Constructive validity of a generalized Kreisel-Putnam rule

Ivo Pezlar[*]

Abstract

In this paper, we propose a computational interpretation of the generalized Kreisel-Putnam rule, also known as the generalized Harrop rule or simply the Split rule, in the style of BHK semantics. We will achieve this by exploiting the Curry-Howard correspondence between formulas and types. First, we inspect the inferential behavior of the Split rule in the setting of a natural deduction system for intuitionistic propositional logic. This will guide our process of formulating an appropriate program that would capture the corresponding computational content of the typed Split rule. Our investigation can also be reframed as an effort to answer the following question: is the Split rule constructively valid in the sense of BHK semantics? Our answer is positive for the Split rule as well as for its newly proposed general version called the S rule.

Keywords: inferentialism; proof-theoretic semantics; BHK semantics; Kreisel-Putnam rule; Harrop rule; Split rule; constructive validity; Curry-Howard correspondence

1 Introduction

The Harrop rule (Harrop (1960)) also known as the Independence of Premise rule or the Kreisel-Putnam rule:

$$\frac{\neg C \to (A \vee B)}{(\neg C \to A) \vee (\neg C \to B)}\text{ Harrop}$$

is an intriguing rule. It is an admissible but not a derivable rule of intuitionistic logic (Iemhoff (2001)), despite being proof-theoretically valid (Piecha et al. (2014)) in a variant of Dummett-Prawitz-style semantics (Prawitz (1971), Prawitz (1973)). If we add it to intuitionistic logic, we obtain the Kreisel-Putnam logic (Kreisel and Putnam (1957)), which is stronger than intuitionistic logic yet still has the disjunction property (whenever $A \vee B$ is a theorem, either A or B is a theorem), previously thought to be property specific to intuitionistic

[*]Czech Academy of Sciences, Institute of Philosophy, Jilska 1, Prague 1, 110 00.

logic (Łukasiewicz (1952)). Furthermore, it is admissible in any intermediate logic (Prucnal (1979)).

Yet, its generalized version, which we call the Split rule:

$$\frac{C \to (A \vee B)}{(C \to A) \vee (C \to B)}\,\text{Split}$$

where C is restricted to Harrop formulas,[1] is arguably even more interesting. If we add it to intuitionistic logic, we obtain inquisitive intuitionistic logic (Punčochář (2016), Ciardelli et al. (2020)), which has both the disjunctive property and the structural completeness property (enjoyed by classical logic: every admissible rule is derivable), again it can be shown to be proof-theoretically valid in a variant of Dummett-Prawitz-style semantics (Stafford (2021)), yet it is not closed under uniform substitution. Furthermore, it is admissible in any intermediate logic (Minari and Wronski (1988)) and it also makes a surprising appearance in domain logics (Abramsky (1991), Zhang (1991)) and we are confident that this list is not complete.[2]

It is worth mentioning that if we generalize the Split rule even further by lifting the restriction of C to Harrop formulas, we get the Full Split rule:

$$\frac{C' \to (A \vee B)}{(C' \to A) \vee (C' \to B)}\,\text{Full Split}$$

where C' is an arbitrary formula. This rule is also of significant interest: if we add it to intuitionistic logic, we obtain Gödel-Dummett logic (Gödel (1932), Dummett (1959)) which in turn found its uses in, e.g., relevance logic (Dunn and Meyer (1971)) or fuzzy logic (Hájek (1998)). However, in this paper, we will be interested exclusively in the Split rule restricted to Harrop formulas. Therefore, any future mention of the Split rule will always refer to the restricted variant, never to the Full Split rule.

Despite its significance, the Split rule itself remains mostly unexplored, especially in terms of its proof-theoretic meaning and computational content (a recent exception to this is Condoluci and Manighetti (2018) examining the admissibility of the related Harrop rule from the computational view). In this paper, we fill this gap and propose a computational interpretation of the Split rule. We will achieve this by exploiting the Curry-Howard correspondence (Curry and Feys (1958), Howard (1980)) between formulas and types (also known as the propositions-as-types principle). First, we inspect the inferential behavior of the Split rule in the setting of a natural deduction system for intuitionistic propositional logic. This will then guide our process of formulating an appropriate program that would capture the corresponding computational content of the typed Split rule. Our investigation can be thus also reframed as an effort to

[1] Formulas in which every disjunction occurs only within the antecedents of implications (a more formal definition is presented in Section 2.1).

[2] I would like to thank Vít Punčochář for bringing this rule and its unexpected appearances in domain logics to my attention.

answer the following question: is the Split rule *constructively valid* in the sense of BHK semantics? And by this we mean:

> Can we find an effective function that would transform arbitrary proofs of the premise of the Split rule into proofs of its conclusion?

Our answer is positive: we propose a function S for a general version of the typed Split rule which we call the S rule:

$$\dfrac{\begin{matrix}[C]\\ \vdots \\ A \vee B\end{matrix} \qquad \begin{matrix}[C \to A]\\ \vdots \\ D\end{matrix} \qquad \begin{matrix}[C \to B]\\ \vdots \\ D\end{matrix}}{D}\ \mathrm{S}$$

where C is restricted to Harrop formulas, and show that it can be used to justify the standard Split rule as well.

Structure. This paper is structured as follows: In Section 2 we briefly introduce the basic concepts presumed by the paper, including the distinction between introduction and elimination rules of natural deduction (2.2). Readers familiar with the natural deduction framework can skip this section. In Section 3 we determine what kind of a rule is the Split rule in relation to the distinction between introduction and elimination rules and propose its generalization called the S rule (3.1). Furthermore, we examine the relationship between the Split rule and the S rule, including their justifications (3.2). And finally in Section 4 we introduce the typed version of the S rule and its corresponding function S.

2 Preliminaries

In this paper, we will be interested only in propositional logic. Specifically, as a starting point, we choose natural deduction for intuitionistic propositional logic (IPC) with informal computational semantics based on the Brouwer-Heyting-Kolmogorov (BHK) interpretation. This choice is motivated by this system's innate affinity towards the formulas-as-types principle, also known as the Curry-Howard correspondence. From Section 4 onwards, we also adopt a propositional fragment of Martin-Löf's constructive type theory with no dependent types, which can be seen as a formalization of the computational semantics of intuitionistic propositional logic. We presuppose basic familiarity with these systems on the reader's side. However, to make the presentation as accessible as possible we try to keep the discussion generally informal whenever reasonable.

The language $\mathcal{L}$ of IPC is a set of formulas $A, B, C, \ldots$ built in the usual way from atomic formulas (propositional variables) $p, q, r, \ldots$ and standard logical constants $\wedge$, $\vee$, $\to$, $\bot$. Negation is defined as $A \to \bot$.

The meaning of connectives is assumed to be given in terms of canonical proofs, which can be thought of as immediate justifications or direct grounds for asserting formulas[3] of the corresponding form. The shape of these canonical proofs is prescribed by introduction rules. We discuss this more at length in the next section but, as an example, the meaning of the conjunction $\wedge$ connective is specified by the conjunction introduction rule:

$$\frac{A \qquad B}{A \wedge B} \wedge\mathrm{I}$$

which corresponds to the BHK clause for conjunction: a proof of the formula $A \wedge B$ consists of a proof of A and a proof of B. Asserting $A \wedge B$ means that we have proofs of A and B at our disposal. Furthermore, using the formulas-as-types principle, we can make these observations more precise and state that the canonical proof of the formula $A \wedge B$ has the form (a, b) where a is a proof object[4] of A and b is a proof object of B. If we have such proof, we can make the judgment, also called assertion, $(a, b) : A \wedge B$ that informs us that $A \wedge B$ is true. Judgments such as $(a, b) : A \wedge B$ are called categorical judgments as they depend on no assumptions. Judgments depending on assumptions are called hypothetical judgments. There are also noncanonical proofs, which can be thought of as delayed justifications or indirect grounds for asserting the corresponding formulas. Noncanonical proofs, however, have to be reducible to the canonical proofs, otherwise, they are meaningless. From the perspective of the formulas-as-types principle, we can view noncanonical proofs as programs and canonical proofs as their values (i.e., as programs that cannot be evaluated/simplified any further).

2.1 Harrop formulas and uniform substitution

Harrop formulas (Harrop (1956)), also known as Rasiowa-Harrop formulas (Rasiowa (1954)), are formulas that do not contain disjunction $\vee$ except in the antecedents of implications. More formally, we can inductively define the set of (propositional) Harrop formulas by the following clauses:

(a) any atomic formula and $\bot$ is a Harrop formula,

(b) if A and B are Harrop formulas, then $A \wedge B$ is a Harrop formula,

(c) if A is an arbitrary formula and B is a Harrop formula, then $A \rightarrow B$ is a Harrop formula.

Alternatively, we could also define Harrop formulas using the notion of a disjunction-free formula (i.e., a Harrop formula as a formula where the rightmost implication has a disjunction-free formula in the consequent) but we will prefer the inductive definition.

[3]Strictly speaking, what we are asserting are propositions, not formulas, but for simplicity, we will treat propositions and formulas as interchangeable terms in this paper.

[4]Proof objects (proof terms, or simply proofs when the meaning is clear from the context), either canonical or noncanonical, should be thought of as symbolic evidence that something is true and they do not carry epistemic force on their own. The carriers of epistemic force are demonstrations (logical derivations) that proceed from judgment(s) to another judgment.

Earlier we have mentioned that systems containing the Split rule (such as, e.g., inquisitive intuitionistic logic) are not closed under uniform substitution. This is because the Split rule itself is not closed under it. A rule (or an axiom) is said to be closed under uniform substitution if after applying a uniform substitution to it the resulting rule is still valid. More formally, let $R(p_1, \dots, p_n)$ be a rule in IPC where $p_1, \dots, p_n$ are the atomic formulas in R and let $\sigma(R(p_1, \dots, p_n))$ represent the result of substituting arbitrary formulas $A_1, \dots, A_n$ for all occurrences of $p_1, \dots, p_n$ in R, i.e., $R(A_1, \dots, A_n)$. Then, $R(p_1, \dots, p_n)$ is said to be closed under uniform substitution if and only if $\sigma(R(p_1, \dots, p_n))$ is a valid rule for any uniform substitution σ.

For example, let us consider the following instance of the Split rule which is valid:

$$\frac{p \to (q \lor r)}{(p \to q) \lor (p \to q)}\ \text{Split}$$

since p, as an atomic formula, is a Harrop formula. Now, let us apply a substitution σ where the formula p is uniformly replaced by the formula $s \lor t$:

$$\frac{(s \lor t) \to (q \lor r)}{((s \lor t) \to q) \lor ((s \lor t) \to q)}\ \text{Split}'$$

Since $s \lor t$ is not a Harrop formula, it is easy to see that the above substitution has produced a rule that is no longer valid. Thus, the Split rule is not closed under uniform substitution.

2.2 Introduction and elimination rules

The computational content of the natural deduction rules of IPC is closely tied to their inferential behavior which is in turn specified by the introduction and elimination rules (including reduction rules, which link them together) for specific logical connectives. Generally speaking, an introduction rule for a connective $\circ$ is a rule whose conclusion has $\circ$ as the main connective. An elimination rule is a rule that has this $\circ$-formula as one of its premises.[5]

Semantically speaking, introduction rules define the meanings of new connectives,[6] while elimination rules show how to use them. From this perspective, introduction rules are self-justifying as they effectively act as stipulations.[7] On the other hand, elimination rules require justification.[8] This justification is typically achieved by relating elimination rules to introduction rules via reduction rules: what can be derived from A by elimination rules is to be determined by

[5]See, e.g., Mancosu et al. (2021), p. 69.

[6]See Gentzen (1935), Gentzen (1969).

[7]See, e.g., Schroeder-Heister (2006), p. 532.

[8]This would not be the case, however, if we were to switch from the "verificationist" approach privileging introduction rules to the "pragmatist" approach that views elimination rules as self-justifying and introduction rules as in need of justification.

the premises from which was A canonically derived, i.e., derived by introduction rules for the main operator of A.

For example, the implication connective $\to$ is specified by the following implication introduction and elimination rules:

$$\begin{array}{c}[A]\\ \vdots \\ \dfrac{B}{A \to B}\,{\to}\mathrm{I}\end{array} \qquad\qquad \dfrac{A \to B \qquad A}{B}\,{\to}\mathrm{E}$$

The introduction rule $\to$I tells us what the canonical (meaning constituting) proofs of formulas of the form $A \to B$ should look like. Specifically, it tells us that in order to prove a formula of the form $A \to B$, i.e., *introduce* it into a proof, we first need to find a proof of B from an assumption that we have a proof of A, which then can be discharged. In other words, we need to find a procedure that takes an arbitrary proof of A and transforms it into a proof of B (in accordance with the BHK interpretation). The elimination rule $\to$E tells us what can we do with formulas of the form $A \to B$ in proofs. Specifically, it tells us that if we have a proof of $A \to B$ (a major premise) and a proof of A (a minor premise), then we can derive B and, in the process, *eliminate* $A \to B$ from the proof.

Note that the rules $\to$I and $\to$E are, in a way, inverted versions of each other: what goes into introducing $\to$ comes out when eliminating it, no more no less. In other words, if we apply the $\to$I rule and then apply $\to$E immediately after, nothing should be gained or lost in the proof, we are just making an unnecessary detour. To check this, consider the following derivation:

$$\dfrac{\begin{array}{c}[A]\\ \vdots \\ \dfrac{B}{A \to B}\,{\to}\mathrm{I}\end{array} \qquad A}{B}\,{\to}\mathrm{E}$$

It starts with deriving B (under the active assumption A, and possible other assumptions) and ends, again, with deriving B. The consecutive applications of $\to$I and $\to$E add no new information: it is just a roundabout way of getting where we started in the first place, i.e., the derivation of B. These detours in derivations can be removed in a process called a detour conversion or a reduction (see Prawitz (1965)). We can depict this process via the following "metarule" (where $\mathcal{D}$ represents a derivation):

$$\dfrac{\begin{array}{c}[A]^n\\ \mathcal{D} \\ \dfrac{B}{A \to B}\,{\to}\mathrm{I}^n\end{array} \qquad \begin{array}{c}\mathcal{D}'\\ A\end{array}}{B}\,{\to}\mathrm{E} \qquad \xRightarrow[\to\text{red}]{\text{reduces to}} \qquad \begin{array}{c}\mathcal{D}'\\ A\\ \mathcal{D}\\ B\end{array}$$

A proof with no detours is called a normal proof or a proof in a normal form.

Analogously, no new information should be gained if we apply introduction rules immediately after elimination rules. Consider, e.g., the following derivation:

$$\dfrac{\dfrac{A \to B \qquad [A]}{B}\ {\to}\mathrm{E}}{A \to B}\ {\to}\mathrm{I}$$

Similarly as above, it starts with $A \to B$ (and an active assumption A) and ends with deriving $A \to B$ (with A discharged). We can depict this process, sometimes called expansion, via the following metarule:

$$\begin{array}{c}\mathcal{D}\\ A \to B\end{array} \quad \xRightarrow[\to\mathrm{exp}]{\text{expands to}} \quad \begin{array}{c}\mathcal{D}\\ \dfrac{\dfrac{A \to B \qquad [A]}{B}\ {\to}\mathrm{E}}{A \to B}\ {\to}\mathrm{I}\end{array}$$

When introduction and elimination rules behave in this way, i.e., when elimination rules do not allow us to derive more (i.e., they are not too strong = local soundness, see Pfenning and Davies (2001)) or less (i.e., they are not too weak = local completeness) than the introduction rules justify, it is said that they are harmonious (see Dummett (1991), Tennant (1978)). For a famous example of introduction and elimination rules that are not harmonious, see Prior's Tonk (Prior (1960)).

2.3 Formulas-as-types interpretation

The computational content corresponding to implication introduction and elimination rules can be thought of as expressed by the notions of function abstraction and function application from lambda calculus. If we decorate the rules with the appropriate proof objects, we obtain the typed variants of →I and →E:

$$\dfrac{\begin{array}{c}[x : A]\\ \vdots \\ b(x) : B\end{array}}{\lambda x.b(x) : A \to B}\ {\to}\mathrm{I} \qquad\qquad \dfrac{c : A \to B \qquad a : A}{\mathsf{ap}(c, a) : B}\ {\to}\mathrm{E}$$

The function constants λ and ap that appear in the conclusions of the rules can be understood as the constructor and the selector for the type $A \to B$, respectively. Constructors give us as values canonical objects of the corresponding types (they show us how to inhabit them), and selectors then show us how we can define functions (via pattern matching and/or recursion) on the types specified by the constructors. Thus, selectors are functions operating on the type specified by constructors. In the case of →I and →E, λ constructs proof objects of the type $A \to B$, i.e., essentially lambda codings of functions from A to B, while the selector ap shows us how we can apply these lambda codings of functions to arguments.

We will need one more rule that will show us how to compute the program specified by the selector, specifically, how to compute the function application $\mathsf{ap}(c, a)$ when c has the form of a canonical proof object, i.e., $c = \lambda x.b(x)$. This is achieved by the following rule that corresponds to the metarule $\to$red for detour reduction we discussed earlier:

$$\frac{\begin{array}{c}[x : A] \\ \vdots \\ b(x) : B\end{array} \qquad a : A}{\mathsf{ap}(\lambda x.b(x), a) = b(a) : B} \to\mathrm{C}$$

where $b(a)$ is the result of substituting a for x in b. This kind of rule is called computation rule:[9] it shows how selectors operate on the canonical proof objects generated by the constructors of the corresponding type. Hence, we can say that computation rules link programs (i.e., the functions/selectors appearing in the conclusions of typed elimination rules) to the corresponding constructors (i.e., values appearing in the conclusions of typed introduction rules).

By adding Split to IPC we are going beyond intuitionistic logic, so we have to make a few comments about the formulas-as-types principle as it was initially intended for intuitionistic reasoning only. The standard way of carrying over this principle to stronger logics than intuitionistic is to simply assume that the new axioms of the stronger logic hold for arbitrary formulas and assign them corresponding proof objects in the spirit of the formulas-as-types principle.

For example, in the case of classical logic, we might assume that the law of excluded middle $A \vee \neg A$ holds for arbitrary formulas and assign it the proof object lem, thus obtaining the judgment $\mathsf{lem} : A \vee \neg A$. The issue here is, of course, that nobody knows how the proof object lem is supposed to be computed. If we did we would be in possession of a universal decidability procedure for every formula.[10] Thus, the function lem effectively represents an unexecutable black box program.

With the Split rule, we approach the issue analogously. We adopt a new Split axiom schema:

$$(C \to (A \vee B)) \to ((C \to A) \vee (C \to B))$$

where C is restricted to Harrop formulas, transform it into a rule form (using its antecedent as the premise and its consequent as the conclusion for the rule) and assign it corresponding proof objects in the style of the formulas-as-types principle:

$$\frac{f : C \to (A \vee B)}{\mathsf{s}(f) : (C \to A) \vee (C \to B)} \,\mathrm{Split}$$

[9] Note that we use the explicit style of computation rules with displayed premises, which we adopt from Martin-Löf (1984).

[10] Recall that in the constructive setting the law of excluded middle is not a "meaningless" tautology but a judgment of decidability of the proposition A.

where f represents an arbitrary proof object of $C \to (A \vee B)$ and s the function associated with the Split rule. The question will then become: how do we compute $\mathsf{s}(f)$? Can we find a general procedure for evaluating the s function (program) and thus expressing the computational content of the corresponding rule? Our answer will be positive but it will require some further generalizations of the Split rule.

Now, let us examine the Split rule.

3 Generalizing the Split rule

3.1 The Split rule

What kind of a rule is the Split rule from the perspective of introduction and elimination rules of natural deduction?[11] For convenience, let us repeat the rule, also recall that C is restricted to Harrop formulas:

$$\frac{C \to (A \vee B)}{(C \to A) \vee (C \to B)}\ \text{Split}$$

At first look, it seems to be either an introduction rule for the disjunction connective (since $\vee$ appears as the main connective in the conclusion) or an elimination rule for either implication or disjunction, since those are the connectives appearing in the premise.

First, let us consider it as an introduction rule for disjunction. In IPC, the meaning of the $\vee$ connective is already fixed via its standard introduction rules:

$$\frac{A}{A \vee B}\ \vee\mathrm{I}_L \qquad \frac{B}{A \vee B}\ \vee\mathrm{I}_R$$

which are taken as fully specifying the meaning of disjunction as a sort of weakening of A or B into $A \vee B$. So, adding a supplementary introduction rule in the form of the Split rule would not only be unwarranted but would also shift the meaning of disjunction since viewing an inference from $C \to (A \vee B)$ to $(C \to A) \vee (C \to B)$ as weakening does not seem appropriate. Moreover, note that disjunction appears in the premise of the Split rule (i.e., it presupposes we have already introduced $\vee$ and thus understand what it means[12]). Hence, it does not seem reasonable to view the Split rule as a disjunction introduction-like rule.[13]

[11] Recall that the computational content is tied to the inferential content which is then tied to the introduction and elimination rules and their reductions. Thus, we want to position the Split rule within the introduction and elimination rules environment to guide our investigations of its computational content.

[12] In general, the fact that a connective we want to define via an introduction rule already appears among the premises of that rule doesn't necessarily result in paradoxes (see, e.g., Tranchini (2019), Pezlar (2021)) since we can have recursive definitions but this is not the case here.

[13] When discussing the Split rule, we will talk about introduction-like and elimination-like rules to distinguish them from the standard introduction and elimination rules associated with connectives.

So, by default, it seems to be an elimination-like rule (or rather more generally, a non-introduction rule) either for implication or disjunction. Both choices are feasible but, for the sake of space, in this paper, we will investigate only the second option as it leads to a simpler generalization. Note, however, that if we really want to treat the Split rule as a disjunction elimination-like rule, we need disjunction $\vee$ to be the main connective of its premise, otherwise, it would not fit the general pattern of elimination rules. To get around this issue, we decompose the original premise of the Split rule into a hypothetical judgment. The resulting rule, built in the style of standard disjunction elimination rule, then looks as follows:

$$\dfrac{\begin{matrix}[C] \\ \vdots \\ A \vee B\end{matrix} \qquad \begin{matrix}[C \to A] \\ \vdots \\ D\end{matrix} \qquad \begin{matrix}[C \to B] \\ \vdots \\ D\end{matrix}}{D}\,\mathrm{S}$$

where C is restricted to Harrop formulas. We will call this the S rule and it can be read as follows: if we derive $A \vee B$ under the assumption C and furthermore we can derive D separately from both $C \to A$ and $C \to B$, then we can proceed to D and discharge the assumptions C, $C \to A$, and $C \to B$.

For now, we will leave the question of justification of this rule open and return to it in Section 4 once we have introduced its typed variant.

Example. To get a better idea of how the S rule works, let us demonstrate it in practice. Consider the following two formulas:

$$(p \to (q \vee r)) \to ((p \to q) \vee (p \to r))$$

and

$$((s \vee t) \to (q \vee r)) \to (((s \vee t) \to q) \vee ((s \vee t) \to r))$$

With the S rule, we can prove the first formula as follows:

$$\dfrac{\dfrac{\dfrac{[p \to (q \vee r)]^2 \qquad [p]^1}{q \vee r}\to\mathrm{E} \qquad \dfrac{[p \to q]^3}{(p \to q) \vee (p \to r)}\vee\mathrm{I}_L \qquad \dfrac{[p \to r]^4}{(p \to q) \vee (p \to r)}\vee\mathrm{I}_R}{(p \to q) \vee (p \to r)}\mathrm{S}_{1,3,4}}{(p \to (q \vee r)) \to ((p \to q) \vee (p \to r))}\to\mathrm{I}_2$$

However, we are not able to prove the second formula because its proof would require us to assume instead of an atomic p, which is a Harrop formula (since every atom is a Harrop formula), a formula $q \vee r$ which is not a Harrop formula. Thus, we cannot apply the S rule since C is restricted to Harrop formulas only.[14]

Before we progress towards our main objective of analyzing the computational content of the S rule in Section 4, let us first take a closer look at how it relates to the Split rule.

[14]Recall that the Split rule and/or the S rule are not closed under uniform substitution: the second formula is a substitution instance of the first formula with $(s \vee t)$ substituted for p.

3.2 The Split rule and the S rule

The relationship between the Split rule and the S rule is perhaps best understood as similar to the relationship between elimination rules and general (generalized, parallel) elimination rules.

What are general elimination rules? Simply put, they are elimination rules following the "indirect" pattern of disjunction elimination rule $\lor$E which can be seen as utilizing the principle of proof by induction: to show that an arbitrary D follows from $A \lor B$ it is sufficient to show that it follows from the "base cases", i.e., the canonical proofs of $A \lor B$ from A and from B. If we apply this style of reasoning to the elimination rules for other connectives of IPC we obtain corresponding general elimination rules which have a more general form than their standard variants but are logically equivalent.[15]

For example, the general elimination rules for conjunction $\land$ and implication $\to$ are as follows:

$$\dfrac{A \land B \qquad \begin{matrix}[A, B]\\ \vdots \\ D\end{matrix}}{D}\ {\land}\mathrm{GE} \qquad\qquad \dfrac{A \to B \qquad A \qquad \begin{matrix}[B]\\ \vdots \\ D\end{matrix}}{D}\ {\to}\mathrm{GE}$$

Let us comment briefly on the general elimination rule for implication. We can read it as follows: if we derive $A \to B$, and we derive some further consequences D from B (at that point without knowing whether A is true) and if it turns out that A is indeed true, then we will know that D is true as well and we can derive it (recall that the immediate justification for deriving $A \to B$ is the existence of a derivation of B under the assumption A. So if D can be derived from B, then it must be already derivable from A).

So, returning to the S rule, we can think of it as a general version of the Split rule, similarly as $\to$GE is a general version of $\to$E:

$$\dfrac{A \to B \qquad B}{B}\ {\to}\mathrm{E} \quad \text{generalizes to} \quad \dfrac{A \to B \qquad A \qquad \begin{matrix}[B]\\ \vdots \\ D\end{matrix}}{D}\ {\to}\mathrm{GE}$$

$$\dfrac{C \to (A \lor B)}{(C \to A) \lor (C \to B)}\ \mathrm{Split} \quad \text{generalizes to} \quad \dfrac{\begin{matrix}[C]\\ \vdots \\ A \lor B\end{matrix} \qquad \begin{matrix}[C \to A]\\ \vdots \\ D\end{matrix} \qquad \begin{matrix}[C \to B]\\ \vdots \\ D\end{matrix}}{D}\ \mathrm{S}$$

As we have mentioned, elimination rules and their general counterparts are equivalent. Does this hold for the Split rule and the S rule as well? As it turns out, they are indeed equivalent. This can be shown as follows. Suppose that we

[15]See, e.g., Schroeder-Heister (2014). General elimination rules have many useful properties. For example, they ensure the structural correspondence between natural deduction derivations and sequent calculus derivations required for the translation of the former to the latter. See Negri and von Plato (2001).

have at our disposal the S rule and further suppose that we have derived the premises for the Split rule, i.e., we have a derivation (i.e., we know the premise) $\begin{matrix} \mathcal{D}_1 \\ C \to (A \vee B) \end{matrix}$ of $C \to (A \vee B)$. Now, assuming $\mathcal{D}_1$ is either a canonical proof of $C \to (A \vee B)$ (i.e., we have immediate justification for asserting it) or that can be reduced to such a proof by a series of finite steps, it is of the form $\dfrac{\begin{matrix} [C] \\ \mathcal{D}'_1 \\ A \vee B \end{matrix}}{C \to (A \vee B)} \to\mathrm{I}$ where $\begin{matrix} C \\ \mathcal{D}'_1 \\ A \vee B \end{matrix}$ is a derivation of $A \vee B$ from C. From this derivation we can obtain by means of the S rule (and the $\vee$I rules) a derivation:

$$\dfrac{\begin{matrix} [C] \\ \mathcal{D'}_1 \\ A \vee B \end{matrix} \qquad \dfrac{[C \to A]}{(C \to A) \vee (C \to B)} \vee\mathrm{I}_L \qquad \dfrac{[C \to B]}{(C \to A) \vee (C \to B)} \vee\mathrm{I}_R}{(C \to A) \vee (C \to B)} \mathrm{S}$$

which is a derivation of the conclusion $(C \to A) \vee (C \to B)$ of the Split rule.

Next, the other direction. Let us suppose that the Split rule is at our disposal and that we have derived the premises of the S rule, i.e., we have a derivation $\begin{matrix} C \\ \mathcal{D}_1 \\ A \vee B \end{matrix}$ of $A \vee B$ from C, a derivation $\begin{matrix} C \to A \\ \mathcal{D}_2 \\ D \end{matrix}$ of D from $C \to A$, and a derivation $\begin{matrix} C \to B \\ \mathcal{D}_3 \\ D \end{matrix}$ of D from $C \to B$. Thus, since we have a derivation $\begin{matrix} C \\ \mathcal{D}_1 \\ A \vee B \end{matrix}$ (i.e., the first premise of the S rule), we can further assume that we can obtain a derivation $\begin{matrix} \mathcal{D}'_1 \\ C \to (A \vee B) \end{matrix}$ of $C \to (A \vee B)$ via the $\to$I rule.

Now, by an application of the Split rule using $\mathcal{D'}_1$ as a premise we can obtain a derivation:

$$\dfrac{\begin{matrix} \mathcal{D}'_1 \\ C \to (A \vee B) \end{matrix}}{(C \to A) \vee (C \to B)}$$

and finally by an application of the $\vee$E rule using this derivation, $\mathcal{D}_2$ and $\mathcal{D}_3$ as premises we can obtain a derivation of D, i.e., the conclusion of the S rule.

3.3 Justification of the Split rule

We have said that introduction rules are typically viewed as self-justifying in contrast to elimination rules that require further justification. Now, since we have decided to view the Split rule as an elimination-like rule it is in need of further justification as well. As we have mentioned, justification of the elimination rule involves specifying certain reduction procedures for derivations that were derived by elimination rules. For example, the implication elimination rule can be regarded as justified with respect to the reduction procedure $\to$red.

This line of reasoning is another way to understand the connection between the Split rule and the S rule: we can view the latter as a means of justifying the former. More specifically, we can consider the following reduction procedure for justifying the Split rule:[16]

$$\cfrac{\begin{array}{c}\mathcal{D}\\ C \to (A \vee B)\end{array}}{(C \to A) \vee (C \to B)}\,\text{Split} \quad \xRightarrow[\text{Split-red}]{\text{reduces to}}$$

$$\cfrac{\cfrac{\begin{array}{c}\mathcal{D}\\ C \to (A \vee B)\end{array} \qquad [C]^1}{A \vee B}\,\to\text{E} \qquad \cfrac{[C \to A]^2}{(C \to A) \vee (C \to B)}\,\vee\text{I}_L \qquad \cfrac{[C \to B]^3}{(C \to A) \vee (C \to B)}\,\vee\text{I}_R}{(C \to A) \vee (C \to B)}\,\text{S}_{1,2,3}$$

Furthermore, note that this reduction is distinct from the standard reductions as it relies on "external" rules. Namely, disjunction introduction rules, which are trivially justified as introduction rules, implication elimination rule (which is justified by its corresponding reduction rule $\to$red), and the S rule. Compare this with, e.g., the standard reduction procedure for implication $\to$red which relies only on given subderivations and invokes no other rules.

We can see that this justification of the Split rule relies, among other rules, on the S rule. The S rule – as an elimination-like rule itself – thus needs justification as well. Thus, we will also need to supply reduction procedures for it. Since we treat the S rule as a disjunction elimination-like rule, the notion of reduction/detour conversion still makes sense, as it is possible to introduce the disjunction $A \vee B$ under the assumption C and then immediately eliminate it via the S rule, which then constitutes an unnecessary detour in a derivation (analogously with the Split rule). The reduction rules we obtain are as follows:

$$\cfrac{\cfrac{\begin{array}{c}[C]\\ \mathcal{D}_1\\ A\end{array}}{A \vee B}\,\vee\text{I}_L \qquad \begin{array}{c}[C \to A]\\ \mathcal{D}_2\\ D\end{array} \qquad \begin{array}{c}[C \to B]\\ \mathcal{D}_3\\ D\end{array}}{D}\,\text{S} \quad \xRightarrow[\text{S-red}_L]{\text{reduces to}} \quad \begin{array}{c}[C]\\ \mathcal{D}_4\\ C \to A\\ \mathcal{D}_2\\ D\end{array}$$

where $\begin{array}{c}[C]\\ \mathcal{D}_4\\ C \to A\end{array}$ is constructed from $\begin{array}{c}C\\ \mathcal{D}_1\\ A\end{array}$ via application of $\to$I.

$$\cfrac{\cfrac{\begin{array}{c}[C]\\ \mathcal{D}_1\\ B\end{array}}{A \vee B}\,\vee\text{I}_R \qquad \begin{array}{c}[C \to A]\\ \mathcal{D}_2\\ D\end{array} \qquad \begin{array}{c}[C \to B]\\ \mathcal{D}_3\\ D\end{array}}{D}\,\text{S} \quad \xRightarrow[\text{S-red}_R]{\text{reduces to}} \quad \begin{array}{c}[C]\\ \mathcal{D}_4\\ C \to B\\ \mathcal{D}_3\\ D\end{array}$$

[16]I thank Antonio Piccolomini d'Aragona for suggesting this variant of the reduction rule. Note also that we agree with Schroeder-Heister (2006) (p. 553) that "[i]n principle, reductions should be definable for derivation structures ending with any non-introduction inference."

where $\begin{array}{c}[C]\\ \mathcal{D}_4\\ C \to B\end{array}$ is constructed from $\begin{array}{c}C\\ \mathcal{D}_1\\ B\end{array}$ via application of $\to$I.

With the reductions S-red$_L$ and S-red$_R$ the justification of the Split rule is achieved (we will inspect these reductions, more specifically, their typed variants in the form of computation rules more in the next section).

4 Formulas as types: Typing the S rule

Let us return to our original task, i.e., investigating the computational content of the Split rule in the style of BHK semantics and guided by the formulas-as-types principle while assuming a propositional fragment of Martin-Löf's constructive type theory on the background.

As mentioned, we will regard the Split rule as a disjunction elimination-like rule that can be generalized into the S rule. Thus, let us start by considering the standard typed variant of the disjunction elimination rule before we try to model the selector for S based on it.

The typed introduction and elimination rules specifying the constructors and the selector for disjunction are the following:

$$\frac{a : A}{\mathsf{inl}(a) : A \vee B}\ \vee\mathrm{I}_L \qquad \frac{b : B}{\mathsf{inr}(b) : A \vee B}\ \vee\mathrm{I}_L$$

$$\frac{c : A \vee B \qquad \begin{array}{c}[x : A]\\ \vdots\\ d(x) : D\end{array} \qquad \begin{array}{c}[y : B]\\ \vdots\\ e(y) : D\end{array}}{\mathsf{D}(c, x.d, y.e) : D}\ \vee\mathrm{E}$$

The constructors inl and inr for the type $A \vee B$ are called injections and they tell us from which disjunct was the disjunction constructed. The selector D is a function that takes three arguments (an arbitrary proof of $A \vee B$, a function $d(x)$ that transforms an arbitrary proof of A into a proof of D, and a function $e(y)$ that transforms an arbitrary proof of B into a proof of D) and returns a proof of D as a value. The notation '$x.d$' means that the variable x becomes bound in $d(x)$ via D, analogously for '$y.e$'.

Note that the selector D operates essentially as a pattern-matching program that incorporates the method of proof by cases by its ability to generate subproofs: it checks whether $A \vee B$ was derived from A or from B (i.e., whether the forms of its canonical proofs are $\mathsf{inl}(a)$ or $\mathsf{inr}(b)$): if from A, then we should continue by computing the subprogram $d(x)$, if it was derived from B, then we should continue by computing the subprogram $e(y)$. The corresponding computation rules for these two cases, which show us how the selector D operates on the canonical proof objects generated by the constructors of the type $A \vee B$, are as follows:

$$\dfrac{a : A \qquad \begin{matrix}[x : A]\\ \vdots \\ d(x) : D\end{matrix} \qquad \begin{matrix}[y : B]\\ \vdots \\ e(y) : D\end{matrix}}{\mathsf{D}(\mathsf{inl}(a), x.d, y.e) = d(a) : D}\,\vee\mathrm{C}_L \qquad \dfrac{b : B \qquad \begin{matrix}[x : A]\\ \vdots \\ d(x) : D\end{matrix} \qquad \begin{matrix}[y : B]\\ \vdots \\ e(y) : D\end{matrix}}{\mathsf{D}(\mathsf{inr}(b), x.d, y.e) = e(b) : D}\,\vee\mathrm{C}_L$$

where $d(a)$ is the result of substituting a for x in d, analogously for $e(b)$.

Now, let us return to the S rule. If we want to produce a typed variant of the S rule in the style of the typed disjunction elimination rule, we would need to find a program (a three-argument function), let us call it S, that would incorporate the method of proof by cases and could bind variables (i.e., it could discharge assumptions).

Immediately, we can see that the selector D seems as a good basis for the typed S rule. It appears to fit all the requirements: it takes three arguments, it can discharge assumptions, but most importantly, the core mechanism of the selector D is subproof generation, specifically, case analysis, not just substitution. And, as it turns out, that is exactly what we need to express the computational content of the S rule, and thus effectively also of the Split rule.

Let us produce the typed variant of the S rule taking all these considerations into account, which will give us the new selector S:[17]

$$\dfrac{\begin{matrix}[z : C]\\ \vdots \\ c(z) : A \vee B\end{matrix} \qquad \begin{matrix}[x : C \to A]\\ \vdots \\ d(x) : D\end{matrix} \qquad \begin{matrix}[y : C \to B]\\ \vdots \\ e(y) : D\end{matrix}}{\mathsf{S}(z.c, x.d, y.e) : D}\,\mathrm{S}$$

where C is restricted to Harrop formulas. Note that this rule differs from the typed disjunction elimination rule in three key aspects: the first premise is a hypothetical judgment:

$$\begin{matrix}z : C\\ \vdots \\ c(z) : A \vee B\end{matrix}$$

i.e., $z : C \vdash c(z) : A \vee B$ in linear notation, the assumptions of the subproofs take the form of an implication (i.e., they are not composed of subformulas of the original disjunction), and the formula C has to be a Harrop formula.

Now, before we can get to the explanation of how to compute the selector S, we first need to make a short detour and explain the form of the major premise of the S rule, i.e., the hypothetical judgment $z : C \vdash c(z) : A \vee B$. In Martin-Löf's constructive type theory, the standard meaning of the hypothetical judgment of the general form (where B is a non-dependent type):

$$x : A \vdash b(x) : B$$

[17] We will refrain from calling the selector S "split" as in the literature on type theory a selector named split already appears but it is used as the selector for conjunction, or more precisely, for the cartesian product of two types (see, e.g., Nordström et al. (1990), p. 73).

is that $b(a)$ is a proof object for B assuming we have a proof object a for A. In other words, its meaning is explained via reducing it to the corresponding categorical judgment:

$$b(a) : B$$

And the meaning explanation of this judgment is that $b(a)$ is a program that upon computation yields a canonical proof object of the type B. Note that $b(a) : B$ is obtained by substituting the closed proof object a of the type A for the free variable x in $b(x)$ and B.[18] To put it differently, in order to be able to compute the open proof object $b(x)$ depending on $x : A$ to obtain the canonical proof object of the type B, we first need to replace the free variable x with the appropriate closed proof object a.

Now, let us return to the S rule. Observe that the major premise of the S rule is a more specific hypothetical judgment than the one presented above as its assumption is restricted to Harrop formulas only. This fact, together with Smith (1993)'s results showing us that we can consider open proof objects computable to a canonical form as long as they range over Harrop formulas,[19] allows us to introduce a specialized variant of the hypothetical judgment of the form:

$$z : C \vdash b(z) : B$$

where C is restricted to Harrop formulas with the following modified meaning explanation: $b(z)$ is a program that upon computation yields a canonical proof object of the type B.[20] In other words, this hypothetical judgment behaves essentially as a categorical one since $z : C$ is a computationally irrelevant assumption and as such it is not needed for the evaluation of $b(z) : B$.[21] Also, note that this meaning explanation mirrors the meaning explanation of the corresponding categorical judgment, so it does not break the general idea of explaining hypothetical judgments via categorical ones.

With this specialized hypothetical judgment, we can now explain the computational interpretation of the S selector. So, how do we compute this new program (i.e., noncanonical proof object) of the form $\mathsf{S}(c, d, e)$? We begin by computing c to the canonical form. First, note that c is actually an open term $c(z)$, i.e., it depends on the variable z. Computing a program with a hole might seem odd at first, however, we have to keep in mind that it is not just any hole: the variable z is of type C which is restricted to Harrop formulas only. This makes it an instance of the special kind of hypothetical judgment we have just introduced above and, consequently, it means that $c(z)$ can be computed to a canonical form as is. If the value of $c(z)$ is of the form $\mathsf{inl}(a(z))$ with $a : A$ and $z : C$, then lambda abstract over the variable z to obtain $\lambda z.a(z)$ (of type $C \to A$) and continue by computing $d(\lambda z.a(z))$ of type D. If the value of $c(z)$ is

[18] Furthermore, we also need to know that $b(x)$ is extensional in the sense that if $a = a' : A$, then $b(a) = b(a') : A$. See Martin-Löf (1984).

[19] For details, see Appendix A.

[20] I thank Ansten Klev for this suggestion.

[21] It seems to correspond to proof irrelevant assumptions of the form $x \div A$ introduced in Pfenning (2001), however, for the sake of space, we will not explore this connection further here.

of the form $\mathsf{inr}(b(z))$ with $b : B$ and $z : C$, then lambda abstract over the variable z to obtain $\lambda z.b(z)$ (of type $C \to B$) and continue by computing $e(\lambda z.b(z))$ of type D.

This explanation of S is expressed by the following computation rules:

$$\dfrac{\begin{matrix}[z : C] \\ \vdots \\ a(z) : A\end{matrix} \quad \begin{matrix}[x : C \to A] \\ \vdots \\ d(x) : D\end{matrix} \quad \begin{matrix}[y : C \to B] \\ \vdots \\ e(y) : D\end{matrix}}{\mathsf{S}(z.\mathsf{inl}(a(z)), x.d, y.e) = d(\lambda z.a(z)) : D} \qquad \dfrac{\begin{matrix}[z : C] \\ \vdots \\ b(z) : B\end{matrix} \quad \begin{matrix}[x : C \to A] \\ \vdots \\ d(x) : D\end{matrix} \quad \begin{matrix}[y : C \to B] \\ \vdots \\ e(y) : D\end{matrix}}{\mathsf{S}(z.\mathsf{inr}(z.b(z)), x.d, y.e) = e(\lambda z.b(z)) : D}$$

We can observe that the selector S behaves analogously to the selector D: the main difference is that the function $d(x)$ requires an argument of type $C \to A$, not A as with D, so instead of substituting a for x in $d(x)$ we substitute $\lambda z.a(z)$ for x in $d(x)$. Also, note that the selector S binds the variable z in $a(z)$. Analogously for the function $e(y)$.

Note. We can regard the typed S rule as a generalized version of the typed disjunction elimination rule. In other words, we can view the selector D as a special case of the selector S: by choosing $c(z)$ to be $\mathsf{inl}(a)$ or $\mathsf{inr}(b)$ we are making the derivation of $A \vee B$ independent of the assumption $z : C$, which is then correspondingly reflected in the assumptions of the subproofs. Thus, we get A and B instead of $C \to A$ and $C \to B$ since $A \vee B$ no longer depends on C. The computation and expansion rules will be changed accordingly.

Since we have found a function that transforms the premises of the rule S into its conclusion, we can say that the rule is constructively valid in the sense of BHK semantics.[22] But what about the original Split rule itself? Is it also constructively valid? First, note that we cannot simply replace the black box placeholder selector s discussed at the beginning:

$$\frac{f : C \to (A \vee B)}{\mathsf{s}(f) : (C \to A) \vee (C \to B)}$$

where C is a Harrop formula, with the selector S:

$$\frac{f : C \to (A \vee B)}{\mathsf{S}(f) : (C \to A) \vee (C \to B)}$$

and expect it to work. The reason for that is that S takes different arguments than the above derivation provides, namely S is a function that takes three arguments (a function c that transforms an arbitrary proof of C into a proof of $A \vee B$, a function d that transforms an arbitrary proof of $C \to A$ into a proof of D, and a function e that transforms an arbitrary proof of $C \to B$ into a proof of

[22] Recall that by a constructively valid rule in the sense of BHK semantics we mean a rule for which we can find an effective function that transforms arbitrary proofs of the premises into proofs of the conclusion. Also, we should not conflate the notions of constructive validity, schematic validity (Piccolomini d'Aragona (2024)), and proof-theoretic validity (see Schroeder-Heister (2006)), which in turn should not be conflated with admissibility. These notions are no doubt related but distinct (see also de Campos Sanz et al. (2014)).

D) and returns a proof of D as a value. On the other hand, s would have to be a function that requires a single argument (an arbitrary proof f of $C \to (A \vee B)$) and returns a proof of $(C \to A) \vee (C \to B)$ as a value.

However, since the Split rule and the S rule are interderivable (see Section 3.2), we can justify its constructive validity in an indirect way.[23] We simply apply the Split-red reduction rule (now typed) to a derivation ending with the Split rule (assuming $\mathcal{D}$ is a closed valid derivation):

$$\dfrac{\overset{\mathcal{D}}{f : C \to (A \vee B)}}{\mathsf{s}(f) : (C \to A) \vee (C \to B)}\,\text{Split} \quad \xRightarrow[\text{Split-red}]{\text{reduces to}}$$

$$\dfrac{\dfrac{\overset{\mathcal{D}}{f : C \to (A \vee B)} \qquad [z : C]^1}{\mathsf{ap}(f, z) : A \vee B}\,{\to}\text{E} \qquad \dfrac{[x : C \to A]^2}{\mathsf{inl}(x) : (C \to A) \vee (C \to B)}\,\vee\text{I}_L \qquad \dfrac{[y : C \to B]^3}{\mathsf{inr}(y) : (C \to A) \vee (C \to B)}\,\vee\text{I}_R}{\mathsf{S}(z.\mathsf{ap}(f, z), x.\mathsf{inl}(x), y.\mathsf{inr}(y)) : (C \to A) \vee (C \to B)}\,\text{S}_{1,2,3}$$

Now, since we know that the S rule is valid, we can claim that this derivation is also valid (assuming $\mathcal{D}$ is valid), and use this piece of knowledge to further justify the claim that the Split rule is valid as well.[24] Furthermore, it can be shown that if we add the S selector, i.e., the typed S rule, to a propositional fragment of Martin-Löf's constructive type theory, it retains normalization (see Appendix B).

Note. Could perhaps a simpler justification be found for the Split rule? First, let us consider the following (untyped) derivation containing an application of the Split rule:

$$\dfrac{\dfrac{\dfrac{\overset{[C]^1}{\overset{\mathcal{D}'}{A}}}{A \vee B}\,\vee\text{I}_L}{C \to A \vee B}\,{\to}\text{I}_1}{(C \to A) \vee (C \to B)}\,\text{Split}$$

An analogous derivation can be provided for the right disjunct B but we will now focus only on the left disjunct A. Now, consider the following reduction procedure (compare with Split-red):

$$\dfrac{\dfrac{\dfrac{\overset{[C]^1}{\overset{\mathcal{D}'}{A}}}{A \vee B}\,\vee\text{I}_L}{C \to A \vee B}\,{\to}\text{I}_1}{(C \to A) \vee (C \to B)}\,\text{Split} \quad \xRightarrow[\text{Split-red2}]{\text{reduces to}} \quad \dfrac{\dfrac{\overset{[C]^1}{\overset{\mathcal{D}'}{A}}}{C \to A}\,{\to}\text{I}_1}{(C \to A) \vee (C \to B)}\,\vee\text{I}_L$$

[23] I thank Antonio Piccolomini d'Aragona for this observation for raising the issue discussed in the following note.

[24] Interestingly, Plisko (2009) shows that the Harrop rule (and thus also the Split rule) is not valid in realizability semantics (Kleene (1945); I thank one of the reviewers for bringing this to my attention). Considering that realizability semantics and BHK semantics are often taken to be more or less corresponding to each other, this constitutes an interesting topic for further research. However, it is beyond the scope of the present paper.

Can this be regarded as a simpler, more basic justification of the Split rule since it makes no use of the S rule and only relies on ∨I and →I rules? We believe so, however, with one important caveat – it rather shows that Split is proof-theoretically valid, not necessarily constructively valid: the above reduction procedure still gives us no indication as to how the corresponding selector function that transforms proofs of the premise of the Split rule into proofs of the conclusion should look like. This becomes more clear when we consider the typed variants:

$$\dfrac{\dfrac{\dfrac{\begin{matrix}[z : C]^1\\ \mathcal{D}'\\ a(z) : A\end{matrix}}{\mathsf{inl}(a(z)) : A \vee B}\,\vee\mathrm{I}_L}{\lambda z.\mathsf{inl}(a(z)) : C \to A \vee B}\,\to\mathrm{I}_1}{\mathsf{s}(\lambda z.\mathsf{inl}(a(z))) : (C \to A) \vee (C \to B)}\,\mathrm{Split} \quad \xRightarrow[\text{Split-red2}]{\text{reduces to}} \quad \dfrac{\dfrac{\begin{matrix}[z : C]^1\\ \mathcal{D}'\\ a(z) : A\end{matrix}}{\lambda z.a(z) : C \to A}\,\to\mathrm{I}_1}{\mathsf{inl}(\lambda z.a(z)) : (C \to A) \vee (C \to B)}\,\vee\mathrm{I}_L$$

The question that still remains open is how the function s should be computed. As mentioned above, we cannot simply replace it with S. And supplying some straightforward computational rule for s such as $\mathsf{s}(\lambda z.\mathsf{inl}(a(z))) = \mathsf{inl}(\lambda z.a(z)) : (C \to A) \vee (C \to B)$ would also not suffice as we would still need to take care of the other disjunct as well. And if we decide to merge everything into a single selector (one that combines the mechanisms of withdrawing assumptions, making substitutions, and analyzing subcases), we would ultimately arrive back at the selector S or some variant of it.

5 Conclusion

We have presented a general version of the generalized Kreisel-Putnam rule, also known as the Split rule, called the S rule, and shown that it is constructively valid in the sense of BHK semantics. Specifically, we have found an effective function that transforms arbitrary proofs of the premises into proofs of the conclusion. We have called this function the selector S and it can be used to indirectly justify the Split rule itself. The most tricky part of the typed S rule/selector S lies in the fact that it requires an evaluation of an open proof object to a canonical form. This issue can be, however, overcome once we realize that the free variables of the open proof object range only over Harrop formulas which are computationally irrelevant. Furthermore, we have checked that if we add this rule into a propositional fragment of constructive type theory, it retains normalization.

In terms of future work, there are several directions to consider: i) investigating alternative variants of the generalized Split rule, including a higher-level natural deduction variant and a dependent type variant, ii) inspecting the possibility of treating the generalized Split rule as an implication elimination-like rule instead of a disjunction elimination-like rule, iii) examing the generalized Split rule from perspectives of other semantics than BHK (e.g., realizability semantics), and finally iv) exploring the Split rule in a first-order setting, including its existential analog:

$$\frac{C \to \exists x A}{\exists x(C \to A)} \; \exists \text{ Split}$$

where C is a first-order Harrop formula, i.e., a formula that does not contain $\vee$ or $\exists$ except in the antecedents of implications, and x is not free in C.

Concerning related work, Condoluci and Manighetti (2018) proposed a typed rule for the Harrop rule that follows the same general pattern as our typed Split rule (i.e., it is based on the typed disjunction elimination rule). Interestingly, they arrived at this pattern differently: while our approach is bottom-up (we have started by studying the inferential behavior of the Split rule and then generalized it), their approach was top-down (they have started with Visser rules (Roziére (1993), Iemhoff (2005)) as a basis for all admissible rules and considered the Harrop rule as a special case). Furthermore, although they also relied on the propositions-as-types principle in their investigation, their goal was different from ours. They were interested in examining the notion of admissibility, while we are interested in the proof-theoretic/computational meaning and constructive validity of the Split rule itself.

A Appendix: Open proof objects computable to canonical forms

Most of the work necessary to show that we can compute open proof objects ranging over Harrop formulas to canonical values has been already done and can be divided into two main observations:

> Observation 1. Harrop formulas have no computational/constructive content due to the fact that disjunction can never appear as the main connective.[25] This is a well-known fact commonly used in proof extraction research to simplify extracted programs (Goad (1980), Sasaki (1986), Berger et al. (2006), Schwichtenberg and Wainer (2012)).
>
> Observation 2. Smith (1993) translated Kleene-Aczel's slash computability relation (Kleene (1962), Aczel (1968)) to a type-theoretic setting, specifically to Martin-Löf's constructive type theory, and showed that it can be used to formulate conditions for proof objects with free variables ranging over Harrop formulas to be computable to canonical forms, i.e., values.

First, however, we need to discuss how Harrop formulas and the slash computability relation are translated into a type-theoretic setting. The notion of Harrop formula introduced in 2.1 can be easily carried over with one exception:

[25] In this paper, we are solely interested in the variant of the Split rule where C is a Harrop formula, however, it might be interesting to consider also different variants. For example, a variant where C is an almost negative formula (Troelstra (1973)), or a normal formula (Nepeivoda (1978), Nepeivoda (1982)), or a singleton formula (Sasaki (1986)), or a rank 0 formula (Hayashi and Nakano (1988)) that generalize them all.

we cannot incorporate $\bot$ into the definition of Harrop formulas if we assume $\bot$ is the formula that has no proof. If we did, the key Theorem 1 (see below) would no longer hold. Specifically, we could not be able to recursively construct a proof object for $\bot$ since, simply put, there is none (see also Smith (1993)). In other words, the notion of absurdity $\bot$ in our system behaves computationally differently than Kleene-Aczel's absurdity $\bot_S$ (defined as $0 = 1$) assumed in the original definition of slash computability relation. While in our system there is no proof object t of type $\bot$ that could be computable, in Kleene-Aczel's system a proof object t of type $\bot_S$ is computable whenever $t : \bot_S$ is derivable, i.e., whenever we have inconsistent premises.

However, given these considerations, a workaround can be devised: let us introduce an alternative specification of absurdity, denoted $\bot_A$, that can be proven only in inconsistent contexts, thus mimicking $\bot_S$. In contrast to $\bot$, $\bot_A$ has no associated rules, specifically, no elimination rule, i.e., no ex falso quodlibet. From this perspective, it might be reminiscent of absurdity from minimal logic, however, the effect of the missing elimination rule is achieved by adopting axioms of the form $\bot_A \to p$ for atomic formulas p. Then, any formula can be derived from $\bot_A$ via these axioms and introduction rules, so the logical strength of the system remains unchanged,[26] only a computational interpretation of absurdity is modified. And it is this change that allows us to include absurdity in the definition of Harrop formulas (so the clause (a) from Section 2.1 would read "any atomic formula and $\bot_A$ is a Harrop formula") and its computational behavior will mirror the behavior of $\bot_S$ (see clause 2 in the definition below). Now, let us proceed to define the slash computability relation.

For a propositional fragment of CTT, the slash computability relation $\Gamma \mid t : A$, read as "Γ slashes $t : A$", can be inductively defined by the following clauses (see Smith (1993) for a definition of slash relation for full CTT):

1. $\Gamma \mid t : X_i$ if $\Gamma \vdash t : X_i$, $i = 1, 2, \dots$ (X_i is an atomic formula),
2. $\Gamma \mid t : \bot_A$ if $\Gamma \vdash t : \bot_A$.[27]
3. $\Gamma \mid t : A \land B$ if $\Gamma \vdash t : A \land B$ and $\Gamma \vdash t = (a, b) : A \land B$ and there exist proof objects a and b such that $\Gamma \mid a : A$ and $\Gamma \mid b : B$,
4. $\Gamma \mid t : A \lor B$ if $\Gamma \vdash t : A \lor B$ and $\Gamma \vdash t = \mathsf{inl}(a) : A \lor B$ for some proof object a such that $\Gamma \mid a : A$ or $\Gamma \vdash t = \mathsf{inr}(b) : A \lor B$ for some proof object b such that $\Gamma \mid b : B$,
5. $\Gamma \mid t : A \to B$ if $\Gamma \vdash t : A \to B$ and $\Gamma \vdash t = \lambda x.b(x) : A \to B$ and there exists a proof object $b(x)$ such that $\Gamma, x : A \vdash b(x) : B$ and for all proof objects a, $\Gamma \mid a : A$ implies $\Gamma \mid b(a) : B$.

[26] This method is inspired by Goad (1980).

[27] Smith (1993) uses here the absurdity $\bot_M$ of minimal logic which differes from $\bot$ in that it does not have elimination rule. Our $\bot_A$ also does not have elimination rule, however, it does not lead to minimal logic since we adopt the axioms of the form $\bot_A \to p$. Note that if we were to adopt $\bot$ instead of $\bot_A$, the clause 2 would read "$\Gamma \mid t : \bot$ does not hold for any proof object t".

Informally, $\Gamma \mid t : A$ can be read as "a proof object t of type A is computable in context Γ".

Now, the first important result for showing how to compute open proof objects ranging over Harrop formulas to canonical values is the following theorem:

Theorem 1 (Theorem 3 in Smith (1993), p. 195). If C is a Harrop formula in the context Γ then there exists a proof object $h(z)$ such that $\Gamma, z : C \mid h(z) : C$. The proof object $h(z)$ can be recursively constructed as follows:[28]

(i) $\Gamma, x : X_i \mid x : X_1, i = 1, 2 \ldots,$

(ii) $\Gamma, x : \bot_A \mid x : \bot_A,$

(iii) if $\Gamma, x : A \mid a(x) : A$ and $\Gamma, x : A, y : B \mid b(x, y) : B$ then $\Gamma, z : A \wedge B \mid (a(\mathsf{fst}(z)), b(\mathsf{fst}(z), \mathsf{snd}(z))) : A \wedge B,$[29]

(iv) if $\Gamma, x : A, y : B \mid b(x, y) : B$ then $\Gamma, z : A \to B \mid \lambda x.b(x, \mathsf{ap}(z, x)) : A \to B.$

This theorem establishes that we can construct a proof object $h(z)$ for a Harrop formula C computable to a canonical form, i.e., value, by only using the assumption $z : C$. From this perspective, we can say that a Harrop formula C is "self-constructable". In other words, we can essentially trivially prove C from "itself", i.e., by using only the assumption $z : C$.

It is this fact that makes it possible to compute proof objects with "Harrop-shaped holes" in them, which brings us to the other crucial result (again due to Smith (1993)) needed for the justification of the typed S rule: specifically, that we can have open proof objects, i.e., proof terms with free variables ranging over the Harrop formulas C, computable to a canonical form. This is established by the following result:

Corollary 1 (Smith (1993), p. 196). Let C be a Harrop formula and let $h(z)$ be constructed according to the corresponding clauses in Theorem 1. If $z : C \vdash b(z) : B$ then there exists a proof object in a canonical form, i.e., a value $v(z)$ of the formula B such that $z : C \vdash b(h(z)) = v(z) : B$.

Note. Observe that we do not need to consider the computational interpretation of open proof objects in general, just of those open proof objects whose free variables range over Harrop formulas and thus are computationally irrelevant, i.e., they have no computational content.[30] Consequently, the effect of the adoption of this kind of open proof objects on notions such as canonical proofs or computation rules is limited although not insignificant. For example, the modified kind of hypothetical judgments restricted to Harrop assumptions, such as those in Corollary 1, requires the inclusion of specific forms of η-expansion (see Smith (1993)).

[28] We present a simplified propositional version of the clauses.

[29] The functions fst and snd are the selectors for conjunction corresponding to the left and right elimination rules.

[30] See the literature on proof extraction, e.g., Berger et al. (2006).

Now, let us see how we can complete the justification of the typed S rule. First, we have observed that Harrop formulas have at most one constructor (= have no computational content, Observation 1). Then, we have observed that there is Smith's type-theoretic translation of Kleene-Aczel's slash computability relation that can be used to specify conditions for proof objects with free variables ranging over Harrop formulas to be computable to a canonical form (Observation 2, using Observation 1). And it is this fact that finally allows us to compute the major premise $c(z) : A \vee B$ of the S rule. More specifically, utilizing the Corollary 1 (Smith (1993)), we know that if:

$$\begin{array}{c} z : C \\ \vdots \\ c(z) : A \vee B \end{array}$$

then there exists a canonical value $v(z)$ of the type $A \vee B$ such that:

$$\begin{array}{c} z : C \\ \vdots \\ c(h(z)) = v(z) : A \vee B. \end{array}$$

And we know that $v(z)$ of $A \vee B$ has to be either $\mathsf{inl}(a(z))$ or $\mathsf{inr}(b(z))$, which is what we needed to establish for the computation rules for the selector S (introduced in Section 4) to work as intended.

B Appendix: Normalization

Smith (1993) proved normalization for Martin-Löf's constructive type theory (CTT, Martin-Löf (1984), Nordström et al. (1990)) using a type-theoretic translation of Kleene-Aczel's slash computability relation (Kleene (1962), Aczel (1968)). Specifically, he showed that if $a : A$ can be derived (within the empty context), then a can be computed to a canonical form, i.e., a value of the type A. The overall structure of the proof follows closely the standard structure of normalization proofs for typed terms using Tait's reducibility/computability method (Tait (1967)). In fact, as Smith himself notes, Tait's computability predicate $Comp_A(a)$ can be seen as a special case of the slash computability relation with the empty context, i.e., $\mid a : A$.

Once again, we assume a propositional fragment of CTT with no dependent types containing conjunction, disjunction, implication, and absurdity (no propositional identity type). The only difference is the alternative formulation of the "elimination rule" for absurdity (discussed in Appendix A). All the general rules, including the substitution rules and judgemental identity rules, are as defined in Nordström et al. (1990) for intensional theory with decidable type checking. The judgemental identity $a = b : A$ is understood as definitional identity (Martin-Löf (1984)). As usual, introduction rules for logical constants describe what are the canonical forms, i.e., values. Computation rules are terminating with a value v as soon as the outermost form of v is a canonical form (= lazy evaluation).

We will not reproduce here, however, the whole normalization proof, we only show that its key ingredient Theorem 2:

Theorem 2. Let Δ be a context $x_1 : T_1, \ldots, x_m : T_m$ and $t_1, \ldots, t_m$ proof objects such that $\Gamma \mid t_i : T_i, 0 < i \leq m$. Then $\Delta \vdash a(\vec{x}) : A$ implies $\Gamma \mid a(\vec{t}) : A$.

where $\vec{x} = x_1, \ldots, x_m$ and $\vec{t} = t_1, \ldots, t_m$, holds in a propositional fragment of CTT extended with the typed S rule.

Theorem 2 together with Corollary 2:

Corollary 2. If $\Gamma \mid a : A$, then there exists a canonical proof object, i.e., a value v of type A, such that $\Gamma \vdash a = v : A$.

which follows from the definition of slash computability relation $\Gamma \mid t : A$, then give the normalization result (for details, see Smith (1993)).

The type-theoretic version of the S rule for a propositional fragment of CTT is as follows (where C is a Harrop formula):

$$\frac{\begin{array}{rcl} \Delta, z : C & \vdash & c(\vec{x}, z) : A \vee B \\ \Delta, x : C \to A & \vdash & d(\vec{x}, x) : D \\ \Delta, y : C \to A & \vdash & e(\vec{x}, y) : D \end{array}}{\Delta \ \vdash \ \mathsf{S}(z.c(\vec{x}, z), x.d(\vec{x}, x), y.e(\vec{x}, y)) : D}$$

Computation rules:

- split left: $\mathsf{S}(z.\mathsf{inl}(a(z)), x.d, y.e) = d(\lambda z.a(z)) : D$
- split right: $\mathsf{S}(z.\mathsf{inr}(b(z)), x.d, y.e) = e(\lambda z.b(z)) : D$

Proof of Theorem 2. By induction on the structure of the derivation $\Delta \vdash a : A$ (together with Lemmas 1 and 2, see below). As mentioned above, we show only the case for the new typed rule S, i.e., the selector S. The presentation of the proof itself follows Smith (1993)'s treatment of the elimination rule for disjoint union/disjunction to make the differences between selectors S and D more apparent.

By induction hypothesis we obtain:

(1) $\Gamma \mid c(\vec{t}, z) : A \vee B$ for all z such that $\Gamma \mid z : C$

(2) $\Gamma \mid d(\vec{t}, \lambda z.a(z)) : D$ for all $\lambda z.a(z)$ such that $\Gamma \mid \lambda z.a(z) : C \to A$

(3) $\Gamma \mid e(\vec{t}, \lambda z.b(z)) : D$ for all $\lambda z.b(z)$ such that $\Gamma \mid \lambda z.b(z) : C \to B$

From (1) we get via the definition of slash computability relation $\mid$ either:

(4) $\Gamma \vdash c(\vec{t}, z) = \mathsf{inl}(a(z)) : A \vee B$ for some term $a(z)$ such that $\Gamma, z : C \mid a(z) : A$. $\Gamma, z : C \mid a(z) : A$ implies $\Gamma, z : C \vdash a(z) : A$. Furthermore, since C is a Harrop formula we can obtain, via Corollary 4 (Smith (1993)), that $\Gamma, z : C \vdash a(c(z)) = v(z) : A$, i.e., that $a(c(z))$ of type A can be computed to a canonical value even if it contains free variables, as long as those variables range over Harrop formulas. The proof object $c(z) : C$ is recursively constructed from the assumption $z : C$ as shown in Smith (1993), Theorem 3.

(5) Analogously to (4).

Let us assume that (4) holds and continue by case analysis. By definition of S, we get:

(6) $\Gamma \vdash \mathsf{S}(z.\mathsf{inl}(a(z))), x.d(\vec{t}, x), y.e(\vec{t}, y) = d(\vec{t}, \lambda z.a(z)) : D$

Before we can put together (2) and (4) to obtain (7), we need to check that $\Gamma \mid \lambda z.a(z) : C \to A$. We proceed accordingly to the definition of slash $\mid$ for $\to$ type (see above). First, (i) and (ii) are fulfilled since from (4), we have $\Gamma, z : C \vdash a(z) : A$ and from that, via $\to$I, we get $\Gamma \vdash \lambda z.a(z) : C \to A$. As for (iii), there is a proof object $a(z)$ such that $\Gamma, z : C \vdash a(z) : A$, we just need to show that for all proof object c, $\Gamma \mid c : C$ implies $\Gamma \mid \mathsf{ap}(a, c) : A$.

Let us begin by assuming $\Gamma \mid c : C$. From this it follows via Corollary 1 (Smith (1993)) that $\Gamma \vdash c = v : C$. Thus, we also get $\Gamma \mid v : C$ (by Lemma 1). Now, we want to show that $\Gamma \mid \mathsf{ap}(a, c)$, i.e., $\Gamma \mid \mathsf{ap}(\lambda z.a(z), c)$. By induction hypothesis instantiated with the help of $\Gamma \mid v : C$, we get $\Gamma \mid a(v) : A$. And since we know that $\mathsf{ap}(\lambda z.a(z), c) = a(c)$, we also get that $\mathsf{ap}(\lambda z.a(z), c) = a(v)$. And from that, we can finally obtain (via Lemma 1) $\Gamma \mid \mathsf{ap}(\lambda z.a(z), c)$ which is what we wanted to show.

(7) $\Gamma \mid d(\vec{t}, \lambda z.\mathsf{inl}(a(z))) : D$

From (6) and (7) we obtain via Lemma 1:

(8) $\Gamma \mid \mathsf{S}(z.\mathsf{inl}(a(z))), x.d(\vec{t}, x), y.e(\vec{t}, y) : D$

And from (4) and (8) and Lemma 2 we get the required:

(8) $\Gamma \mid \mathsf{S}(z.c(\vec{t}, z), x.d(\vec{t}, x), y.e(\vec{t}, y)) : D$.

Lemma 1 (Smith (1993)). Let $\Gamma \mid a : A$ and $\Gamma \vdash b : A$, then $\Gamma \vdash a = b : A$ implies $\Gamma \mid b : A$.

Proof. Follows from the definition of slash $\mid$. We demonstrate the case for disjunction $A \vee B$.

Let us assume $\Gamma \mid c : A \vee B$, $\Gamma \vdash c' : A \vee B$, and $\Gamma \vdash c = c' : A \vee B$. We want to show that $\Gamma \mid c' : A \vee B$. By the definition of slash relation for disjunction, we need to show that $\Gamma \vdash c' : A \vee B$ (by assumption) and we need to show that $\Gamma \vdash c' = \mathsf{inl}(a')$ for some a' such that $\Gamma \mid a' : A$ or that $\Gamma \vdash c' = \mathsf{inr}(b')$ for some b' such that $\Gamma \mid b' : B$. Let us assume the first case.

Now, since we have $\Gamma \mid c : A \vee B$ (by assumption), this means we have either $\Gamma \mid a : A$ or $\Gamma \mid b : B$ (by the definition of slash relation for disjunction), i.e., that $\Gamma \vdash c = \mathsf{inl}(a) : A \vee B$ or $\Gamma \vdash c = \mathsf{inr}(b) : A \vee B$. Let us again assume the first case. Since we have assumed that $c = c'$, this means that $\mathsf{inl}(a) = \mathsf{inl}(a')$, and thus we have found $c' = \mathsf{inl}(a)$ for a such that $\Gamma \mid a : A \vee B$ and we can claim $\Gamma \mid c' : A \vee B$.

The second case proceeds analogously.

Lemma 2 (Smith (1993)). Let $\Gamma \mid a : A$ and let B be a type in the context Γ, then $\Gamma \vdash A = B$ implies $\Gamma \mid a : B$.

Proof. By induction on the structure of the type A. We demonstrate the case for disjunction $A \vee B$.

Let us assume $\Gamma \mid c : A \vee B$, $\Gamma \vdash C \vee D$ *type*, and $\Gamma \vdash A \vee B = C \vee D$, and show that $\Gamma \mid c : C \vee D$. First, $\Gamma \mid c : A \vee B$ implies $\Gamma \vdash c : A \vee B$ and from that and $\Gamma \vdash A \vee B = C \vee D$ we can conclude that $\Gamma \vdash c : C \vee D$. Now, we just need to show that $\Gamma \vdash c = \mathsf{inl}(a) : C \vee D$ for some term a such that $\Gamma \mid a : C$ or that $\Gamma \vdash c = \mathsf{inr}(b) : C \vee D$ for some term b such that $\Gamma \mid b : D$. Let us consider only the first case, as the second one proceeds analogously. From $\Gamma \mid c : A \vee B$ (by assumption) we can get (via the definition of slash relation) that either $\Gamma \vdash c = \mathsf{inl}(a) : A \vee B$ for some term a such that $\Gamma \mid a : A$ or that $\Gamma \vdash c = \mathsf{inr}(b) : A \vee B$ for some term b such that $\Gamma \mid b : B$. Let us consider the first case. From $\Gamma \vdash c = \mathsf{inl}(a) : A \vee B$ and $\Gamma \vdash A \vee B = C \vee D$, we can conclude that $\Gamma \vdash c = \mathsf{inl}(a) : C \vee D$ for some a such that $\Gamma \mid a : C$, and thus get $\Gamma \mid c : C \vee D$, which is what we wanted.

Funding. This paper is an outcome of the project Logical Structure of Information Channels, no. 21-23610M, supported by the Czech Science Foundation and realized at the Institute of Philosophy of the Czech Academy of Sciences.

Acknowledgments. I would like to thank Antonio Piccolomini d'Aragona, Sara Ayhan, Ansten Klev, Will Stafford, and Vít Punčochář for their most helpful notes on earlier versions of this paper. I would also like to thank the two anonymous referees for their helpful comments and the audiences of the 35th Logica symposium (The Premonstratensian Monastery of Teplá, 2022), the Seminar on Applied Mathematical Logic (Institute of Computer Science, Czech Academy of Sciences, 2023), and of the 14th Tbilisi Symposium on Logic, Language and Computation (TbiLLC, Telavi, 2023) where earlier versions of this paper were presented and discussed.

References

Samson Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51(1-2):1–77, 1991. 10.1016/0168-0072(91)90065-T.

Peter Aczel. Saturated intuitionistic theories. In H. A. Schmidt, K. Schütte, and H. J. Thiele, editors, *Contributions to Mathematical Logic*, pages 1–11. North-Holland, Amsterdam, 1968.

Ulrich Berger, Stefan Berghofer, Pierre Letouzey, and Helmut Schwichtenberg. Program extraction from normalization proofs. *Studia Logica*, 82(1):25–49, 2006. 10.1007/S11225-006-6604-5.

Ivano Ciardelli, Rosalie Iemhoff, and Fan Yang. Questions and dependency in intuitionistic logic. *Notre Dame Journal of Formal Logic*, 61(1):75–115, 2020. 10.1215/00294527-2019-0033.

Andrea Condoluci and Matteo Manighetti. Admissible tools in the kitchen of intuitionistic logic. In *Electronic Proceedings in Theoretical Computer Science, EPTCS*, volume 281, pages 10–23, Waterloo, 2018. Open Publishing Association. 10.4204/EPTCS.281.2.

Haskell Curry and Robert Feys. *Combinatory Logic*. Combinatory Logic. North-Holland, Amsterdam, 1958.

Wagner de Campos Sanz, Thomas Piecha, and Peter Schroeder-Heister. Constructive semantics, admissibility of rules and the validity of Peirce's law. *Logic Journal of the IGPL*, 22(2):297–308, 2014. 10.1093/JIGPAL/JZT029.

Michael Dummett. A propositional calculus with denumerable matrix. *Journal of Symbolic Logic*, 24(2):97–106, 1959. 10.2307/2964753.

Michael Dummett. *The Logical Basis of Metaphysics*. Duckworth, London, 1991.

J. Michael Dunn and Robert K. Meyer. Algebraic completeness results for Dummett's LC and its extensions. *Mathematical Logic Quarterly*, 17(1):225–230, 1971. 10.1002/MALQ.19710170126.

Gerhard Gentzen. Untersuchungen über das logische Schließen. I. *Mathematische Zeitschrift*, 39(1):176–210, 1935. 10.1007/BF01201353.

Gerhard Gentzen. *The Collected Papers of Gerhard Gentzen*. Studies in logic and the foundations of mathematics. North-Holland, Amsterdam, 1969.

Christopher Alan Goad. *Computational Uses of the Manipulation of Formal Proofs*. PhD thesis, Stanford University, Stanford, 1980.

Kurt Gödel. Zum intuitionistischen Aussagenkalkül. *Anzeiger Akademie der Wissenschaften Wien, Math.-naturwissensch. Klasse*, (69):65–66, 1932.

Petr Hájek. *Metamathematics of Fuzzy Logic*. Kluwer, Dordrecht, 1998.

Ronald Harrop. On disjunctions and existential statements in intuitionistic systems of logic. *Mathematische Annalen*, 132(4):347–361, 1956. 10.1007/BF01360048/METRICS.

Ronald Harrop. Concerning formulas of the types A→BvC, A→(Ex)B(x) in intuitionistic formal systems. *Journal of Symbolic Logic*, 25(1):27–32, 1960. 10.2307/2964334.

Susumu Hayashi and Hiroshi Nakano. *PX: A Computational Logic.* MIT Press, Cambridge, MA, 1988.

William Alvin Howard. The formulae-as-types notion of construction. In Haskell B. Curry, J. Roger Hindley, and Jonathan P. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism.* Academic Press, London, 1980.

Rosalie Iemhoff. On the admissible rules of intuitionistic propositional logic. *Journal of Symbolic Logic*, 66(1):281–294, 2001. 10.2307/2694922.

Rosalie Iemhoff. Intermediate logics and Visser's rules. *Notre Dame Journal of Formal Logic*, 46(1):65–81, 2005. 10.1305/NDJFL/1107220674.

Stephen C. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10(4):109–124, 1945. 10.2307/2269016.

Stephen C. Kleene. Disjunction and existence under implication in elementary intuitionistic formalisms. *Journal of Symbolic Logic*, 27(1):11–18, 1962. 10. 2307/2963675.

Georg Kreisel and Hilary Putnam. Unableitbarkeitsbeweismethode für den Intuitionistischen Aussagenkalkül. *Zeitschrift für Mathematische Logik and Grundlagen der Mathematik*, (3):74–78, 1957.

Jan Łukasiewicz. On the intuionistic theory of deduction. *Indagationes Mathematicae*, (14):202–212, 1952.

Paolo Mancosu, Sergio Galvan, and Richard Zach. *An Introduction to Proof Theory: Normalization, Cut-Elimination, and Consistency Proofs.* Oxford University Press, Oxford, 2021.

Per Martin-Löf. *Intuitionistic Type Theory: Notes by Giovanni Sambin of a series of lectures given in Padua, June 1980.* Bibliopolis, Napoli, 1984.

Pierluigi Minari and Andrzej Wronski. The property (HD) in intermediate logics: a partial solution of a problem of H. Ono. *Reports on Mathematical Logic*, (22):21–25, 1988.

Sara Negri and Jan von Plato. *Structural Proof Theory.* Cambridge University Press, Cambridge, 2001.

Nikolai N. Nepeivoda. Logical Approach to Programming. In L. Jonathan Cohen, Jerzy Łoś, Helmut Pfeiffer, and Klaus-Peter Podewski, editors, *Logic, Methodology and Philosophy of Science VI. Proceedings of the Sixth International Congress of Logic, Methodology and Philosophy of Science*, volume 104, pages 109–122. Elsevier, Amsterdam, 1982. 10.1016/S0049-237X(09)70185-5.

Nikolai Nikolaevich Nepeivoda. A relation between the natural deduction rules and operators of higher level algorithmic languages. *Doklady Akademii Nauk SSSR*, 239(3):526–529, 1978.

Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's type theory: an introduction.* Clarendon Press, Oxford, 1990.

Ivo Pezlar. A note on paradoxical propositions from an inferential point of view. In Martin Blicha and Igor Sedlár, editors, *The Logica Yearbook 2020*, pages 183–199, London, 2021. College Publications.

Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. *Proceedings - Symposium on Logic in Computer Science*, pages 221–230, 2001. 10.1109/LICS.2001.932499.

Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001. 10.1017/S0960129501003322.

Antonio Piccolomini d'Aragona. A note on schematic validity and completeness in Prawitz's semantics. In F. Bianchini, V. Fano, and P. Graziani, editors, *Current Topics in Logic and the Philosophy of Science. Papers from SILFS 2022 conference*, London, 2024. College Publications.

Thomas Piecha, Wagner de Campos Sanz, and Peter Schroeder-Heister. Failure of completeness in proof-theoretic semantics. *Journal of Philosophical Logic*, 44(3):321–335, 2014. 10.1007/S10992-014-9322-X.

Valery Plisko. A survey of propositional realizability logic. *Bulletin of Symbolic Logic*, 15(1):1–42, 2009. 10.2178/BSL/1231081768.

Dag Prawitz. *Natural Deduction: A Proof-theoretical Study.* Almqvist & Wiksell, Stockholm, reprinted edition, 1965.

Dag Prawitz. Ideas and results in proof theory. In Jens Erik Fenstad, editor, *Proceedings of the second scandinavian logic symposium. Studies in logic and the foundations of mathematics*, pages 235–307. North-Holland Publishing Company, 1971.

Dag Prawitz. Towards a foundation of a general proof theory. In P. Suppes, L. Henkin, A. Joja, and G. C. Moisil, editors, *Proceedings of the Fourth International Congress for Logic, Methodology and Philosophy of Science, Bucharest, 1971*, pages 225–250. North-Holland Publishing Company, 1973.

Arthur Prior. The runabout inference ticket. *Analysis*, 21(1):38–39, 1960. 10.1093/analys/21.2.38.

Tadeusz Prucnal. On two problems of Harvey Friedman. *Studia Logica*, (38):257–262, 1979.

Vít Punčochář. A generalization of inquisitive semantics. *Journal of Philosophical Logic*, 45(4):399–428, 2016. 10.1007/S10992-015-9379-1.

Helena Rasiowa. Constructive theories. *Bulletin de l'Academie polonaise des sciences. Serie des sciences mathematiques, astronomiques, et physiques*, 2: 121–124, 1954.

Paul Roziére. Admissible and derivable rules in intuitionistic logic. *Mathematical Structures in Computer Science*, 3(2):129–136, 1993. 10.1017/S0960129500000165.

James Toshio Sasaki. *Extracting Efficient Code From Constructive Proofs*. PhD thesis, Cornell University, Ithaca, 1986.

Peter Schroeder-Heister. Validity concepts in proof-theoretic semantics. *Synthese*, 148(3):525–571, 2006. 10.1007/s11229-004-6296-1.

Peter Schroeder-Heister. Generalized elimination inferences, higher-level rules, and the implications-as-rules interpretation of the sequent calculus. In Luiz Carlos Pereira, Edward Hermann Haeusler, and Valeria de Paiva, editors, *Advances in Natural Deduction*, volume 39, pages 1–29. Springer, Dordrecht, 2014. 10.1007/978-94-007-7548-0_1.

Helmut Schwichtenberg and Stanley S. Wainer. *Proofs and Computations*. Cambridge University Press, Cambridge, 2012. 10.1017/CBO9781139031905.

Jan Smith. An Interpretation of Kleene's Slash in Type Theory. In Gerard Huet and Gordon Plotkin, editors, *Logical Environments*, pages 189–197. Cambridge University Press, Cambridge, 1993.

Will Stafford. Proof-theoretic semantics and inquisitive logic. *Journal of Philosophical Logic*, 50(5):1199–1229, 2021. 10.1007/S10992-021-09596-7.

William Walker Tait. Intensional interpretations of functionals of finite type I. *Journal Symbolic Logic*, 32(2):198–212, 1967.

Neil Tennant. *Natural Logic*. Edinburgh University Press, Edinburgh, 1978.

Luca Tranchini. Proof, meaning and paradox: some remarks. *Topoi*, 38(3): 591–603, 2019. 10.1007/s11245-018-9552-6.

Anne S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer, Berlin, 1973. 10.1007/BFB0066739.

Guo-Qiang Zhang. *Logic of Domains*. Birkhäuser, Boston, MA, 1991. 10.1007/978-1-4612-0445-9.