence*, which demonstrates the intricacies arising from the local canonicity property of ICPs. Confluence then follows from strong normalization by Newman's Lemma.

The reduction rules for ICPs interact in several intricate ways. Not only can a single node have multiple redexes along different edges, even a single edge may reduce in more than one way. This is due to the multiple ways an arena morphism can be composed inductively, which factor out the formula isomorphisms of associativity, symmetry, and currying, as well as the interaction of conjunction with contraction. Concretely, we have the following equations:

$$
\begin{aligned}
f + g &= g + f & \varnothing + \varnothing &= \varnothing \\
f + (g + h) &= (f + g) + h & [k, \varnothing] &= k \\
(k + l) \triangleright f &= k \triangleright (l \triangleright f) & [k_1, k_2] + [l_1, l_2] &= [k_1 + l_1, k_2 + l_2]
\end{aligned}
$$

We recognize two kinds of critical pairs:

Single-edge when multiple reduction steps apply to a single cut-edge, due to the above equations;

Single-node when multiple reduction steps on distinct edges split the same node.

We do not consider non-splitting reductions on different edges of the same node as critical pairs, since the reductions are independent and converge immediately.

Figure 9 shows how the critical pairs converge. In the following, we will explain the notation used, and consider the precise equations that give rise to the single-edge diagrams.

We use $\twoheadrightarrow$ for the reflexive-transitive closure of $\rightharpoonup$, and dashed arrows are implied by the diagram. Note that the last four diagrams use a different colour scheme to help identify arena morphisms and subtrees across reduction steps.

The first five diagrams cover the single-edge critical pairs, and the last three the single-node critical pairs. The latter, $[\Rightarrow]/[\Rightarrow]^2$, $[\wedge]/[\Rightarrow]$, and $[\Rightarrow]/[\Rightarrow]^3$, are similar to critical pairs found in λ-calculi and proof nets, and converge accordingly.

The single-edge critical pairs are new and delicate. We introduce the notation $t + s$ to mean the following.

$$t+s = \frac{\overset{\tau}{\varphi}\;\;\overset{\sigma}{\psi}}{f+g} \qquad \text{where} \qquad t = \frac{\overset{\tau}{\varphi}}{f} \qquad s = \frac{\overset{\sigma}{\psi}}{g}$$

In the first four diagrams in Figure 9, we depict only the ports and subtrees involved, but omit the node they are attached to. The five single-edge confluence diagrams are due to the following equations:

$$\begin{array}{llcl@{\qquad}llcl}
[\wedge]/[\mathsf{w}]: & \varnothing+\varnothing & = & \varnothing & [\wedge]/[\mathsf{c}]^1: & [k_1,k_2]+[l_1,l_2] & = & [k_1+l_1,k_2+l_2] \\
[\wedge]/[\wedge]: & k+(l+m) & = & (k+l)+m & [\wedge]/[\mathsf{c}]^2: & [k_1,k_2]+l & = & [k_1+l,k_2+\varnothing] \\
[\Rightarrow]/[\Rightarrow]^1: & (k+l)\triangleright f & = & k\triangleright(l\triangleright f) & & & &
\end{array}$$

Since the eight diagrams in Figure 9 cover all cases of single-edge and single-node critical pairs, we have the following proposition.

▶ **Proposition 9.** *Reduction $\rightharpoonup$ is locally confluent.*

The strong normalization property is stated without proof; the proofs can be found in the appendix of the technical report on HAL [17].

▶ **Theorem 10** (Strong normalization)**.** *Combinatorial-tree reduction is strongly normalizing.*

6 Combinatory lambda-calculus

To further illustrate the reduction process, we connect ICPs to the λ-calculus, via an explicit-substitution λ-calculus that we call the ***combinatory λ-calculus***. The calculus is a Curry–Howard interpretation of sequent calculus, of the kind studied by Graham-Lengrand [28]. We include constants c to match those of combinatorial trees.

▶ **Definition 11.** *The **combinatory λ-calculus** has **normal terms** N, M, **patterns** p, q, and **terms** S, T given by the following grammars.*

$$M, N ::= x \mid \langle M, N\rangle \mid \lambda p.\, M \mid M[p \leftarrow xN]$$

$$p, q ::= x \mid \langle p, q\rangle \qquad S, T ::= c \mid M[p_1 \leftarrow T_1, \ldots, p_n \leftarrow T_n]$$

*The **binding variables** $\mathsf{bv}(p)$ of p and the **free variables** $\mathsf{fv}(M)$ of M are as follows; in $M[p \leftarrow xN]$ we require that $\mathsf{fv}(M) \cap \mathsf{bv}(p) \neq \varnothing$, and in $\langle p, q\rangle$ that $\mathsf{bv}(p) \cap \mathsf{bv}(q) = \varnothing$.*

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{bv}(x) & = & x & \mathsf{bv}(\langle p,q\rangle) & = & \mathsf{bv}(p) \cup \mathsf{bv}(q) \\
\mathsf{fv}(x) & = & x & \mathsf{fv}(\langle M,N\rangle) & = & \mathsf{fv}(M) \cup \mathsf{fv}(N) \\
\mathsf{fv}(\lambda p.\, M) & = & \mathsf{fv}(M) - \mathsf{bv}(p) & \mathsf{fv}(M[p \leftarrow xN]) & = & (\mathsf{fv}(M) - \mathsf{bv}(p)) \cup \{x\} \cup \mathsf{fv}(N)
\end{array}$$

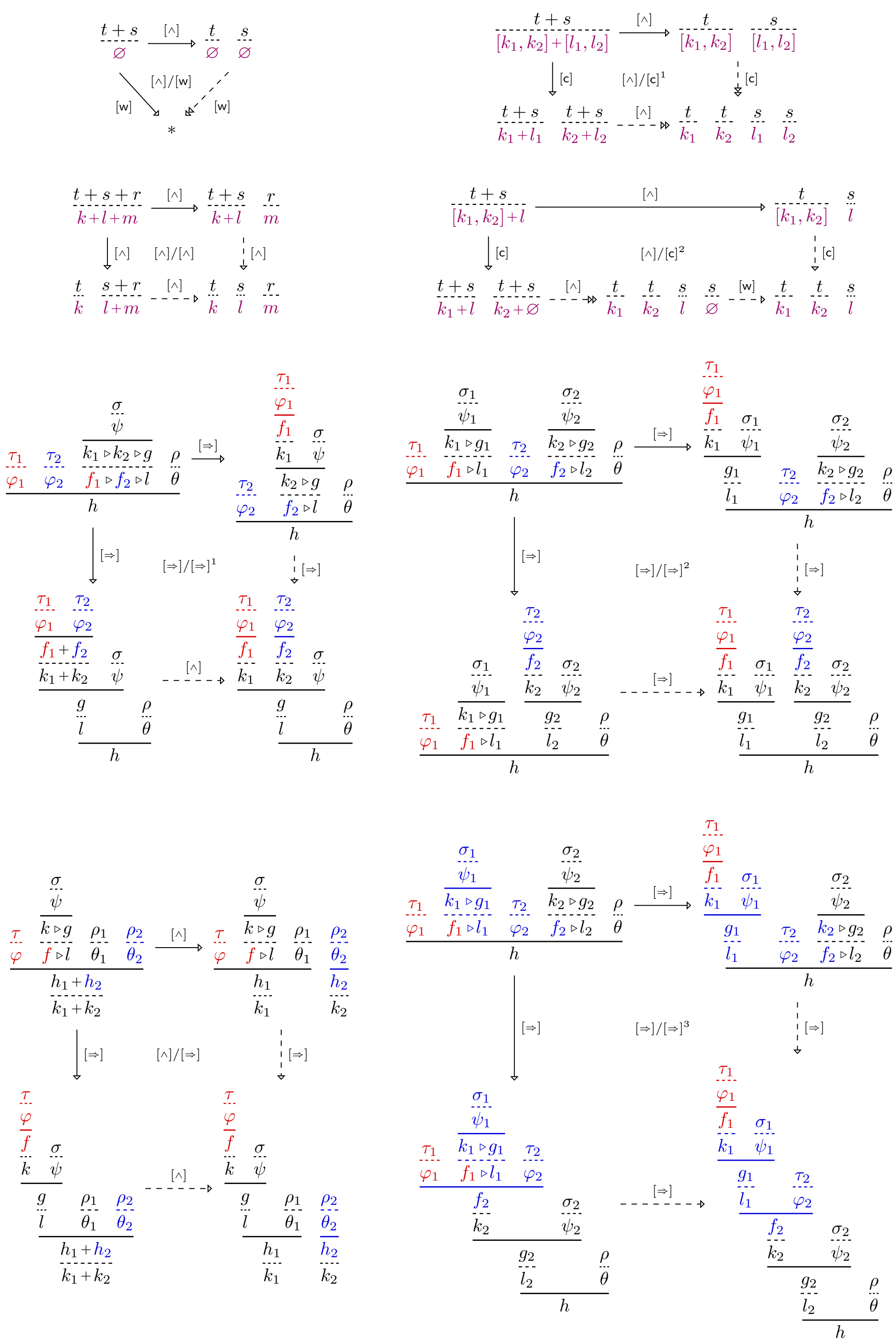


Figure 9 Single-edge and single-node confluence diagrams.

$$\frac{}{1 \vdash 1 \;\Mapsto\; x:P \vdash x:P}\;\langle\!\langle \mathsf{ax} \rangle\!\rangle \qquad \frac{\varphi, k, l \vdash f \;\Mapsto\; \Gamma, p:A, q:B \vdash M:C}{\varphi, k+l \vdash f \;\Mapsto\; \Gamma, \langle p,q\rangle : A \wedge B \vdash M:C}\;\langle\!\langle \wedge\mathsf{L} \rangle\!\rangle$$

$$\frac{\varphi \vdash f \;\Mapsto\; \Gamma \vdash M:C}{\varphi, \varnothing \vdash f \;\Mapsto\; \Gamma, p:A \vdash M:C}\;\langle\!\langle \mathsf{w} \rangle\!\rangle \qquad \frac{\varphi \vdash f \;\Mapsto\; \Gamma \vdash M:A \qquad \psi \vdash g \;\Mapsto\; \Delta \vdash N:B}{\varphi, \psi \vdash f+g \;\Mapsto\; \Gamma, \Delta \vdash \langle M, N\rangle : A \wedge B}\;\langle\!\langle \wedge\mathsf{R} \rangle\!\rangle$$

$$\frac{\varphi, k, l \vdash f \;\Mapsto\; \Gamma, p:A, p:A \vdash M:C}{\varphi, [k,l] \vdash f \;\Mapsto\; \Gamma, p:A \vdash M:C}\;\langle\!\langle \mathsf{c} \rangle\!\rangle \qquad \frac{\varphi \vdash f \;\Mapsto\; \Gamma \vdash N:A \qquad k, \psi \vdash g \;\Mapsto\; p:B, \Delta \vdash M:C}{\varphi, f \triangleright k, \psi \vdash g \;\Mapsto\; \Gamma, x:A \Rightarrow B, \Delta \vdash M[p \mapsfrom xN]:C}\;\langle\!\langle \Rightarrow\mathsf{L} \rangle\!\rangle$$

$$\frac{\varphi, k \vdash f \;\Mapsto\; \Gamma, p:A \vdash M:B}{\varphi \vdash k \triangleright f \;\Mapsto\; \Gamma \vdash \lambda p.\, M : A \Rightarrow B}\;\langle\!\langle \Rightarrow\mathsf{R} \rangle\!\rangle$$

Figure 10 From ICPs to simply-typed combinatory λ-terms.

In $\lambda p.\, M$, $M[p \mapsfrom xN]$, and $M[p_1 \leftarrow T_1, \ldots, p_n \leftarrow T_n]$ the variables in the patterns p and p_i bind in M. The construction $M[p \mapsfrom xN]$ is a ***shared application***, with a variable x as function and the term N as argument, where the pattern p may bind variables with multiple occurrences in M. The condition that $\mathsf{bv}(p)$ and $\mathsf{fv}(M)$ must intersect means at least one variable becomes bound; this corresponds to the condition (‡) on the rule $\Rightarrow\mathsf{L}$ for ICPs in Figure 2 (that the consequent of a left-implication must not be introduced by weakening). The construction $[p_1 \leftarrow T_1, \ldots, p_n \leftarrow T_n]$ is an ***environment***, and corresponds to attaching the subtrees to a node in a combinatorial tree. We abbreviate it by $[e]$, or $[p_1 \leftarrow T_1, e]$, etc.

▶ **Definition 12.** *Figure 10 gives the (non-deterministic)* ***translation*** *from ICPs to simply-typed, normal terms of the combinatory λ-calculus. We extend it to combinatorial trees as follows: $\Mapsto$ is the identity on constants, and if*

$$k_1, \ldots, k_n, \varphi \vdash f \qquad \Mapsto \qquad p_1 : A_1, \ldots, p_n : A_n, \Delta \vdash M : B$$

and if $t_i \;\Mapsto\; \Gamma_i \vdash T_i : A_i$ (with $t_i \neq \star$) for all $i \leq n$, then

$$\frac{\begin{array}{ccc} \dfrac{t_1}{k_1} & \cdots \;\dfrac{t_n}{k_n} & \dfrac{\star}{\varphi} \end{array}}{f} \qquad \Mapsto \qquad \Gamma_1, \ldots, \Gamma_n, \Delta \vdash M[p_1 \leftarrow T_1, \ldots, p_n \leftarrow T_n] : B\ .$$

The shared applications $[p \mapsfrom xN]$ of the combinatory λ-calculus are subject to permutations, creating an equivalence $\sim$ on terms. We define it below, where we abbreviate $[p \mapsfrom xN]$ by $[a]$, with $\mathsf{bv}(a) = \mathsf{bv}(p)$ and $\mathsf{fv}(a) = \{x\} \cup \mathsf{fv}(M)$.

$$\begin{array}{rcll}
\langle M[a], N\rangle & \sim & \langle M, N\rangle[a] & \mathsf{bv}(a) \cap \mathsf{fv}(N) = \varnothing \\
\langle M, N[a]\rangle & \sim & \langle M, N\rangle[a] & \mathsf{bv}(a) \cap \mathsf{fv}(M) = \varnothing \\
\lambda p.\,(M[a]) & \sim & (\lambda p.\, M)[a] & \mathsf{bv}(p) \cap \mathsf{fv}(a) = \varnothing \\
M[p \mapsfrom xN[a]] & \sim & M[p \mapsfrom xN][a] & \mathsf{bv}(a) \cap \mathsf{fv}(N) = \varnothing \\
M[a][b] & \sim & M[b][a] & \mathsf{bv}(b) \cap \mathsf{fv}(a) = \varnothing, \mathsf{bv}(a) \cap \mathsf{fv}(b) = \varnothing
\end{array}$$

The above equivalence factors out sequent calculus permutations. We will further assume combinatory λ-terms equivalent modulo the formula-isomorphisms (symmetry, associativity, and currying). These are factored out simply by considering patterns modulo these rules, but there is a catch: patterns and pairs are connected through cuts, or explicit substitutions, and laws must be applied to both simultaneously. We show an example with currying to demonstrate that a full definition is intricate, and leave it implicit.

$$M[z \mapsfrom x\langle P, Q\rangle][x \leftarrow \lambda\langle p, q\rangle.\, N] \quad \sim \quad M[z \mapsfrom yQ][y \mapsfrom xP][x \leftarrow \lambda p.\, \lambda q.\, N]$$

With the above equivalence on terms, the following is a direct corollary of local canonicity (Theorem 3).

▶ **Proposition 13.** *Combinatorial trees canonically represent typed combinatory λ-terms:*

$$S \sim T \quad \iff \quad \exists t.\ t \Mapsto S \ \wedge\ t \Mapsto T$$

We reduce combinatory λ-terms modulo the equivalence $\sim$. We write $\{T/x\}$ for the substitution of x by T, and if the patterns p, q are isomorphic as trees and $\mathsf{bv}(p) \cap \mathsf{bv}(q) = \varnothing$ then $\{q/p\}$ is the substitution induced by

$$\{\langle q_1, q_2\rangle / \langle p_1, p_2\rangle\} = \{q_1/p_1\}\{q_2/p_2\} \ .$$

▶ **Definition 14.** *Reduction of combinatory λ-terms modulo $\sim$ is by the following rules, where: $[e_P]$ and $[e_Q]$ bind only in P respectively Q; in $\langle\Rightarrow\rangle$ we require $x \notin \mathsf{fv}(P) \cup \mathsf{fv}(Q)$; in $\langle\mathsf{c}\rangle$ we require $\mathsf{bv}(q) \cap \mathsf{fv}(M) \neq \varnothing$; and in $\langle\mathsf{w}\rangle$ that $\mathsf{bv}(p) \cap \mathsf{fv}(M) = \varnothing$.*

$$\begin{aligned}
M[x \leftarrow y[e], e'] &\xrightarrow{\langle 1\rangle} M\{y/x\}[e, e'] \\
M[\langle p, q\rangle \leftarrow \langle P, Q\rangle[e_P, e_Q], e] &\xrightarrow{\langle\wedge\rangle} M[p \leftarrow P[e_P], q \leftarrow Q[e_Q], e] \\
P[p \leftarrow xQ][e_Q, x \leftarrow \lambda q. M[e], e_P] &\xrightarrow{\langle\Rightarrow\rangle} P[p \leftarrow M[q \leftarrow Q[e_Q], e], e_P] \\
M\{p/q\}[p \leftarrow T, e] &\xrightarrow{\langle\mathsf{c}\rangle} M[q \leftarrow T, p \leftarrow T, e] \\
M[p \leftarrow T, e] &\xrightarrow{\langle\mathsf{w}\rangle} M[e]
\end{aligned}$$

Comparing the reduction rules with the corresponding ones for ICPs in Figure 6, together with Proposition 13, gives:

▶ **Proposition 15.** *Reduction on ICPs and combinatory λ-terms (modulo equivalence) commutes with interpretation*

$$\begin{array}{ccc}
t & \xrightarrow{[x]} & s \\
\Downarrow & & \Downarrow \\
T & \xrightarrow{\langle x\rangle} & S
\end{array}$$

The comparison with λ-calculus allows us to make a further observation. ICP normalization is a form of *closed reduction* [7] (there called *weak reduction*), where a redex $(\lambda x.M)N$ may not be reduced if N contains free variables that are bound by the surrounding context. This has the benefit to implementation that alpha-conversion becomes unnecessary. Our construction of combinatorial trees is even stronger: it is impossible to construct such a redex, or to produce one by reduction. This can be observed from the combinatory λ-calculus, which does not support abstraction at the level of terms T, only at the level of normal terms.

Abstraction on terms can be introduced as a defined operation, called ***lambda-lifting*** [26]. The analogous operation on ICP combinatorial trees would be a transformation

$$\frac{\star :: A \quad \star :: \Gamma}{t :: B} \quad \mapsto \quad \frac{\star :: \Gamma}{t' :: A \Rightarrow B} \ .$$

We can perform it by abstracting over $\star :: A$ locally, in the node where it resides, and transform every node on the path from there to the root as follows,

$$\frac{k :: C \quad \varphi}{f :: D} \quad \mapsto \quad \frac{i \triangleright k :: A \Rightarrow C \quad \varphi}{i \triangleright f :: A \Rightarrow D}$$

where the port $k :: C$ is that on the path to $\star :: A$, and the arena morphism $i : [\![A]\!] \to [\![A]\!]$ is the identity on $[\![A]\!]$. In effect, one is threading the abstraction over A *through* the cuts in the tree, rather than adding it as a connection *outside* of them.

By way of example, below is the reduction corresponding to the ICP normalization sequence in Figure 7.

$$\begin{array}{ll|ll}
 & v[v \leftarrowtail gw][w \leftarrowtail yz][y \leftarrowtail ng][n \leftarrow \lambda f.\lambda x.x, g \leftarrow S, z \leftarrow 4] & & \dots \\
\sim & v[v \leftarrowtail gw][w \leftarrowtail yg][y \leftarrowtail nz][n \leftarrow \lambda x.\lambda f.x, z \leftarrow 4, g \leftarrow S] & \xrightarrow{\langle \mathsf{w} \rangle} & v[v \leftarrowtail gw][w \leftarrow x[x \leftarrow 4], g \leftarrow S] \\
\xrightarrow{\langle \Rightarrow \rangle} & v[v \leftarrowtail gw][w \leftarrowtail yg][y \leftarrow \lambda f.x[x \leftarrow z[z \leftarrow 4]], g \leftarrow S] & \xrightarrow{\langle 1 \rangle} & v[v \leftarrowtail gw][w \leftarrow 4, g \leftarrow S] \\
\xrightarrow{\langle 1 \rangle} & v[v \leftarrowtail gw][w \leftarrowtail yg][y \leftarrow \lambda f.x[x \leftarrow 4]], g \leftarrow S] & \xrightarrow{\langle \Rightarrow \rangle} & v[v \leftarrow 16] \\
\xrightarrow{\langle \mathsf{c} \rangle} & v[v \leftarrowtail gw][w \leftarrowtail yh][y \leftarrow \lambda f.x[x \leftarrow 4]], g \leftarrow S, h \leftarrow S] & \xrightarrow{\langle 1 \rangle} & 16 \\
\xrightarrow{\langle \Rightarrow \rangle} & v[v \leftarrowtail gw][w \leftarrow x[f \leftarrow h[h \leftarrow S], x \leftarrow 4], g \leftarrow S] \dots & &
\end{array}$$

7 Supercombinators

Supercombinators [24] are the basis of an efficient implementation of functional programming [29]. The main reason for their efficiency is that expressions are compiled into trees (or graphs) over a fixed set of operators, each given as an instruction set that implements the appropriate reduction sequence.

▶ **Definition 16.** ***Supercombinators*** *C, D and* ***supercombinator expressions*** *E_X, F_X, where X is a set of variables, are given by the following grammars.*

$$C, D ::= \lambda x_1 \dots \lambda x_n . E_{\{x_1,\dots,x_n\}} \qquad E_X, F_X ::= x \in X \;\mid\; C \;\mid\; F_X\, E_X$$

The set X restricts which variables may occur free in a supercombinator expression, so that each supercombinator is a closed term; we may omit it as superscript for brevity. The grammar for supercombinators C may be extended to include constants. Reduction is *weak head reduction* on an expression $E_\varnothing$, as given by the rule below. It applies only at top-level, not in context, and if there are fewer than n arguments to a supercombinator with n abstractions, reduction halts.

$$(\lambda x_1 \dots \lambda x_n . E)\, F_1 \dots F_n\, F_{n+1} \dots F_{n+m} \;\mapsto\; E\{F_1/x_1\}\dots\{F_n/x_n\}\, F_{n+1} \dots F_{n+m}$$

During reduction, substitutions are applied only to the top-level $E_\varnothing$ expression, and not to supercombinators, which remain fixed. This allows them to be compiled into instruction sets to carry out the appropriate reduction by the rule $\mapsto$ above.

Structurally, supercombinators are trees or graphs where each node is a supercombinator C in which each occurring supercombinator D is considered as a *pointer* to the node for D. This is highly similar to combinatorial trees, which feature the same tree structure except with ICPs for nodes. The main dissimilarities between supercombinators and combinatorial trees are then as follows.

- Supercombinator reduction is by an abstract machine, where combinatorial-tree reduction is a variant of cut-elimination.
- Supercombinators are trees over β-normal λ-terms where abstractions may not occur under an application, where nodes in combinatorial trees are η-expanded β-normal sequent proofs modulo permutations.

These differences are conceptually shallow, but risk burying a formal comparison in technicalities. We will therefore interpret supercombinators in the combinatory λ-calculus instead (which, mainly, does not require η-expansion), and simulate reduction only up to explicit substitutions.

▶ **Definition 17.** *The relations $\blacktriangleright$ and $\triangleright$, defined inductively below, interpret supercombinators respectively supercombinator expressions into the combinatory λ-calculus.*

$$\frac{E \triangleright M[e]}{\lambda x_1 \dots x_n.E \blacktriangleright (\lambda x_1 \dots x_n.M)[e]} \qquad \frac{}{x \triangleright x} \qquad \frac{C \blacktriangleright T}{C \triangleright x[x \leftarrow T]} \qquad \frac{E \triangleright x[a_1]\dots[a_k][e] \qquad F \triangleright M[f]}{EF \triangleright y[y \hookleftarrow xM][a_1]\dots[a_k][e,f]}$$

Note how this indeed translates a supercombinator to a term $(\lambda x_1 \dots \lambda x_n.N)[e]$ consisting of a normal form $\lambda x_1 \dots \lambda x_n.N$ with a subtree for each occurring supercombinator in the explicit substitutions $[e]$. To simulate reduction, a reduct is translated as follows.

$$\dfrac{\dfrac{\dfrac{E \triangleright M[e]}{\lambda x_1 \dots \lambda x_n.E \blacktriangleright (\lambda x_1 \dots \lambda x_n.M)[e]}}{\lambda x_1 \dots \lambda x_n.E \triangleright y[y \leftarrow (\lambda x_1 \dots \lambda x_n.M)[e]]} \qquad F_1 \triangleright N_1[f_1] \quad \dots \quad F_n \triangleright N_n[f_n]}{(\lambda x_1 \dots \lambda x_n.E)\, F_1 \dots F_n \triangleright z_n[z_n \hookleftarrow z_{n-1}N_n]\dots[z_1 \hookleftarrow yN_1][y \leftarrow (\lambda x_1 \dots \lambda x_n.M)[e], f_1, \dots, f_n]}$$

Reduction for this term proceeds as follows.

$$\begin{aligned}
& z_n[z_n \hookleftarrow z_{n-1}N_n]\dots[z_2 \hookleftarrow z_1N_2][z_1 \hookleftarrow yN_1][y \leftarrow (\lambda x_1.\lambda x_2 \dots \lambda x_n.M)[e], f_1, f_2, \dots, f_n] \\
\xrightarrow{\langle\Rightarrow\rangle}\ & z_n[z_n \hookleftarrow z_{n-1}N_n]\dots[z_2 \hookleftarrow z_1N_2][z_1 \leftarrow (\lambda x_2 \dots \lambda x_n.M)[x_1 \leftarrow N_1[f_1], e], f_2, \dots, f_n] \\
\xrightarrow{\langle\Rightarrow\rangle}\!\!\!\!\rightarrow\ & z_n[z_n \leftarrow M[x_1 \leftarrow N_1[f_1], \dots, x_n \leftarrow N_n[f_n], e]]
\end{aligned}$$

The result corresponds to the supercombinator reduct $E\{F_1/x_1\}\dots\{F_n/x_n\}$, except that the explicit substitutions $[x_i \leftarrow N_i[f_i]]$ are not evaluated as substitutions. They cannot be: combinatory λ-term reduction does not differentiate between the interpretation of the top-level supercombinator expression $E_\varnothing$ on which reduction takes place, and which does admit substitutions, and internal subcombinator expressions which do not. We will therefore contend ourselves with the "moral" equivalence of both reductions.

8 Lambda-calculus

To complete the exposition, we map the combinatory λ-calculus onto the regular λ-calculus with pairing. We have the following terms and rewrite rules, where $i \in \{1,2\}$.

$$M, N ::= x \mid \lambda x.M \mid MN \mid \pi_i M \mid \langle M, N\rangle \qquad (\lambda x.M)N \rightarrow_\beta M\{N/x\} \qquad \pi_i\langle M_1, M_2\rangle \rightarrow_\pi M_i$$

The translation from combinatory λ-terms into λ-terms $\lfloor\cdot\rfloor$ is as follows, where we substitute for a pattern via $\{M/\langle p,q\rangle\} = \{\pi_1 M/p, \pi_2 M/q\}$.

$$\begin{aligned}
\lfloor x \rfloor &= x \\
\lfloor \langle M, N\rangle \rfloor &= \langle \lfloor M \rfloor, \lfloor N \rfloor\rangle \\
\lfloor \lambda p.M \rfloor &= \lambda x.\lfloor M \rfloor\{x/p\} \\
\lfloor M[p \hookleftarrow xN] \rfloor &= \lfloor M \rfloor\{x\lfloor N \rfloor/p\} \\
\lfloor M[p_1 \leftarrow T_1, \dots, p_n \leftarrow T_n] \rfloor &= \lfloor M \rfloor\{\lfloor T_1 \rfloor/p_1\}\dots\{\lfloor T_n \rfloor/p_n\}
\end{aligned}$$

The combined translation then takes ICP combinatorial trees to λ-terms. As with the combinatory λ-calculus, we assume λ-terms equivalent ($\sim$) modulo formula-isomorphisms (symmetry, associativity, currying). Sequent permutations are already naturally factored out, but at the cost of exponential growth. We will demonstrate this here.

In the combinatory λ-calculus, the reason that an application must occur in an explicit substitution is precisely that the consequent of a left-implication may have been contracted, the situation highlighted in the introduction:

$$\dfrac{\Gamma \vdash A \qquad \dfrac{B, B, \Delta \vdash C}{B, \Delta \vdash C}\,\mathsf{c}}{\Gamma, A \Rightarrow B, \Delta \vdash C}\,{\Rightarrow}\mathsf{L} \qquad \approx \qquad \dfrac{\dfrac{\Gamma \vdash A \qquad \dfrac{\Gamma \vdash A \quad B, B, \Delta \vdash C}{B, \Gamma, A \Rightarrow B, \Delta \vdash C}\,{\Rightarrow}\mathsf{L}}{\Gamma, A \Rightarrow B, \Gamma, A \Rightarrow B, \Delta \vdash C}\,{\Rightarrow}\mathsf{L}}{\Gamma, A \Rightarrow B, \Delta \vdash C}\,\mathsf{c}$$

The corresponding equivalence on combinatory terms is:

$$M\{p/q\}[p \leftarrow\!\!\shortmid xN] \qquad \approx \qquad M[q \leftarrow\!\!\shortmid xN][p \leftarrow\!\!\shortmid xN]$$

(where $\mathsf{bv}(q) \cap \mathsf{fv}(M) \neq \varnothing$), while both translate to the same λ-term $\lfloor M \rfloor \{x \lfloor N \rfloor / p\}$. Repeated duplication incurred in this way gives rise to exponential growth.

Let ***strong equivalence*** $S \approx T$ on combinatory λ-terms be the equivalence generated by the above and $\sim$. We have the following proposition.

▶ **Proposition 18.** *For combinatory λ-terms S, T, we have*

$$S \approx T \iff \lfloor S \rfloor = \lfloor T \rfloor \ .$$

9 Conclusion

We have given a direct and natural account of normalization for intuitionistic combinatorial proofs. We believe our approach of *external rewriting*, here manifested in the notion of *combinatorial tree*, applies much more broadly, in the following two ways.

Firstly, specifically for the present, intuitionistic case, our notion of composition is highly abstract: what we have are simply trees of normal forms, with the natural reduction rules given by the meta-level sequent calculus. As a generalization of super-combinators, a correspondence we aim to make more precise in future work, we hope that our approach leads to improvements in compiler design. Perhaps the ability to express all normal forms, and the more fine-grained reduction steps, will allow more efficient program transformations, while retaining the benefits of super-combinators.

Secondly, our aim has been towards a notion of composition for combinatorial proofs in general, and to illustrate this we briefly sketch how our construction applies to classical combinatorial proofs [18]. Our combinatorial trees generalize to *combinatorial graphs*, which are still connected and acyclic (i.e. still a mathematical tree), but without a designated root. Nodes are classical combinatorial proofs over one-sided sequents, and edges are cuts connecting dual formulae. As may be expected of a semantic account of classical cut-elimination, one does not obtain strong normalization because of the Lafont examples [14] (specifically, a cut on two contracted formulae), but weak normalization is expected to hold. This is the subject of current work.

References

1. Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Information and Computation*, 163:409–470, 1996.
2. Matteo Acclavio and Lutz Straßburger. On combinatorial proofs for logics of relevance and entailment. In Rosalie Iemhoff and Michael Moortgat, editors, *26th Workshop on Logic, Language, Information and Computation (WoLLIC 2019)*. Springer, 2019.

3 Matteo Acclavio and Lutz Straßburger. On combinatorial proofs for modal logic. In Serenella Cerrito and Andrei Popescu, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings*, volume 11714 of *Lecture Notes in Computer Science*, pages 223–240. Springer, 2019. doi:10.1007/978-3-030-29026-9_13.

4 Gianluigi Bellin and Jacques van de Wiele. Subnets of proof-nets in MLL^-. In *Advances in Linear Logic*, pages 249–270, 1995.

5 Ulrich Berger and Helmut Schwichtenberg. An inverse of the evaluation functional for typed λ-calculus. In *6th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 203–212, 1991.

6 Kim B. Bruce, Roberto Di Cosmo, and Giuseppe Longo. Provable isomorphisms of types. *Mathematical Structures in Computer Science*, 2(2):231–247, 1992.

7 Naim Çağman and J. Roger Hindley. Combinatory weak reduction in lambda calculus. *Theoretical Computer Science*, 198(1–2):239–249, 1998.

8 Roberto Di Cosmo. A short survey of isomorphisms of types. *Mathematical structures in computer science*, 15(5):825–838, 2005.

9 Gerhard Gentzen. Untersuchungen über das logische Schließen I, II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1934–1935. English translation in: The Collected Papers of Gerhard Gentzen, M.E. Szabo (ed.), North-Holland 1969.

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

11 Jean-Yves Girard. Geometry of interaction 2: Deadlock-free algorithms. In *International Conference on Computer Logic*, pages 76–93, 1988.

12 Jean-Yves Girard. Proof-nets: the parallel syntax for proof-theory. *Logic and Algebra*, pages 97–124, 1996.

13 Jean-Yves Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(3):301–506, 2001. doi:10.1017/S096012950100336X.

14 Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types.* Cambridge University Press, 1989.

15 Willem Heijltjes. Proof nets for additive linear logic with units. In *IEEE 26th Annual Symposium on Logic in Computer Science (LICS)*, pages 207–216, 2011.

16 Willem Heijltjes, Dominic Hughes, and Lutz Straßburger. Intuitionistic proofs without syntax. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2019.

17 Willem B. Heijltjes, Dominic J. D. Hughes, and Lutz Straßburger. Normalization without syntax. Technical Report HAL-03654060, Inria, 2022. URL: https://hal.inria.fr/hal-03654060.

18 Dominic Hughes. Proofs without syntax. *Annals of Mathematics*, 164(3):1065–1076, 2006.

19 Dominic Hughes and Willem Heijltjes. Conflict nets: efficient locally canonical mall proof nets. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2016.

20 Dominic Hughes and Rob van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. *Transactions on Computational Logic*, 6(4):784–842, 2005.

21 Dominic J. D. Hughes. First-order proofs without syntax, 2019. arXiv preprint 1906.11236. arXiv:1906.11236.

22 Dominic J. D. Hughes, Lutz Straßburger, and Jui-Hsuan Wu. Combinatorial proofs and decomposition theorems for first-order logic. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470579.

23 Dominic J.D. Hughes. Simple free star-autonomous categories and full coherence. *Journal of Pure and Applied Algebra*, 216(11):2386–2410, 2012.

24 R.J.M. Hughes. Super-combinators: a new implementation method for applicative languages. In *ACM Symposium on Lisp and Functional Programming*, pages 1–10, 1982.

25 J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Information and Computation*, 163(2):285–408, 2000.

26 Thomas Johnsson. Lambda lifting: Transforming programs to recursive equations. In *Conference on Functional Programming Languages and Computer Architecture*, volume 201 of *LNCS*, pages 190–203, 1985.

27 Yves Lafont. Interaction nets. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 95–108, 1990.

28 Stéphane Lengrand. *Normalisation and equivalence in proof theory and type theory.* PhD thesis, University of St. Andrews, 2006.

29 Simon L. Peyton-Jones. *The implementation of functional programming languages.* Prentice Hall, 1987.

30 Andrea Aler Tubella and Lutz Straßburger. Introduction to deep inference. Lecture notes for ESSLLI'19, 2019. URL: `https://hal.inria.fr/hal-02390267`.