Herbrand Schemes for Cyclic Proofs

Bahareh Afshari[1], Sebastian Enqvist[2], and Graham E. Leigh[1]

[1]Department of Philosophy, Linguistics and Theory of Science, University of Gothenburg, Sweden
[2]Department of Philosophy, Stockholm University, Sweden

6th February 2024

Abstract

Recent work by Afshari et al. introduce a notion of *Herbrand schemes* for first-order logic by associating a higher-order recursion scheme to a sequent calculus proof. Calculating the language of associated Herbrand schemes directly yields Herbrand disjunctions. As such, these schemes can be seen as programs extracted from proofs. In this article, this computational interpretation is explored further by removing the restriction of *acyclicity* from Herbrand schemes which amounts to admitting recursively defined programs. It is shown that the notion of proof corresponding to these generalised Herbrand schemes is *cyclic* proofs, considered here in the context of classical theories of inductively defined predicates. In particular, for proofs with end sequents of a form generalising the notion of Σ_1-sequents in the first-order setting, Herbrand schemes extract Herbrand expansions from cyclic proofs via a simulation of non-wellfounded cut elimination.

1 Introduction

The aim of this work is to connect two strands of research in proof theory, both of which concern computational interpretations of proofs in sequent calculus. One is the pursuit of computational/constructive content for classical logic; the other is representations of induction and recursion in systems of non-wellfounded proofs.

The first avenue has a long history (see e.g. [9, 13, 10, 18, 17]). Our focus is the classical logic counterpart to witness extraction: *Herbrand's theorem.* For a valid Σ_1-formula A, Herbrand's theorem states the existence of a finite set of 'witnesses' for the existential quantifiers in A that, instantiated, induce a propositional tautology. This is in contrast to the *single* witness that can be provided for an intuitionistically valid existential formula. The theorem can be lifted to arbitrary formulas by Skolemizing universal quantifiers.

The set of terms witnessing Herbrand's theorem (henceforth 'Herbrand sets') are readily extracted from cut-free sequent calculus proofs. If we view cut elimination as program execution via a form of Curry-Howard correspondence, we

`bahareh.afshari@gu.se`; `sebastian.enqvist@philosophy.su.se`; `graham.leigh@gu.se`

see that classical sequent calculus proofs represent non-deterministic computations, with elements of the extracted Herbrand set representing possible return values of different threads. This computational perspective on classical sequent calculus proofs was explored in depth in Urban's thesis [22]. The background for our work is a series of papers taking a grammar-theoretic approach to Herbrand extraction, culminating in [2] where sequent calculus proofs are associated with *higher-order recursion schemes* [16] that can be used to directly compute the Herbrand content of a proof. These are called *Herbrand schemes*, and can be seen as programs extracted from sequent calculus proofs. We have recently refined this approach in a companion paper to the present work [1].

The second line of research underlying the basis of the work presented here is the study of *non-wellfounded proofs* and in particular *cyclic* proofs, the non-wellfounded proofs with a finite graphical representation. Such proofs have been extensively studied for a wide range of logical systems that deal with induction in one form or another. Examples of particular relevance to this paper are cyclic proof systems for theories of inductively defined predicates [6, 7, 8, 5] and for arithmetic [5, 12, 21]. From a Curry-Howard-style computational perspective cyclic proofs are natural with recursion built into the proof system itself rather than explicitly added in the form of induction rules/axioms. This perspective on cyclic proofs can be said to have started with the work of Santocanale [19] and later Fortier [14], and has been explored by a number of authors, most notably, in the context of linear logic with explicit fixed-point operators [4, 3]. Of particular interest for us is the work of Das on cyclic proofs for Gödel's System T [11], especially given previously known connections between Herbrand extraction and functional interpretations [15].

We extend Herbrand schemes beyond first-order logic to the setting of cyclic proofs for classical theories of inductively defined predicates, in order to obtain a richer computational interpretation of classical logic. The choice of cyclic proofs for this extension is rather natural as it is already present in the Herbrand schemes. To obtain recursion schemes that match the (limited) level of expressivity in first-order logic, a restriction called *acyclicity* is imposed in [2], which amounts to removing the 'recursion' from recursion schemes. We show lifting the restriction of acyclicity corresponds precisely to taking the step from well-founded to cyclic proofs. Stated as a slogan, unrestricted Herbrand schemes are to cyclic proofs what acyclic Herbrand schemes are to well-founded proofs. The companion paper [1] provides the groundwork for taking this step by streamlining the formulation of Herbrand schemes so that the introduction of inductive predicates and treatment of quantifiers and connectives can be carried out smoothly.

The main technical contribution is to show that Herbrand schemes can 'simulate' the process of 'multi-cuts' elimination. In particular, when the end sequent of a cyclic proof is of a special form generalising the notion of Σ_1 sequent from first-order logic, cut elimination leads to a finite well-founded proof from which we can read off a Herbrand expansion (in a sense that will be made precise). As a consequence of our simulation theorem, Herbrand schemes associated with proofs of such 'constructive' sequents can be used to compute Herbrand expansions.

The restriction of the main result to proofs with constructive end sequents and, hence well-founded normal forms, may seem restrictive given our aim to view Herbrand schemes as a computational interpretation of *cyclic* proofs. It is worthwhile noting that (1) *every* proof has an associated Herbrand scheme, regardless of the form of the end sequent, and (2) Herbrand schemes are *compositional* with respect to the cut rule, in the sense that cuts are interpreted as function application. This means that every proof is associated with a well-defined 'program' whose type is determined by the end sequent, and that can be applied via cuts. Constructive sequents can be seen as a form of ground type, from which the computation yields a concrete value in the form of a Herbrand expansion.

To illustrate the inner-workings of Herbrand schemes, we look a cyclic proof π of the pigeonhole principle for an arbitrary number of pigeonholes: for any function $f\colon \mathbb{N} \to \mathbb{N}$ bounded by some natural number n, there are $k < m$ such that $f(k) = f(m)$. In order to obtain concrete sets of possible values for k, m, we can apply this proof to a concrete natural number n via a cut on some proof that 'n is a natural number'. In the case $n = 1$, for which we will compute a Herbrand set, this yields the pigeonhole principle for *two* pigeonholes previously studied in [22] and later in [2, 1]. The point of interest here is that the Herbrand scheme associated with the proof π can be seen as an independent and re-usable program, that has as its extensional interpretation a relation $R_\pi \subseteq \mathbb{N} \times \mathcal{P}^{<\omega}(\mathbb{N} \times \mathbb{N})$ associating concrete values for n with finite sets of values for k, m,i.e., Herbrand sets. From our simulation theorem it follows that this relation R_π offers at least one Herbrand set for each given value for n.

One crucial step in the making is to interpret the cut rule compositionally as function application; this was not quite the case for the original version of Herbrand schemes presented in [2]. Indeed, another motivation of our reformulation of Herbrand schemes in [1] was to make Herbrand schemes fully compositional with respect to cut, which is achieved by introducing a form of 'call-by-current-continuation'-like operator. We follow the same approach here.

Outline of the paper We start by introducing classical theories of inductively defined predicates and their non-wellfounded/cyclic proof systems in Section 2. In Section 3 we introduce the type theory that will form the basis of our Herbrand schemes, which are then introduced in Section 4. Section 5 introduces the precise notion of a Herbrand expansion in this context, as well as the language of a Herbrand scheme, and proves some basic observations. Herbrand expansions are defined for sequents in a specific form, called *constructive* sequents. In Section 6 we apply the framework in an analysis of a cyclic proof of the pigeonhole principle. Section 7 introduces cut reduction and permutation rules, using so called 'multicuts' that are common in the literature on non-wellfounded proofs, and show how multicut reductions and permutations can be simulated by rewrites for Herbrand schemes. Finally in Section 8, we prove our main result, showing that Herbrand expansions obtained by eliminating cuts can also be computed directly from the Herbrand scheme associated with a proof. Together with a cut elimination theorem derived from [20], we obtain as a corollary that the Herbrand scheme associated with any proof of a constructive sequent always derives some Herbrand expansion of the end-sequent. We finish with some concluding remarks in Section 9.

2 Inductive predicates and non-wellfounded proofs

We introduce classical theories of inductive predicates and cyclic proofs for such theories, essentially following the approach of [6, 7, 8].

2.1 Inductive predicates and languages

We begin with the definition of a vocabulary:

Definition 1. A *vocabulary* $\mathcal{L}$ is a tuple $(C, F, O, I, \mathsf{ar}, |\cdot|)$ consisting of a non-empty set C of individual constants, a set F of function symbols, a set O of *ordinary* predicates, a set I of *inductive* predicates, a function $\mathsf{ar}\colon F \cup O \cup I \to \mathbb{N}$ assigning an *arity* to each symbol, and a map $|\cdot|\colon I \to \mathsf{Ord}$ assigning an ordinal *rank* to each inductive predicate. We write $P \prec_{\mathcal{L}} Q$ if $|P| < |Q|$, and $P \preceq_{\mathcal{L}} Q$ if $|P| \leq |Q|$.

Let a vocabulary $\mathcal{L}$ as above be fixed. We assume that individual variables are partitioned into two disjoint infinite sets, one set of *ordinary* variables and one set of *eigenvariables*. As the set C of constants is non-empty we fix a distinguished element $\mathsf{c} \in C$. The $\mathcal{L}$-terms are defined as usual. The set of $\mathcal{L}$-*clauses* is defined by the grammar:

$$A := P\vec{t} \mid Q\vec{s} \mid \neg A$$

where P and Q range over ordinary and inductive predicates respectively and $\vec{t}, \vec{s}$ are tuples of $\mathcal{L}$-terms of the appropriate lengths. An *ordinary* $\mathcal{L}$-clause is one that does not contain any inductive predicates. Note that an ordinary clause is either of the form $Q\vec{s}$ or $\neg Q\vec{s}$ where Q is an ordinary predicate. Hence we will also refer to ordinary $\mathcal{L}$-clauses as *literals*.

The terminology of *clauses* rather than formulas is meant to put emphasis on the frugality of the syntax, with negation as the only explicit connective. We shall see later that formulas using the standard connectives and first-order quantifiers can be introduced as abbreviations, and we to expressions in the extended syntax using such abbreviations as formulas.

Definition 2. A *language* is a pair $(\mathcal{L}, \mathcal{R})$ consisting of a vocabulary together with an assignment $\mathcal{R}$ of a finite non-empty set of *production rules* to each inductive predicate. We assume that the set $\mathcal{R}(r)$ is listed in a fixed enumeration.

Given an inductive predicate P, every production rule $r \in \mathcal{R}(P)$ is associated with a tuple of variables $\vec{x}$ of length $n \leq \mathsf{ar}(P)$ called its *parameters*, and a tuple of variables $\vec{y}$ called its *input variables*, and the rule has the form:

$$\frac{A_1 \quad \cdots \quad A_k}{P(\vec{x}, \vec{t}(\vec{x}, \vec{y}))}$$

subject to the following constraints:

1. If an inductive predicate Q occurs in A_i, then $Q \preceq_{\mathcal{L}} P$.
2. If an inductive predicate Q occurs negatively in A_i, then $Q \prec_{\mathcal{L}} P$.

We assume for notational convenience that the arguments of P are ordered so that the parameters of a production rule are always the first n arguments of P in the conclusion.

For example, the usual natural number predicate N has the following two production rules:

$$\frac{}{N0} \qquad \frac{Ny}{N(sy)}$$

Here, y is an input variable and s a certain function symbol of arity 1. We can define a predicate L for lists of natural numbers by isolating a constant ϵ, binary function symbol $\frown$ and the production rules:

$$\frac{}{L\epsilon} \qquad\qquad \frac{Nx \quad Ly}{L(x^\frown y)}$$

For the rank of the two inductive predicates we could choose $|N| = |N|$. Both x and y are input variables in the second production rule above. Note that $x^\frown y$ expresses the 'prepend' operation that attaching a natural number x to the left side of a list y.

Definition 3. A *model* is a first-order structure for the vocabulary consisting of inductive and ordinary predicates along with individual constants and function symbols, subject to the constraint that each inductive predicate is interpreted as the smallest subset of the domain closed under all its production rules. An *assignment* is a function mapping individual variables to elements of the domain as usual. The satisfaction relation $M, V \vDash A$ and the value $[\![t]\!]_M^V$ of a term, given a model M and valuation V, are defined as expected.

2.2 Extended language

Given a language $\mathcal{L}$, the *extended language* is defined by the following grammar:

$$A := R\vec{t} \mid \neg A \mid A \vee A \mid A \wedge A \mid \exists x A$$

where R is either an inductive predicate or an ordinary predicate. Free and bound variables are defined as usual. We can view every formula in this extended language as an abbreviation of a formula expressed in the basic language. More precisely, we first introduce the following uniform constructions of inductive predicates:

Abstraction Given a formula A with all free variables belonging to $\vec{x}$, we define the inductive predicate $\lambda\vec{x}.A$ with the following production rule:

$$\frac{A[\vec{y}/\vec{x}]}{(\lambda\vec{x}.A)\vec{y}}$$

Parameters of the rule are $\vec{y}$. There are no input variables.

Sum Given n-ary and m-ary predicates P and Q respectively, we introduce the $n+m$-ary predicate $P+Q$:

$$\frac{P\vec{x}}{(P+Q)\vec{x}\vec{y}} \qquad \frac{Q\vec{y}}{(P+Q)\vec{x}\vec{y}}$$

Parameters are $\vec{x}, \vec{y}$; there are no input variables.

Product Given n-ary respectively m-ary predicates P and Q respectively, define $n+m$-ary $P \times Q$ with productions:

$$\frac{P\vec{x} \quad Q\vec{y}}{(P \times Q)\vec{x}\vec{y}}$$

Parameters are $\vec{x}, \vec{y}$; there are no input variables.

Projection Given an $n+1$-ary predicate P, define the n-ary predicate ΣP with the following production:

$$\frac{P\alpha\vec{x}}{(\Sigma P)\vec{x}}$$

Parameters are $\vec{x}$; the input variable is α.

Combining the above construction with abstraction we can project other arguments than the first. With P as above and $i < n+1$, define the n-ary predicate $\Sigma_i P$ to be $\Sigma\lambda x\vec{y}\vec{z}.\, P\vec{y}x\vec{z}$ where $|\vec{y}| = i$ and $|\vec{z}| = n-i$. This inductive predicate has the derived rule:

$$\frac{P\vec{y}\alpha\vec{z}}{(\Sigma_i P)\vec{y}\vec{z}}$$

Parameters of the rule are $\vec{y}, \vec{z}$ where $|\vec{y}| = i$; input variable is α.

Translation Finally, we can define a translation t from the extended language to the basic language as follows. We define t to be the identity on atomic sentences, and put $t(\neg A) = \neg t(A)$ (note that the basic language is closed under negation). For disjunction, given formulas A, B we put

$$F_{A \vee B} := \lambda\vec{x}.(F_A + F_B)\vec{x}\vec{x}$$

where $\vec{x}$ is a list of all free variables occurring in A, B. Similarly we put:

$$F_{A \wedge B} := \lambda\vec{x}.(F_A \times F_B)\vec{x}\vec{x}$$

For the existential quantifier, given a formula A we put:

$$F_{\exists z A} = \Sigma_i F_A$$

where $\vec{x}z\vec{y}$ is a list of all the free variables of A, $|\vec{x}| = i$.

Given a formula $A(\vec{x})$ with free variables among $\vec{x}$, we take $A(\vec{u})$ as an abbreviation of $F_A\vec{u}$.

2.3 Non-wellfounded sequent calculus

We now introduce the sequent calculus associated with a language $\mathcal{L}$. Suppose P is an inductive predicate with a production rule r of the form:

$$\frac{A_1 \quad \cdots \quad A_n}{P(\vec{x}, \vec{t}(\vec{x}, \vec{y}))}$$

Then P has the following right sequent calculus rule associated with the i-th production rule r:

$$\frac{\Gamma_1 \Rightarrow \Theta_1, A_1[\vec{s}/\vec{x}, \vec{u}/\vec{y}] \quad \cdots \quad \Gamma_k \Rightarrow \Theta_k, A_k[\vec{s}/\vec{x}, \vec{u}/\vec{y}]}{\Gamma_1, \dots, \Gamma_n \Rightarrow \Theta_1, \dots, \Theta_n, P(\vec{s}, \vec{t}(\vec{s}, \vec{u}))}\, R(P, i)$$

$$\frac{\Gamma \Rightarrow \Delta}{A, \Gamma \Rightarrow \Delta}\,\mathsf{wL} \qquad \frac{A, A, \Gamma \Rightarrow \Delta}{A, \Gamma \Rightarrow \Delta}\,\mathsf{cL} \qquad \frac{\Gamma, A, \Sigma \Rightarrow \Delta}{A, \Gamma, \Sigma \Rightarrow \Delta}\,\mathsf{pL}$$

$$\frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, A}\,\mathsf{wR} \qquad \frac{\Gamma \Rightarrow \Delta, A, A}{\Gamma \Rightarrow \Delta, A}\,\mathsf{cR} \qquad \frac{\Gamma \Rightarrow \Delta, A, \Pi}{\Gamma \Rightarrow \Delta, \Pi, A}\,\mathsf{pR}$$

$$\frac{\Gamma \Rightarrow \Delta, A \qquad A, \Sigma \Rightarrow \Pi}{\Gamma, \Sigma \Rightarrow \Delta, \Pi}\,\mathsf{cut}$$

Figure 1: Structural rules of the sequent calculus

$$\frac{}{P\vec{t} \Rightarrow P\vec{t}}\,\mathsf{id} \qquad \frac{\Gamma \Rightarrow \Delta, A}{\neg A, \Gamma \Rightarrow \Delta}\,\neg\mathsf{L}$$

$$\frac{\Gamma \Rightarrow \Delta}{\Gamma[\sigma] \Rightarrow \Delta[\sigma]}\,\mathsf{Sub}(\sigma) \qquad \frac{A, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg A}\,\neg\mathsf{R}$$

Figure 2: Logical rules of the sequent calculus

The left rule associated with P is:

$$\frac{\Delta_1, \Gamma[\vec{w}_1/\vec{z}] \Rightarrow \Theta[\vec{w}_1/\vec{z}] \qquad \cdots \qquad \Delta_k, \Gamma[\vec{w}_k/\vec{z}] \Rightarrow \Theta[\vec{w}_k/\vec{z}]}{P(\vec{s}, \vec{u}), \Gamma[\vec{u}/\vec{z}] \Rightarrow \Theta[\vec{u}/\vec{z}]}\,L(P, \vec{\alpha})$$

where $\vec{\alpha}$ is a tuple of eigenvariables and, for each i, if the i-th production rule for P has premisses $A_1, \ldots, A_k$ and conclusion $P(\vec{x}, \vec{t}_i(\vec{x}, \vec{\alpha}))$ then: $\vec{w}_i = \vec{t}_i(\vec{u}, \vec{\alpha})$ and $\Delta_i = A_1[\vec{u}/\vec{x}][\vec{\alpha}/\vec{y}], ..., A_k[\vec{u}/\vec{x}][\vec{\alpha}/\vec{y}]$.

For example, the natural number predicate N has the following right rules:

$$\frac{}{\Gamma \Rightarrow \Theta, N0}\,R(N, 0) \qquad \frac{\Gamma \Rightarrow \Theta, Nt}{\Gamma \Rightarrow \Theta, Nst}\,R(N, 1)$$

The left rule for N is:

$$\frac{\Gamma[0/x] \Rightarrow \Theta[0/X] \qquad N\alpha, \Gamma[s\alpha/x] \Rightarrow \Theta[s\alpha/x]}{Nt, \Gamma \Rightarrow \Theta[t/x]}\,L(N, \alpha)$$

The predicate L for lists of natural numbers has the following right sequent calculus rules:

$$\frac{}{\Gamma \Rightarrow \Delta, L\epsilon}\,R(L, 0) \qquad \frac{\Gamma_0 \Rightarrow \Delta_0, Ns \qquad \Gamma_1 \Rightarrow \Delta_1, Lt}{\Gamma_0, \Gamma_1 \Rightarrow \Delta_0, \Delta_1, L(s^\frown t)}\,R(L, 1)$$

Left rule:

$$\frac{\Gamma[\epsilon/x] \Rightarrow \Delta[\epsilon/x] \qquad N\alpha, L\beta, \Gamma[\alpha^\frown\beta/x] \Rightarrow \Delta[\alpha^\frown\beta/x]}{Lt, \Gamma[t/x] \Rightarrow \Delta[t/x]}\,L(L, \alpha\beta)$$

Besides these rules, the sequent calculus associated with the language $\mathcal{L}$ has the rules displayed in figures 1 and 1. In the axioms id, P is required to be an ordinary predicate. In the rule $\mathsf{Sub}(\sigma)$, σ is a substitution of $\mathcal{L}$-terms for variables.

We also introduce the following abbreviation for a derived rule:

$$\dfrac{\Gamma[\tau] \Rightarrow \Theta[\tau], A[\tau] \quad A, \Sigma \Rightarrow \Pi}{\Gamma[\tau], \Sigma[\tau] \Rightarrow \Theta[\tau], \Pi[\tau]}\,\mathsf{app} \;:=\; \dfrac{\Gamma[\tau] \Rightarrow \Theta[\tau], A[\tau] \quad \dfrac{A, \Sigma \Rightarrow \Pi}{A[\tau], \Sigma[\tau] \Rightarrow \Pi[\tau]}\,\mathsf{Sub}(\tau)}{\Gamma[\tau], \Sigma[\tau] \Rightarrow \Theta[\tau], \Pi[\tau]}\,\mathsf{cut}$$

For example:

$$\dfrac{\dfrac{}{N0} \quad \dfrac{\dfrac{}{Nx \Rightarrow Nx} \quad \dfrac{}{\Rightarrow L\epsilon}}{Nx \Rightarrow L(x^\frown \epsilon)}}{\Rightarrow L(0^\frown \epsilon)}\,\mathsf{app}$$

Viewed as a program, the right subproof takes an arbitrary natural number and wraps it as a list (i.e. it is the unit of the list monad applied to $\mathbb{N}$), and the application of the rule (app) applies this program to the concrete natural number 0.

By a *derivation* we mean any tree constructed according to the sequent rules. We allow derivations to be infinite, which is equivalent to being non-wellfounded since derivations are finitely branching trees.

Definition 4. A *branch* of a derivation π is a maximal (finite or infinite) sequence $(B_i)_{i<\kappa}$ of sequents, where $\kappa \leq \omega$, such that B_0 is the root sequent of π and B_{i+1} is a premiss of B_i whenever $i + 1 < \kappa$.

A *trace* or *thread* on a branch $(B_i)_{i<\kappa}$ of a derivation π is a (finite or infinite) sequence $(A_i)_{i<\kappa}$ of formulas such that A_i occurs in B_i, and if $i + 1 < k$ then either A_i is a side formula occurrence with $A_{i+1} = A_i$, or A_i is the principal formula of the conclusion and A_{i+1} a minor formula of the premiss B_{i+1}. An infinite thread $(A_i)_{i<\omega}$ is said to be *progressive* if, for infinitely many i, A_i is the principal formula of a left rule for some inductive predicate.

Finally, a *valid derivation*, or just a *proof*, is a derivation in which every infinite branch contains at least one progressive thread. A proof is called *regular* if it has only finitely many subproofs up to isomorphism. We also refer to regular proofs as *cyclic proofs*.

Definition 5. A derivation π is called *clean* if, whenever an eigenvariable α appears in a term introduced by a right rule application, then α must appear in the conclusion of the rule application.

2.4 Proofs as programs

Our aim in this paper is to generalize the grammar-theoretic approach to Herbrand extraction introduced in [2] to provide a computational interpretation of non-wellfounded proofs. We will assign higher-order recursion schemes, called *Herbrand schemes*, to sequent calculus proofs. The recursion scheme associated with a proof can be viewed as a program, and program execution corresponds to term rewriting.

As a warm-up example consider an inductive predicate S for 'sum' with the following production rules:

$$\dfrac{}{Sx0x} \qquad \dfrac{Sxyz}{Sx(sy)(sz)}$$

The following cyclic proof π_S can be viewed as a program to compute the sum of two natural numbers:

$$
\begin{prooftree}
\AxiomC{}
\UnaryInfC{$\Rightarrow Sx0x$}
\UnaryInfC{$\Rightarrow \exists zSx0z$}
\AxiomC{$Ny \Rightarrow \exists zSxyz \dagger$}
\UnaryInfC{$N\alpha \Rightarrow \exists zSx\alpha z$}
\AxiomC{}
\UnaryInfC{$Sx\alpha\beta \Rightarrow Sx\alpha\beta$}
\UnaryInfC{$Sx\alpha\beta \Rightarrow Sx(s\alpha)(s\beta)$}
\UnaryInfC{$Sx\alpha\beta \Rightarrow \exists zSx(s\alpha)z$}
\UnaryInfC{$\exists zSx\alpha z \Rightarrow \exists zSx(s\alpha)z$}
\RightLabel{cut}
\BinaryInfC{$N\alpha \Rightarrow \exists zSx(s\alpha)z$}
\BinaryInfC{$Ny \Rightarrow \exists zSxyz \dagger$}
\end{prooftree}
$$

This is a finite representation of the non-wellfounded proof obtained by unfolding the graph in which the two sequents labelled (†) are identified. Note that this is an intuitionistically valid proof, i.e. there are no right applications of contraction. So we can expect its computational interpretation to be a *deterministic* program, the extension of which is a function $f : \mathbb{N} \to \mathbb{N}$ (taking the left argument as a fixed parameter). For classical proofs in general, we get In order to get a concrete output from the program described by this proof, we have to apply it to concrete input values via a cut. For example:

$$
\begin{prooftree}
\AxiomC{}
\UnaryInfC{$N0$}
\UnaryInfC{$Ns0$}
\UnaryInfC{$Nss0$}
\AxiomC{}
\UnaryInfC{$\Rightarrow Sx0x$}
\UnaryInfC{$\Rightarrow \exists zSx0z$}
\AxiomC{$Ny \Rightarrow \exists zSxyz \dagger$}
\UnaryInfC{$N\alpha \Rightarrow \exists zSx\alpha z$}
\AxiomC{}
\UnaryInfC{$Sx\alpha\beta \Rightarrow Sx\alpha\beta$}
\UnaryInfC{$Sx\alpha\beta \Rightarrow Sx(s\alpha)(s\beta)$}
\UnaryInfC{$Sx\alpha\beta \Rightarrow \exists zSx(s\alpha)z$}
\UnaryInfC{$\exists zSx\alpha z \Rightarrow \exists zSx(s\alpha)z$}
\RightLabel{cut}
\BinaryInfC{$N\alpha \Rightarrow \exists zSx(s\alpha)z$}
\BinaryInfC{$Ny \Rightarrow \exists zSxyz \dagger$}
\RightLabel{app}
\BinaryInfC{$\Rightarrow \exists zS(ss0)(ss0)z$}
\end{prooftree}
$$

Eliminating the cuts yields the finite cut-free proof:

$$
\begin{prooftree}
\AxiomC{}
\UnaryInfC{$\Rightarrow S(ss0)(0)(ss0)$}
\UnaryInfC{$\Rightarrow S(ss0)(s0)(sss0)$}
\UnaryInfC{$\Rightarrow S(ss0)(ss0)(ssss0)$}
\UnaryInfC{$\Rightarrow \exists zS(ss0)(ss0)z$}
\end{prooftree}
$$

Here we have applied the program to compute the sum of 2 and 2, and we get the value 4 as the witness extracted for the existential quantifier. Our perspective here is that the computational content of the proof π_S - its denotational semantics - is represented by this input-output relationship, i.e. how concrete witnesses can be extracted from proofs in which it features via applications/cuts. To make this perspective clear, it is necessary to identify those end sequents for which it makes sense to expect a proof to provide concrete witnesses.

Definition 6. Given a language $\mathcal{L}$, we define the notion of *constructive* inductive predicates and constructive formulas by well-founded induction on $\prec$ as follows. An inductive predicate P is said to be every constructive if, for every production rule r associated with P, every premiss A_i of r is P-constructive. An $\mathcal{L}$-formula A is P-constructive if it is either an ordinary $\mathcal{L}$-formula, or of the form $P\vec{t}$, or of the form $Q\vec{t}$ where Q is a constructive inductive predicate. A

$$\frac{}{\bot : \Sigma} \qquad \frac{\sigma : \Sigma \quad s : \iota \quad \alpha \in \mathsf{FV}}{[s/\alpha]\sigma : \Sigma} \qquad \frac{t \text{ an } \mathcal{L}\text{-term} \quad \sigma : \Sigma}{t \cdot \sigma : \iota}$$

Figure 3: Typing rules for substitution stacks and individuals.

formula is said to be *constructive* if it is either an ordinary $\mathcal{L}$-formula or of the form $P\vec{t}$ where P is a constructive inductive predicate.

Finally, a *constructive sequent* is a sequent of the form:

$$A_1, \ldots, A_n \Rightarrow B$$

where each A_i is either a literal or the negation of a constructive formulas, and B is constructive.

Note that the natural numbers predicate N and the list predicate L are both constructive. Furthermore, inductive predicates are closed under projection (Σ_i), product ($\times$) and co-product ($+$), so in the extended language associated with $\mathcal{L}$ constructive formulas are closed under disjunction, conjunction and existential quantifier. So constructive formulas contain all Σ_1-formulas of first-order logic. The sequent $\Rightarrow \exists z Sx(ss0)z$ is constructive, but the sequent $Ny \Rightarrow \exists z Sxyz$ is not.

We also remark that a cut-free proof of a constructive sequent is necessarily well-founded, since it would be impossible for an infinite branch to have a progressive thread. In that sense, if a computation (cut elimination procedure) produces a result (i.e. a cut-free proof) then it terminates in a finite number of steps and produces a finite result.

3 Types and terms

We now introduce the basic type theory on which our computational interpretation by Herbrand schemes will be built.

3.1 Extended terms and substitutions

The first category of types comprises just two *ground* types:

- ι, the type of *individuals*.
- Σ, the type of *substitution stacks*.

Formation rules for terms of the above ground types are displayed in Figure 3. We will shortly introduce further term constructors of which one, function application, provides additional terms of either type above. However, the specific terms derivable from the the three rules of fig. 3 represent canonical terms of type ι and Σ.

Definition 7. A *substitution stack* is any term σ for which the judgement $\sigma : \Sigma$ is derivable using *only* the typing judgements in fig. 3. Likewise, an *extended $\mathcal{L}$-term* is any term t for which the judgement $t : \iota$ is derivable using *only* the typing judgements in fig. 3.

$$\frac{}{\bot : \Sigma} \qquad \frac{\sigma : \Sigma \quad s : \iota \quad \alpha \in \mathsf{EV}}{[s/\alpha]\sigma : \Sigma} \qquad \frac{t \text{ an } \mathscr{L}\text{-term} \quad \sigma : \Sigma}{t \cdot \sigma : \iota}$$

Figure 4: Typing rules for substitution stacks and individuals.

Thus, substitution stacks are finite lists of the form $[s_n/\alpha_n] \cdots [s_1/\alpha_1]\bot$ where α_i is an eigenvariable and $s_i : \iota$ for each $1 \leq i \leq n$, and extended $\mathcal{L}$-terms are $\mathcal{L}$-terms composed with a substitution stack. The $\mathcal{L}$-terms are thus identified with extended $\mathcal{L}$-terms $t \cdot \bot$. Given substitution stacks σ and τ, we write $\sigma\tau$ as shorthand for the substitution stack

$$\sigma\tau := [s_n/\alpha_n] \cdots [s_1/\alpha_1]\tau \qquad \text{where } \sigma = [s_n/\alpha_n] \cdots [s_1/\alpha_1]\bot.$$

Substitution stacks and extended $\mathcal{L}$-terms denote, respectively, substitutions and $\mathcal{L}$-terms. We now define this interpretation, starting with generalising the notion of substitution.

Definition 8. A *substitution* is a partial function $\theta \colon \mathsf{Var} \to \mathsf{Terms}(\mathcal{L})$ from individual variables to $\mathcal{L}$-terms. The unique substitution with empty domain is denoted $\emptyset$. Application of a substitution θ to an $\mathcal{L}$-term t is denoted $t[\theta]$ and defined in the expected way:

$$x[\theta] = \begin{cases} \theta(x), & \text{if } x \in \mathsf{dom}\,\theta, \\ x, & \text{otherwise,} \end{cases} \qquad f(t_1, \dots, t_n)[\theta] = f(t_1[\theta], \dots, t_n[\theta]).$$

Two substitutions θ and θ' can be combined to form a substitution $\theta + \theta'$ with domain $\mathsf{dom}\,\theta \cup \mathsf{dom}\,\theta'$ which applies the substitution θ if on variables in its domain and θ' otherwise:

$$(\theta + \theta')(\alpha) = \begin{cases} \theta(\alpha), & \text{if } \alpha \in \mathsf{dom}\,\theta, \\ \theta'(\alpha), & \text{otherwise.} \end{cases}$$

The operation $\theta, \theta' \mapsto \theta + \theta'$ need not be commutative if $\mathsf{dom}\,\theta \cap \mathsf{dom}\,\theta' \neq \emptyset$. However, if θ and θ' have disjoint domains, then $\theta + \theta' = \theta' + \theta$. When there is no cause for confusion, we write $[t/\alpha]$ for the unique substitution with singleton domain $\{\alpha\}$ and codomain $\{t\}$. Thus, $[t/\alpha] + \theta$ denotes the substitution mapping α to t and $\beta (\neq \alpha)$ to $\theta(\beta)$.

We can now define the *value* of an extended individual term and of a substitution stack, which is to be an $\mathcal{L}$-term and a substitution respectively.

Definition 9. The *value* of an extended individual term v and stack σ is the $\mathcal{L}$-term $\mathsf{Val}(v)$ and substitution $\mathsf{Val}(\sigma)$ respectively defined by mutual recursion:

$$\begin{aligned} \mathsf{Val}(\bot) &= \emptyset \\ \mathsf{Val}([u/\alpha]\sigma) &= [\mathsf{Val}(u)/\alpha] + \mathsf{Val}(\sigma) \end{aligned} \qquad \mathsf{Val}(t \cdot \sigma) = t[\mathsf{Val}(\sigma)]$$

Proposition 1. *For all $\mathcal{L}$-terms t and substitution stacks σ, τ we have* $\mathsf{Val}(\sigma\tau) = \mathsf{Val}(\sigma) + \mathsf{Val}(\tau)$.

Proof. By induction on the length of the stack σ. □

3.2 Basic type constructors

Our underlying type theory is given by the typing/ formation rules presented here. We have a null type $\square$ with the following typing rule:

$$\frac{}{\varepsilon : \square}$$

Types are closed under formation of products and function types. The basic typing rules are:

$$\frac{t : U \to V \quad s : U}{ts : V} \qquad \frac{t : U \quad s : V}{\langle t, s \rangle : U \times V}$$

We also allow products $\prod_{i \in I} V_i$ and co-products $\coprod_{i \in I} V_i$ indexed by finite sets. Typing rules for $I = \{i_0, \ldots, i_{n-1}\}$ are as follows:

$$\frac{t_0 : V_0 \quad \cdots \quad t_{n-1} : V_{n-1}}{\langle t_0, \ldots, t_{n-1} \rangle : \prod_{i \in I} V_i} \qquad \frac{i \in I \quad t : V_i}{\langle i, t \rangle : \coprod_{i \in I} V_i}$$

Furthermore, we have function composition:

$$\frac{t : U \to V \quad s : V \to W}{s \circ t : U \to W}$$

Input and output types We have a primitive type associated with each inductive predicate P, denoted $[P]$ (output type) and $\langle P \rangle$ (input type). We lift this to an association of types with formulas by: $[A] = \langle A \rangle = \square$ for a literal A, $[P\vec{t}] = [P]$ and $\langle P\vec{t} \rangle = \langle P \rangle$ where P is an inductive predicate, and dually $[\neg P\vec{t}] = \langle P \rangle$ and $\langle \neg P\vec{t} \rangle = [P]$. We also have a primitive type $\mathbb{S}$ which is the substitution stack type. Finally, we have a *ground type* ι, the type of individuals.

Note that we haven't explicitly *defined* the input and output types, but rather take them as primitive. Instead, we will have function terms used to explicitly link these types to each other.

Constructors and destructors Given an inductive predicate P we let $\mathcal{R}(P)$ denote the set of production rules for P. For each production r of the form:

$$\frac{A_0 \quad \cdots \quad A_{k-1}}{P(\vec{x}, \vec{t}(\vec{x}, \vec{\alpha}))}$$

where $|\vec{\alpha}| = m$, we associate a *constructor* κ_P^r of type:

$$\kappa_P^r : \iota^m \times [A_0] \times \ldots \times [A_{k-1}] \to [P]$$

For each index $i < k$, let $D(r, i)$ denote the type:

$$(\iota^m \times [A_0] \times \ldots \times [A_{i-1}] \times [A_{i+1}] \times \ldots \times [A_{k-1}]) \to \langle A_i \rangle$$

We define the type of the *destructor* δ_P associated with P, by setting:

$$\delta_P : \left(\prod_{r \in \mathcal{R}(P) \ \& \ i < \mathsf{ar}(r)} D(r, i) \right) \to \langle P \rangle$$

For example, the 'list of natural numbers' predicate has two constructors and a destructor with the following types:

$$\kappa_L^0 : [L]$$

$$\kappa_L^1 : (\iota \times \iota \times [N] \times [L]) \to [L]$$

$$\delta_L : \Big(\big((\iota \times \iota \times [L]) \to \langle N \rangle\big) \times \big((\iota \times \iota \times [N]) \to \langle L \rangle\big) \Big) \to \langle L \rangle$$

The destructor essentially tells us that to provide counter-evidence to some item being a list, we need two functions. Intuitively, one function takes as argument a pair of individuals from which the item can be constructed, together with evidence that the first individual is a natural number, and returns counter-evidence to the second individual being a list. The second function similarly takes as arguments two individuals and evidence that the second one is a list, and returns counter-evidence to the first individual being a natural number.

We also include the 'null' constructor $\perp_{[A]}$ and destructor $\perp_{\langle A \rangle}$ for each formula, with the typing axioms:

$$\frac{}{\perp_{[A]} : [A]} \qquad \frac{}{\perp_{\langle A \rangle} : \langle A \rangle}$$

Peirce operator and choice For each formula A we also associate two special constructors $\mathtt{p}_A$ and $\|_A$ which are typed by the rules:

$$\frac{}{\mathtt{p}_A : (\langle A \rangle \to [A]) \to [A]} \qquad \frac{s : [A] \quad t : [A]}{s \,\|_A\, t : [A]}$$

The symbol $\mathtt{p}$ denotes 'Peirce', as the type of these operators is essentially Peirce's Law:

$$((A \to B) \to A) \to A,$$

or rather its particular instance $(\neg A \to A) \to A$. Note that by definition we have:

$$\mathtt{p}_{\neg A} : ([A] \to \langle A \rangle) \to \langle A \rangle.$$

The term $t \,\|_A\, s$ can be read as 't or s', and corresponds to non-deterministic choice between potential witnesses for A. We will usually drop the subscripts from $\|_A$ and $\mathtt{p}_A$ when clear from context.

Together, the Peirce and choice operators are what allow us to provide computational content to the contraction rule.

4 Herbrand schemes

In this section we describe the recursion schemes associated with sequent calculus proofs. Our aim is to define the recursion scheme $\mathscr{H}(\pi)$ associated with a given sequent calculus proof π. We assume some familiarity with higher-order recursion schemes. For background on recursion schemes see [16]. We will use a variant of recursion schemes involving *pattern matching*, as in [2].

In the present setting, we define a higher-order recursion scheme very generally to be a structure $\mathscr{S} = (\mathscr{N}, \mathscr{T}, \mathscr{V}, \mathscr{R}, \mathsf{S})$ consisting of the following data:

- A set $\mathscr{N}$ of typed *non-terminal symbols*,

- A set $\mathscr{T}$ of typed *terminal symbols*,
- A set $\mathscr{V}$ of typed *variables*,
- A set $\mathscr{R}$ of *rewrite*, or *production*, *rules*,
- A distinguished *start symbol* $\mathsf{S} \in \mathscr{N}$.

Let T be the set of well-typed terms in $\mathscr{N} \cup \mathscr{T} \cup \mathscr{V}$. Formally, a rewrite rule in $\mathscr{R}$ is a pair $(t, s) \in T \times T$ were s and t are of the same type. An *instance* of the rule (t_0, t_1) is a pair $(t_0[\sigma], t_1[\sigma])$ where σ is a type-preserving substitution from variables to T. We say that t_0 *one-step rewrites* to t_1 if (t_0, t_1) is an instance of a rewrite rule in $\mathscr{R}$. We write this as $t_0 \longrightarrow^1 t_1$. We write $t_0 \longrightarrow t_1$ and say that t_0 rewrites to t_1 if the pair (t_0, t_1) is in the reflexive, transitive closure of the one-step rewrite relation. The *language* $\mathscr{L}(\mathscr{S})$ of a recursion scheme $\mathscr{S}$ with start symbol S is the set of terms t containing no variables or non-terminals such that $\mathsf{S} \longrightarrow t$.

Of course, not every recursion scheme in the sense above corresponds to a reasonable model of computation. At the very least one should require the set of rewrite rules to be recursive as well as certain constraints on the form of the first term in the pair $(t, s) \in \mathscr{R}$. Indeed, frequently the rewrites of a higher-order recursion scheme are required to be of the shape:

$$\mathsf{F} x_0 \cdots x_{n-1} \longrightarrow t$$

where F is a non-terminal, x_0, …, x_{n-1} are pairwise distinct variables of appropriate type and t is a term containing no variables other than the x_i.

The more general definition allows for context sensitive rewrites. While the recursion schemes that we associate with proofs require more general rewrites than immediately above, these will be still be heavily constrain in comparison to the definition above. In particular, the only context sensitivity utilised in Herbrand schemes rewrites is *pattern matching.* Rewrites in the recursion schemes that follow will have the general form

$$\mathsf{F} t_0 \cdots t_{n-1} \longrightarrow t$$

where $t_i = \mathsf{f}_i x_{i,0} \cdots x_{i,k_i}$ is a term constructor with k_i the associated arity, and all the $x_{i,j}$ are pairwise distinct. Thus, some rewrite rules will depend not only on the outermost non-terminal but also the shape of its arguments.

Note that we also permit the sets of non-terminals and terminals to be infinite. This is just a technical convenience; rather than assigning separate sets of non-terminals and terminals to each proof, it will be simpler to have fixed sets of non-terminals and terminals with fixed rewrite rules. Essentially, there is a single infinite 'universal' recursion scheme $\mathscr{H}$ with no start symbol, and each individual scheme $\mathscr{H}(\pi)$ is specified by a start symbol and its rewrite rule. In practice, the recursion scheme associated with a proof will always be equivalent to one using only finitely many non-terminals, terminals and variables.

The remainder of this section defines the Herbrand scheme $\mathscr{H}(\hat{\pi})$ associated to a proof $\hat{\pi}$. The terminals of $\mathscr{H}(\hat{\pi})$ comprise all symbols in the type system introduced in Section 3.

The non-terminals of $\mathscr{H}(\hat{\pi})$ comprise the following symbols:

- A *start symbol* $\mathsf{S}_{\hat{\pi}} : [A]$.

- A *proof non-terminal* $\mathsf{F}^\pi_i : \Sigma \to [\Lambda \Rightarrow_i \Pi]$ for each subproof $\pi \vdash \Lambda \Rightarrow \Pi$ of $\hat{\pi}$ and $i < |\Lambda \Rightarrow \Pi|$. These non-terminals compute evidence for the i-th formula occurrence in $\Lambda \Rightarrow \Pi$ from counter-evidence for the remaining formula occurrences in the sequent.
- *Extractor* non-terminals $\mathsf{E}_B : [B] \to [B]$ for each formula B that occurs in $\hat{\pi}$.
- Certain *helper* non-terminals used to express function composition and combinators and to simulate specific cases of λ-abstraction.

Note that the inclusion of a proof non-terminal for every subproof does not contradict our recursion schemes being finite in practice, provided that the proof of which we associate the recursion scheme is regular and that we do not distinguish between isomorphic subproofs. The remainder of this section presents the rewrite rules for the above non-terminals, starting with the helper functions.

4.1 Helper functions

Each of these non-terminals is assigned a single (deterministic) production rule that simulates a particular aspect of λ-abstraction over the underlying type system: composition, exchange, redundancy and substitution formation. By omitting abstraction as a formal constructor, we avoid the need to accommodate β-reduction alongside production rules and to reason about arbitrary λ-abstractions that cannot be simulated by the recursion scheme. Indeed, the particular λ-abstractions expressed by these non-terminals are all *sub-linear* in the sense that they express abstractions $\lambda x\, t$ with at most one occurrence of x in t.

For all types U, V, W and $\vec{V} = V_1, \dots, V_n$, the following non-terminals are included in $\mathscr{H}(\pi)$ with associated production rule:

$$\begin{array}{ll}
\circ : (V \to W) \to (U \to V) \to U \to W & \mathsf{Sbs}_\alpha : \Sigma \to \iota \to \Sigma \quad \text{for each } \alpha \in \mathsf{FV} \\
(x \circ y)z \coloneqq \circ xyz \longrightarrow x(yz) & \mathsf{Sbs}_\alpha\, xy \longrightarrow [y/\alpha]x \\[1ex]
\mathsf{A}^n : (U \to \vec{V} \to W) \to \vec{V} \to U \to W & \mathsf{K} : U \to V \to U \\
\mathsf{A}^n\, w\vec{x}z \longrightarrow wz\vec{x} \quad \text{where } |\vec{x}| = n & \mathsf{K}\, xy \longrightarrow x
\end{array}$$

Despite foregoing λ-abstraction at the formal level, it nonetheless provides a convenient notation for expressing terms constructed from the helper non-terminals. Thus, in the sequel we will more often use the language of λ-calculus than the above non-terminals. Except where stated otherwise, such notation will be strictly confined to constructions that are expressible via the above symbols and other terms/non-terminals.

Here are some examples of this notation:

$$\begin{array}{rcl}
\lambda v.\, \mathsf{F}\sigma v\vec{x} & \text{expands to} & \mathsf{A}^{|\vec{x}|}(\mathsf{F}\sigma)\vec{x} \\
\lambda v.\, \mathsf{F}([v/\alpha]\sigma])\vec{x} & \text{expands to} & (\mathsf{A}^{|\vec{x}|}\mathsf{F}\vec{x}) \circ (\mathsf{Sbs}_\alpha\sigma) \\
\lambda v.\, \mathsf{F}\sigma\vec{w}\vec{x}(z(\mathsf{G}\sigma\vec{w}v\vec{x}\vec{y}))\vec{y} & \text{expands to} & (\mathsf{A}^{|\vec{y}|}(\mathsf{F}\sigma\vec{w}\vec{x})\vec{y}) \circ (z \circ (\mathsf{A}^{|\vec{x}\vec{y}|}(\mathsf{G}\sigma\vec{w})\vec{x}\vec{y}))
\end{array}$$

The K-combinator is helpful to express empty λ-abstractions, for example if $t : U \to V$ and $x : W$ then we can express $\lambda x.t : W \to U \to V$ as $t \circ \mathsf{A}^0\mathsf{K}$, since:

$$\begin{aligned}(t \circ \mathsf{A}^0\mathsf{K})wu &\longrightarrow t(\mathsf{A}^0\mathsf{K}wu)\\ &\longrightarrow t(\mathsf{K}uw)\\ &\longrightarrow tu\end{aligned}$$

For each $\mathcal{L}$-formula A we associate the non-terminal C_A, representing 'generic evidence' for A. If A is a literal, we add the rewrite rule:

$$\mathsf{C}_A \longrightarrow \varepsilon$$

If A is of the form $P\vec{t}$ for an inductive predicate P, we add the rewrite rule:

$$\mathsf{C}_A \longrightarrow \kappa_P^0(\vec{c}_0, \mathsf{C}_{A_0^0}, \dots, \mathsf{C}_{A^0_{k_0-1}}) \parallel \cdots \parallel \kappa_P^{m-1}(\vec{c}_{m-1}, \mathsf{C}_{A_0^{m-1}}, \dots, \mathsf{C}_{A^{m-1}_{k_{m-1}-1}})$$

where P has m production rules and the i-th production rule has k_i premisses $A_0^i, \dots, A^i_{k_i-1}$ and $|\vec{c}_i|$ input variables. For example:

$$\begin{aligned}\mathsf{C}_{Nt} &\longrightarrow \kappa_N^0 \parallel \kappa_N^1(\mathsf{c}, \mathsf{C}_{N\alpha})\\ \mathsf{C}_{Lt} &\longrightarrow \kappa_L^0 \parallel \kappa_L^1(\mathsf{c}, \mathsf{c}, \mathsf{C}_{N\alpha}, \mathsf{C}_{L\beta})\\ \mathsf{C}_{(P\times Q)\vec{u}\vec{v}} &\longrightarrow \kappa^0_{P\times Q}(\mathsf{C}_{P\vec{u}}, \mathsf{C}_{Q\vec{v}})\\ \mathsf{C}_{(P+Q)\vec{u}\vec{v}} &\longrightarrow \kappa^0_{P+Q}(\mathsf{C}_{P\vec{u}}) \parallel \kappa^1_{P+Q}(\mathsf{C}_{Q\vec{v}})\end{aligned}$$

If A is of the form $\neg P\vec{t}$ for an inductive predicate P then we add the rewrite rule below:

$$\mathsf{C}_{\neg P\vec{t}} \longrightarrow \delta_P\big((d_{r,i})_{r\in\mathcal{R}(P)\ \&\ i<\mathsf{ar}(r)}\big)$$

where, for each production rule r of the form:

$$\frac{A_0 \quad \cdots \quad A_{k-1}}{P(\vec{x}, \vec{t}(\vec{x}, \vec{\alpha}))}\,\mathrm{r}$$

where $\mathsf{ar}(r) = k$, and for each index $i < \mathsf{ar}(r)$, the term $d_{r,i}$ is defined to be:

$$\lambda\vec{z}\lambda y_0 \dots y_{i-1}y_{i+1}\dots y_{k-1}.\mathsf{C}_{\neg A_i}$$

4.2 Extractors

Each formula B has associated an *extractor* non-terminal $\mathsf{E}_B : [B] \to [B]$. These non-terminals follow a simple behaviour, recursively eliminating instances of the Peirce operator from their argument and commuting with all other term builders. When encountering a Peirce term the extractor simply evaluates the guarded function on generic evidence for the appropriate type. Their only role is with the start symbol of the Herbrand scheme for which they extract terms suitable for a Herbrand disjunction from arbitrary evidence (or counter-evidence) for a formula. These extractors have the following rewrite rules.

$$\begin{aligned}\mathsf{E}_B\varepsilon &\longrightarrow \varepsilon & \mathsf{E}_B\bot_{[B]} &\longrightarrow \bot_{[B]}\\ \mathsf{E}_{P\vec{t}}\,(\kappa_P^i\vec{u}v_0\dots v_{k-1}) &\longrightarrow \kappa_P^i\vec{u}(\mathsf{E}_{A_0}v_0)\dots(\mathsf{E}_{A_{k-1}}v_{k-1}) & \mathsf{E}_B(x \parallel y) &\longrightarrow \mathsf{E}_Bx \parallel \mathsf{E}_By\\ \mathsf{E}_{\neg P\vec{t}}\,(\delta_P(w)) &\longrightarrow \delta_P(w') & \mathsf{E}_B(\mathsf{p}z) &\longrightarrow \mathsf{E}_B(z\mathsf{C}_{\neg B})\\ \mathsf{E}_{\exists xB}(\mathsf{e}xy) &\longrightarrow \mathsf{e}x(\mathsf{E}_By) & \mathsf{E}_{\neg\neg B}x &\longrightarrow \mathsf{E}_Bx\end{aligned}$$

The above rewrites rely on pattern-matching to determine which rewrite rule is applicable. The rewrites for $\mathsf{E}_{P\vec{t}}$ and $\mathsf{E}_{\neg P\vec{t}}$ require further comment: in the former rewrite rule, we assume that the i-th production rule for P has the k premisses $A_0, \ldots, A_{k-1}$. In the latter rewrite, $w : \langle P \rangle$ is a term of the form:

$$\delta_P(\lambda\vec{z}_0 w_0, \ldots, \lambda\vec{z}_{m-1} w_{m-1})$$

and w' is then:

$$\delta_P(\lambda\vec{z}_0 \mathsf{E}_{\neg B_0} w_0, \ldots, \lambda\vec{z}_{m-1} \mathsf{E}_{\neg B_{m-1}} w_{m-1})$$

where $w_i : \langle B_i \rangle$ for each $i \in \{0, \ldots, m-1\}$.

4.3 Start symbol

The start symbol S_π of $\mathscr{H}(\hat{\pi})$ is associated a single rewrite rule:

$$\mathsf{S}_\pi \longrightarrow \mathsf{E}_A\big(\mathsf{F}_n^{\hat{\pi}} \bot \mathsf{C}_{A_1} \cdots \mathsf{C}_{A_n}\big)$$

where we assume that the endsequent of $\hat{\pi}$ is constructive, i.e. of the form $\Gamma \Rightarrow A$ where A is constructive and $\Gamma = \neg A_1, \ldots, \neg A_n$ where each A_i is constructive.

4.4 Proof non-terminals

Finally, and most importantly, for each proof π with end sequent $\Lambda \Rightarrow \Pi$ and each index $i < |\Lambda \Rightarrow \Pi|$ there is a non-terminal $\mathsf{F}_i^\pi : \Sigma \to [\Lambda \Rightarrow_i \Pi]$. Each such non-terminal has the following distinguished rewrite rules, in order to handle 'undefined' outputs.

$$\mathsf{F}_i^\pi \vec{x} \bot_C \vec{y} \longrightarrow \bot_{\langle A \rangle}$$
$$\mathsf{F}_j^\pi \vec{x} \bot_C \vec{y} \longrightarrow \bot_{[B]}$$

Here, we suppose that i is an index corresponding to a left formula occurrence A, j is an index corresponding to a right formula occurrence B, and that the formula corresponding to the position of the argument $\bot_C$ is either a left or right occurrence of C. The notation $\bot_C$ is supposed to be understood as $\bot_{[C]}$ if C is a left formula occurrence and as $\bot_{\langle C \rangle}$ for a right formula occurrence. Since the distinction between $\bot_{[A]}$ and $\bot_{\langle A \rangle}$ will not affect the evaluation of terms, we will sometimes abuse notation writing the more informal $\bot_A$. This term should always be read as a term of type $[A]$ or $\langle A \rangle$, which should be clear from context.

The remaining production rules associated to F_i^π are of two kinds:

Inference productions A single production rule for F_i^π determined completely by the final inference in π and the immediate subproofs. Production rules associated with structural inference rules are listed in Figure 1. Only one such inference production is associated to each proof non-terminal whose general shape is determined by whether the i-th formula of $\Lambda \Rightarrow \Pi$ is principal. The inference production for F_i^π maps this non-terminal to a term built from terminals and non-terminals F_j^π where π_0 is an immediate subproof of π. In some cases, this term will include constant and helper non-terminals. The production rule for F_i^π depends on pattern-matching only if the final inference is a logical inference and the i is not the index of the principal formula.

Inference	Principal rewrite	Non-principal rewrite
$\dfrac{\overset{\pi_0}{\Gamma \Rightarrow \Delta, A, A}}{\Gamma \Rightarrow \Delta, A}\,\mathsf{cR}$	$\mathsf{F}^{\pi}_{*}\sigma\vec{x} \longrightarrow \mathtt{p}\big(\mathsf{F}^{\pi_0}_{*}\sigma\vec{x}\big) \parallel \mathtt{p}\big(\mathsf{F}^{\pi_0}_{*'}\sigma\vec{x}\big)$	$\mathsf{F}^{\pi}_{i}\sigma\vec{x}z \longrightarrow \mathsf{F}^{\pi_0}_{i}\sigma\vec{x}zz$
$\dfrac{\overset{\pi_0}{A, A, \Gamma \Rightarrow \Delta}}{A, \Gamma \Rightarrow \Delta}\,\mathsf{cL}$	$\mathsf{F}^{\pi}_{*}\sigma\vec{x} \longrightarrow \mathtt{p}\big(\hat{\mathsf{F}}^{\pi_0}_{*}\sigma\vec{x}\big) \parallel \mathtt{p}\big(\hat{\mathsf{F}}^{\pi_0}_{*'}\sigma\vec{x}\big)$	$\mathsf{F}^{\pi}_{i}\sigma z\vec{x} \longrightarrow \mathsf{F}^{\pi_0}_{i+1}\sigma zz\vec{x}$
$\dfrac{\overset{\pi_0}{\Gamma \Rightarrow \Delta, A, \Pi}}{\Gamma \Rightarrow \Delta, \Pi, A}\,\mathsf{pR}$	$\mathsf{F}^{\pi}_{*}\sigma\vec{x}\vec{y} \longrightarrow \mathsf{F}^{\pi_0}_{*}\sigma\vec{x}\vec{y}$	$\mathsf{F}^{\pi}_{i}\sigma\vec{x}z \longrightarrow \begin{cases} \mathsf{F}^{\pi_0}_{i}\sigma\vec{x}\vec{y}, & \text{if } i < \lvert\Gamma\Delta\rvert, \\ \mathsf{F}^{\pi_0}_{i+1}\sigma\vec{x}\vec{y}, & \text{if } i \geq \lvert\Gamma\Delta\rvert. \end{cases}$
$\dfrac{\overset{\pi_0}{\Pi, A, \Gamma \Rightarrow \Delta}}{A, \Pi, \Gamma \Rightarrow \Delta}\,\mathsf{pL}$	$\mathsf{F}^{\pi}_{*}\sigma\vec{y}\vec{x} \longrightarrow \mathsf{F}^{\pi_0}_{*}\sigma\vec{y}\vec{x}$	$\mathsf{F}^{\pi}_{i}\sigma z\vec{y}\vec{x} \longrightarrow \begin{cases} \mathsf{F}^{\pi_0}_{i-1}\sigma\vec{y}z\vec{z}, & \text{if } 0 < i \leq \lvert\Pi\rvert, \\ \mathsf{F}^{\pi_0}_{i}\sigma\vec{y}z\vec{x}, & \text{if } i > \lvert\Pi\rvert. \end{cases}$
$\dfrac{\overset{\pi_0}{\Gamma \Rightarrow \Delta}}{\Gamma \Rightarrow \Delta, A}\,\mathsf{wR}$	$\mathsf{F}^{\pi}_{*}\sigma\vec{x} \longrightarrow \bot_{[A]}$	$\mathsf{F}^{\pi}_{i}\sigma\vec{x}y \longrightarrow \mathsf{F}^{\pi_0}_{i}\sigma\vec{x}$
$\dfrac{\overset{\pi_0}{\Gamma \Rightarrow \Delta}}{A, \Gamma \Rightarrow \Delta}\,\mathsf{wL}$	$\mathsf{F}^{\pi}_{*}\sigma\vec{x} \longrightarrow \bot_{\langle A\rangle}$	$\mathsf{F}^{\pi}_{i}\sigma y\vec{x} \longrightarrow \mathsf{F}^{\pi_0}_{i-1}\sigma\vec{x}$
$\dfrac{\overset{\pi_0}{\Gamma \Rightarrow \Delta, C} \quad \overset{\pi_1}{C, \Lambda \Rightarrow \Pi}}{\Gamma, \Lambda \Rightarrow \Delta, \Pi}\,\mathsf{cut}$	$\mathsf{F}^{\pi}_{i}\sigma\vec{x}_0\vec{y}\vec{x}_1 \longrightarrow \begin{cases} \mathsf{F}^{\pi_0}_{i}\sigma\vec{x}(\mathsf{F}^{\pi_1}_{*}\sigma\vec{y}), & \text{if } i < \lvert\Gamma\rvert, \\ \mathsf{F}^{\pi_0}_{j}\sigma\vec{x}(\mathsf{F}^{\pi_1}_{*}\sigma\vec{y}), & \text{if } i = \lvert\Gamma\Lambda\rvert + j < \lvert\Gamma\Lambda\Delta\rvert, \\ \mathsf{F}^{\pi_1}_{j+1}\sigma(\mathsf{F}^{\pi_0}_{*}\sigma\vec{x})\vec{y}, & \text{if } i = \lvert\Gamma\rvert + j < \lvert\Gamma\Lambda\rvert, \\ \mathsf{F}^{\pi_1}_{j+1}\sigma(\mathsf{F}^{\pi_0}_{*}\sigma\vec{x})\vec{y}, & \text{if } i = \lvert\Gamma\Lambda\Delta\rvert + j. \end{cases}$	

Table 1: Production rules for F^{π}_{i} derived from structural inferences. Subscript $*$ denotes the index of the principal formula of π and the corresponding minor formula/cut formula in the premise(s). In the case of the contraction rules, indices of the two minor formulas are denoted $*$ and $*'$ respectively. In the principal rewrite for cL, $\hat{\mathsf{F}}^{\pi_0}_{i}\sigma\vec{x}$ abbreviates $\lambda v.\, \mathsf{F}^{\pi_0}_{i}\sigma v\vec{x} := \mathsf{A}^{|\vec{x}|}(\mathsf{F}^{\pi_0}_{i}\sigma)\vec{x}$.

Internal productions These are production rules associated every proof non-terminal F^{π}_{i} and applicable whenever at least one argument is guarded by either the Peirce or choice constant $\mathtt{p}x$ or $x \| y$. These productions are listed in Figure 2. Each 'consumes' the matched term constructor and 'reduce' the term F^{π}_{i} to one involving the same non-terminal F^{π}_{i} and, in the case of the Peirce reduction, a non-terminal F^{π}_{j} for $j \neq i$.

Concerning the production rules we observe:

Proposition 2. *All non-terminals of $\mathscr{H}(\pi)$ with the exception of proof non-terminals are deterministic. If two or more productions are applicable to a term $\mathsf{F}^{\pi_0}_{i}\sigma\vec{u} : [A]$ (where $\vec{u}$ is terminal), i.e. there are terms $s \neq t$ such that $\mathsf{F}^{\pi_0}_{i}\sigma\vec{u} : [A] \longrightarrow^1 s$ and $\mathsf{F}^{\pi_0}_{i}\sigma\vec{u} : [A] \longrightarrow^1 t$, then some argument u_i has the form $\mathtt{p}v$ or $v_0 \parallel v_1$.*

In the remainder of the section we give the production rules for proof non-terminals associated with various logical inference rules.

$$\begin{array}{rll}
\mathsf{F}^{\pi}_{i}\sigma\vec{w}\vec{x}(\mathsf{p}z)\vec{y} \longrightarrow \mathsf{p}\big(\lambda v.\, \mathsf{F}^{\pi}_{i}\sigma\vec{w}\vec{x}(z(\mathsf{F}^{\pi}_{i+k}\sigma\vec{w}v\vec{x}\vec{y}))\vec{y}\big) & \text{for } i = |\vec{w}| \text{ and } k = |\vec{x}| \\
\mathsf{F}^{\pi}_{i+k}\sigma\vec{w}(\mathsf{p}z)\vec{x}\vec{y} \longrightarrow \mathsf{p}\big(\lambda v.\, \mathsf{F}^{\pi}_{i+k}\sigma\vec{w}(z(\mathsf{F}^{\pi}_{i}\sigma\vec{w}\vec{x}v\vec{y}))\vec{x}\vec{y}\big) & \text{for } i = |\vec{w}| \text{ and } k = |\vec{x}| > 0 \\
\mathsf{F}^{\pi}_{i}\sigma\vec{w}(x \parallel y)\vec{z} \longrightarrow \mathsf{F}^{\pi}_{i}\sigma\vec{w}x\vec{z} \,\big\|\, \mathsf{F}^{\pi}_{i}\sigma\vec{w}y\vec{z} & \\
\mathsf{F}^{\pi}_{i}\sigma\vec{x}\bot\vec{y} \longrightarrow \bot &
\end{array}$$

Table 2: Production rules associated to Peirce and choice constructors and 'undefined' inputs.

Axiom

Consider an proof consisting of a single axiom:

$$\frac{}{P\vec{s} \Rightarrow P\vec{s}}\,\mathsf{id}$$

Let π denote the proof above and π_0 the trivial subproof comprising the axiom only. For $i \in \{0, 1\}$ there is a proof non-terminal $\mathsf{F}^{\pi_0}_{i}$ of type $\Sigma \to \square \to \square$ (since $\langle P\vec{s}\rangle = [P\vec{s}] = \square$). The rewrite in each case is simply $\mathsf{F}^{\pi_0}_{i}\sigma x \longrightarrow \varepsilon$.

Negation

The first non-trivial logical inference we consider is the right $\neg$-rule. (The rules for the left rule are entirely analogous.) Consider a proof π:

$$\frac{\overset{\pi_0}{A, \Gamma \Rightarrow \Delta}}{\Gamma \Rightarrow \Delta, \neg A}\,\neg\mathsf{R}$$

Let the length of Γ and Δ be m and n respectively. The principal rewrite for this inference is:

$$\mathsf{F}^{\pi}_{m+n}\sigma\vec{x}\vec{y} \longrightarrow \mathsf{F}^{\pi_0}_{0}\sigma\vec{x}\vec{y}$$

where the i-th variable in $\vec{x}$ has the type of evidence for the i-th formula in the sequence $\Gamma\Delta^{\neg}$.

The non-principal rewrites associated to this inference are the following where $z : \langle\neg A\rangle$ and $\vec{x}$ is typed appropriately for the formulas in $\Gamma\Delta^{\neg}$ minus the i-th formula:

$$\mathsf{F}^{\pi_0}_{i}\sigma\vec{x}z \longrightarrow \mathsf{F}^{\pi_1}_{i}\sigma z\vec{x}.$$

Left rule for inductive predicate

Suppose we are given a proof:

$$\frac{\pi_0 : \Delta_0, \Gamma[\vec{v}_0/\vec{z}] \Rightarrow \Theta[\vec{v}_0/\vec{z}] \quad \cdots \quad \pi_{m-1} : \Delta_{m-1}, \Gamma[\vec{v}_{m-1}/\vec{z}] \Rightarrow \Theta[\vec{v}_{m-1}/\vec{z}]}{\pi : P\vec{u}\vec{w}, \Gamma[\vec{w}/\vec{z}] \Rightarrow \Theta[\vec{w}/\vec{z}]}$$

where m is the number of production rules for P. The principal rewrite rule is given by:

$$\mathsf{F}^{\pi}_{*}\sigma\vec{x}\vec{y} \longrightarrow \delta_P\big((d_{r,i})_{r\in\mathcal{R}(P)\ \&\ i<\mathsf{ar}(r)}\big)$$

where, for each production rule r of the form:

$$\frac{A_0 \quad \cdots \quad A_{k-1}}{P(\vec{x}, \vec{t}(\vec{x}, \vec{\alpha}))}\,\mathrm{r}$$

where $\mathsf{ar}(r) = k$, and for each index $i < \mathsf{ar}(r)$, the term $d_{r,i}$ is defined to be:

$$\lambda\vec{z}\lambda y_0 \dots y_{i-1} y_{i+1} \dots y_{k-1}.\, \mathsf{F}_i^{\pi_j}[\vec{z}/\vec{\alpha}]\sigma y_0 \dots y_{i-1} y_{i+1} \dots y_{k-1}\vec{x}.$$

The (pattern-matching) non-principal rewrite rule is given by:

$$\mathsf{F}_i^{\pi}\sigma(\kappa_P^r(\vec{v}, z_0, \dots, z_{k-1}))\vec{x}\vec{y} \longrightarrow \mathsf{F}_{i'}^{\pi_j}[\vec{v}\cdot\sigma/\vec{\alpha}]\sigma z_0 \dots z_{k-1}\vec{x}\vec{y}$$

where r is the j-th production rule for r and its input variables are $\vec{\alpha}$.

As an example, consider a proof using the left rule for the list predicate L:

$$\frac{\pi_1 : \Gamma[\epsilon/x] \Rightarrow \Delta[\epsilon/x] \qquad \pi_2 : N\alpha, L\beta, \Gamma[\alpha^\frown\beta/x] \Rightarrow \Delta[\alpha^\frown\beta/x]}{\pi_0 : Lt, \Gamma[t/x] \Rightarrow \Delta[t/x]}$$

The principal rewrite rule becomes:

$$\begin{aligned}&\mathsf{F}_*^{\pi}\sigma\vec{x}\vec{y}\\ &\quad\longrightarrow \delta_L(\lambda uvw.\mathsf{F}_*^{\pi_1}[u/\alpha][v/\beta]\sigma w\vec{x}\vec{y}, \lambda uvw.\mathsf{F}_{*+1}^{\pi_1}[u/\alpha][v/\beta]\sigma w\vec{x}\vec{y})\end{aligned}$$

The non-principal rewrite rule has the following two cases:

$$\mathsf{F}_i^{\pi}\sigma(\kappa_L^0)\vec{x}\vec{y} \longrightarrow \mathsf{F}_i^{\pi_0}\sigma\vec{x}\vec{y}$$

$$\mathsf{F}_{\pi}^{F} i\sigma(\kappa_L^1(u_0, v_0, u_1, v_1))\vec{x}\vec{y} \longrightarrow \mathsf{F}_{i'}^{\pi_1}[u_0/\alpha][u_1/\beta]\sigma v_0 v_1\vec{x}\vec{y}$$

Right rule for inductive predicate

Suppose we are given a proof:

$$\frac{\pi_0 : \Gamma_0 \Rightarrow \Theta_0, A_0[\vec{u}/\vec{x}, \vec{v}/\vec{\alpha}] \qquad \cdots \qquad \pi_{k-1} : \Gamma_{k-1} \Rightarrow \Theta_{k-1}, A_{k-1}[\vec{u}/\vec{x}, \vec{v}/\vec{\alpha}]}{\pi : \Gamma_0, \dots, \Gamma_{k-1} \Rightarrow \Theta_0, \dots, \Theta_{k-1}, P(\vec{u}, \vec{t}(\vec{u}, \vec{v}))}$$

where the right rule for P is associated with the production rule r of the form:

$$\frac{A_0 \qquad \cdots \qquad A_{k-1}}{P(\vec{x}, \vec{t}(\vec{x}, \vec{\alpha}))}\,\mathrm{r}$$

The principal rewrite rule is:

$$\mathsf{F}_*^{\pi}\sigma\vec{x}_0 \dots \vec{x}_{k-1}\vec{y}_0 \dots \vec{y}_{k-1} \longrightarrow \kappa_P^r(\vec{v}\cdot\sigma, \mathsf{F}_*^{\pi_0}\sigma\vec{x}_0\vec{y}_0, ..., \mathsf{F}_*^{\pi_{k-1}}\sigma\vec{x}_{k-1}\vec{y}_{k-1})$$

Assuming P has m production rules, and the formula occurrence with index i in the conclusion corresponds to a minor formula occurrence with index i' in the j-th premiss, the non-principal rewrite rule is the pattern-matching rule:

$$\begin{aligned}&\mathsf{F}_i^{\pi}\sigma\vec{x}_0 \dots \vec{x}_{k-1}\vec{y}_0 \dots \vec{y}_{k-1}(\delta_P((z_{r',j'})_{r'\in\mathcal{R}(P)\ \&\ j'<\mathsf{ar}(r')}))\\ &\qquad\longrightarrow \mathsf{F}_{i'}^{\pi_j}\sigma\vec{x}_j\vec{y}_j(z_{r,j}(\vec{v}\cdot\sigma, \mathsf{F}_*^{\pi_0}\sigma\vec{x}_0\vec{y}_0, \dots, \mathsf{F}_*^{\pi_{j-1}}\sigma\vec{x}_{j-1}\vec{y}_{j-1},\\ &\qquad\qquad \mathsf{F}_*^{\pi_{j+1}}\sigma\vec{x}_{j-1}\vec{y}_{j+1}, \dots, \mathsf{F}_*^{\pi_{k-1}}\sigma\vec{x}_{k-1}\vec{y}_{k-1}))\end{aligned}$$

As an example consider the following two proofs using right rules for the list predicate:

$$\frac{}{\pi : L\epsilon} \qquad\qquad \frac{\pi_0 : \Gamma_0 \Rightarrow \Delta_0, Nt \qquad \pi_1 : \Gamma_1 \Rightarrow \Delta_1, Ls}{\pi : \Gamma_0, \Gamma_1 \Rightarrow \Delta_0, \Delta_1, L(t^\frown s)}$$

Note that the end sequent of the left proof cannot have any side formulas, since these would have to be split among the premisses, of which there are none. This means we have no non-principal rewrite rules corresponding to this proof. We have the following principal rewrite rule:

$$\mathsf{F}^{\pi}_{*}\sigma \longrightarrow \kappa^0_L$$

For the right proof we have the principal rewrite rule:

$$\mathsf{F}^{\pi}_{*}\sigma\vec{x}_0\vec{x}_1\vec{y}_0\vec{y}_1 \longrightarrow \kappa^1_L(t\cdot\sigma, s\cdot\sigma, \mathsf{F}^{\pi_0}_{*}\sigma\vec{x}_0\vec{y}_0, \mathsf{F}^{\pi_1}_{*}\sigma\vec{x}_1\vec{y}_1)$$

The non-principal rewrite rule is:

$$\mathsf{F}^{\pi}_{i}\sigma\vec{x}_0\vec{x}_1\vec{y}_0\vec{y}_1(\delta_L(z_0,z_1)) \longrightarrow \begin{cases} \mathsf{F}^{\pi_0}_{i'}\sigma\vec{x}_0\vec{y}_0(z_1(t\cdot\sigma, s\cdot\sigma, \mathsf{F}^{\pi_1}_{*}\vec{x}_1\vec{y}_1)), & i \text{ left,} \\ \mathsf{F}^{\pi_1}_{i'}\sigma\vec{x}_1\vec{y}_1(z_0(t\cdot\sigma, s\cdot\sigma, \mathsf{F}^{\pi_0}_{*}\vec{x}_0\vec{y}_0)), & i \text{ right.} \end{cases}$$

4.5 Substitution rule

Proof:

$$\frac{\pi_1 : \Gamma \Rightarrow \Delta}{\pi_0 : \Gamma[\tau] \Rightarrow \Delta[\tau]}$$

The rewrite rule is:

$$\mathsf{F}^{\pi_0}_{i}\sigma\vec{u} \longrightarrow \mathsf{F}^{\pi_1}_{i}\sigma[\tau(x_0)\cdot\sigma/x_0]\ldots[\tau(x_{k-1})\cdot\sigma/x_{k-1}]\vec{u}$$

where $\mathsf{dom}(\tau) = \{x_0,\ldots,x_{k-1}\}$.

4.6 Derived sequent rules and rewrite rules

The connectives defined as constructors for inductive predicates in Section 2.2 come with derived sequent rules, for which we have corresponding derived reduction rules for proof non-terminals. These are presented below.

Right rule $\vee$

This rule is an abbreviation:

$$\frac{\pi_1 : \Gamma \Rightarrow A_j(\vec{u}), \Theta}{\pi_0 : \Gamma \Rightarrow (A_0 \vee A_1)(\vec{u}), \Theta} \quad := \quad \frac{\dfrac{\pi_1 : \Gamma \Rightarrow F_{A_j}\vec{u}, \Theta}{\pi_0' : \Gamma \Rightarrow (F_{A_0} + F_{A_1})\vec{u}\vec{u}, \Theta}}{\pi_0 : \Gamma \Rightarrow (\Lambda\vec{x}.(F_{A_0} + F_{A_1})\vec{x}\vec{x})\vec{u}, \Theta}$$

The derived principal rewrite rule becomes:

$$\mathsf{F}^{\pi_0}_{*}\sigma\vec{y}\vec{z} \longrightarrow \kappa^j_{\vee}(\mathsf{F}^{\pi_1}_{*}\sigma\vec{y}\vec{z})$$

where $\kappa^j_{\vee}(z)$ abbreviates $\kappa_{\Lambda\vec{x}.(F_{A_0}+F_{A_1})\vec{x}\vec{x}}(\kappa^j_{F_{A_0}+F_{A_1}}(z))$

The derived non-principal rewrite rule is:

$$\mathsf{F}^{\pi_0}_{i}\sigma\vec{y}(\delta_{\vee}(z_0,z_1))\vec{z} \longrightarrow \mathsf{F}^{\pi_1}_{i}\sigma\vec{y}z_j\vec{z}$$

where $\delta_{\vee}(z_0,z_1)$ abbreviates $\delta_{\Lambda\vec{x}.(F_{A_0}+F_{A_1})\vec{x}\vec{x}}(\delta_{F_{A_0}+F_{A_1}}(z_0,z_1))$.

Left rule $\vee$

Abbreviation:

$$\frac{\pi_1 : \Gamma, A_0(\vec{u}) \Rightarrow \Theta \qquad \pi_2 : \Gamma, A_1(\vec{u}) \Rightarrow \Theta}{\pi_0 : \Gamma, (A_0 \vee A_1)(\vec{u}) \Rightarrow \Theta} \quad := \quad \cfrac{\cfrac{\pi_1 : \Gamma, F_{A_0}\vec{u} \Rightarrow \Theta \qquad \pi_2 : \Gamma, F_{A_1}\vec{u} \Rightarrow \Theta}{\pi_0' : \Gamma, (F_{A_0} + F_{A_1})\vec{u}\vec{u} \Rightarrow \Theta}}{\pi_0 : \Gamma, (\Lambda\vec{x}.(F_{A_0} + F_{A_1})\vec{x}\vec{x})\vec{u} \Rightarrow \Theta}$$

Principal rewrite rule:

$$\mathsf{F}_*^{\pi_0}\sigma\vec{y} \longrightarrow \delta_\vee(\mathsf{F}_*^{\pi_1}\sigma\vec{y}, \mathsf{F}_*^{\pi_2}\sigma\vec{y})$$

Non-principal rewrites:

$$\mathsf{F}_i^{\pi_0}\sigma\vec{x}(\kappa_\vee^0(z))\vec{y} \longrightarrow \mathsf{F}_i^{\pi_1}\sigma\vec{x}z\vec{y}$$

$$\mathsf{F}_i^{\pi_0}\sigma\vec{x}(\kappa_\vee^1(z))\vec{y} \longrightarrow \mathsf{F}_i^{\pi_2}\sigma\vec{x}z\vec{y}$$

Right rule for $\exists$

Abbreviation:

$$\frac{\pi_1 : \Gamma \Rightarrow A(\vec{u}, t, \vec{v}), \Theta}{\pi_0 : \Gamma \Rightarrow (\exists x A)(\vec{u}, \vec{v}), \Theta} \quad := \quad \frac{\pi_1 : \Gamma \Rightarrow F_A\vec{u}t\vec{v}, \Theta}{\pi_0 : \Gamma \Rightarrow (\Sigma_i F_A)\vec{u}\vec{v}, \Theta}$$

The principal rewrite rule is:

$$\mathsf{F}_*^{\pi_0}\sigma\vec{x}\vec{y} \longrightarrow \kappa_\exists(t \cdot \sigma, \mathsf{F}_*^{\pi_1}\sigma\vec{x}\vec{y})$$

where $\kappa_\exists(z_0, z_1)$ abbreviates $\kappa_{\Sigma_i F_A}(z_0, z_1)$ The non-principal rewrite rule is:

$$\mathsf{F}_j^{\pi_0}\sigma\vec{v}(\delta_\exists f)\vec{w} \longrightarrow \mathsf{F}_j^{\pi_1}\sigma\vec{v}f\vec{w}$$

where $\delta_\exists f$ abbreviates $\delta_{\Sigma_i F_A}(f)$.

Left rule $\exists$

Abbreviation:

$$\frac{\pi_1 : \Gamma, A(\vec{u}, \alpha, \vec{v}) \Rightarrow \Theta}{\pi_0 : \Gamma, (\exists x A)(\vec{u}, \vec{v}) \Rightarrow \Theta} \quad := \quad \frac{\pi_1 : \Gamma, F_A\vec{u}\alpha\vec{v} \Rightarrow \Theta}{\pi_0 : \Gamma, (\Sigma_i F_A)\vec{u}\vec{v} \Rightarrow \Theta}$$

Derived principal rewrite rule:

$$\mathsf{F}_*^{\pi_0}\sigma\vec{v}\vec{w} \longrightarrow \delta_\exists(\lambda x_0.\mathsf{F}_*^{\pi_1}[x_0/\alpha]\sigma\vec{v}\vec{w})$$

Derived non-principal rule:

$$\mathsf{F}_j^{\pi_0}\sigma\vec{v}(\kappa_\exists(s, t))\vec{w} \longrightarrow \mathsf{F}_j^{\pi_1}[s/\alpha]\sigma\vec{v}t\vec{w}$$

Right rule $\wedge$

Abbreviation:

$$\frac{\pi_1 :\ \Gamma_0 \Rightarrow A(\vec{u}), \Theta_0 \qquad \pi_2 :\ \Gamma_1 \Rightarrow B(\vec{u}), \Theta_1}{\pi_0 :\ \Gamma_0, \Gamma_1 \Rightarrow (A \wedge B)(\vec{u}), \Theta_0, \Theta_1} \quad := \quad \frac{\dfrac{\pi_1 :\ \Gamma_0 \Rightarrow F_A\vec{u}, \Theta_0 \qquad \pi_2 :\ \Gamma_1 \Rightarrow F_B\vec{u}, \Theta_1}{\pi_0' :\ \Gamma_0, \Gamma_1 \Rightarrow (F_A \times F_B)\vec{u}\vec{u}, \Theta_0, \Theta_1}}{\pi_0 :\ \Gamma_0, \Gamma_1 \Rightarrow (\Lambda\vec{x}.(F_A \times F_B)\vec{x}\vec{x})\vec{u}, \Theta_0, \Theta_1}$$

Principal rewrite rule:

$$\mathsf{F}_*^{\pi_0}\sigma\vec{x}_0\vec{x}_1\vec{y}_0\vec{y}_1 \longrightarrow \kappa_\wedge(\mathsf{F}_*^{\pi_1}\sigma\vec{x}_0\vec{y}_0, \mathsf{F}_*^{\pi_2}\sigma\vec{x}_1\vec{y}_1)$$

where $\kappa_\wedge(z_0, z_1)$ abbreviates $\kappa_{\Lambda\vec{x}.(F_A\times F_B)\vec{x}\vec{x}}(\kappa_{F_A\times F_B}(z_0, z_1))$. Non-principal rewrite rule:

$$\mathsf{F}_i^{\pi_0}\sigma\vec{x}_0\vec{x}_1(\delta_\wedge(f, g))\vec{y}_0\vec{y}_1 \longrightarrow \begin{cases} \mathsf{F}_{i'}^{\pi_1}\sigma\vec{x}_0(f(\mathsf{F}_0^{\pi_2}\sigma\vec{x}_1\vec{y}_1))\vec{y}_0 & i \text{ left index} \\ \mathsf{F}_{i'}^{\pi_2}\sigma\vec{x}_1(g(\mathsf{F}_0^{\pi_1}\sigma\vec{x}_0\vec{y}_0))\vec{y}_1 & i \text{ right index} \end{cases}$$

where $\delta_\wedge(f, g)$ abbreviates $\delta_{\Lambda\vec{x}.(F_A\times F_B)\vec{x}\vec{x}}(\delta_{F_A\times F_B}(f, g))$.

Left rule $\wedge$

Abbreviation:

$$\frac{\pi_1 :\ \Gamma, A(\vec{u}), B(\vec{u}) \Rightarrow \Theta}{\pi_0 :\ \Gamma, (A \wedge B)(\vec{u}) \Rightarrow \Theta} \quad := \quad \frac{\dfrac{\pi_1 :\ \Gamma, F_A\vec{u}, F_B\vec{u} \Rightarrow \Theta}{\pi_0' :\ \Gamma, (F_A \times F_B)\vec{u}\vec{u} \Rightarrow \Theta}}{\pi_0 :\ \Gamma, (\Lambda\vec{x}.(F_A \times F_B)\vec{x}\vec{x})\vec{u} \Rightarrow \Theta}$$

Derived principal rewrite rule:

$$\mathsf{F}_*^{\pi_0}\sigma\vec{x}\vec{y} \longrightarrow \delta_\wedge(\lambda z_0.\mathsf{F}_*^{\pi_1}\sigma\vec{x}z\vec{y}, \lambda z_1.\mathsf{F}_{*+1}^{\pi_1}\sigma\vec{x}z_1\vec{y})$$

Derived non-principal rule:

$$\mathsf{F}_i^{\pi_0}\sigma\vec{u}\kappa_\wedge(s, t)\vec{v} \longrightarrow \mathsf{F}_{i'}^{\pi_1}\sigma\vec{u}st\vec{v}$$

Here $i' = i$ if $i < *$ and $i' = i + 1$ if $i > *$.

5 Languages

While Herbrand schemes provide a computational interpretation to any proof, we will pay special attention to proofs with a constructive end-sequent. For these proofs, we can derive a final *value* of the computation, represented by terms involving non-deterministic choices, analogous to how Herbrand sets can be extracted from proofs of Σ_1 end sequents in first-order logic. Thus the 'denotational semantics' of Herbrand schemes associated with arbitrary proofs will be examined indirectly, in terms of how they contribute to the values of proofs of constructive end sequents in which they feature via applications of the cut-rule.

5.1 Constructive terms and Herbrand expansions

We begin by setting up a grammar for the terms we expect to derive as values of computations.

Definition 10. The *constructive terms* are generated by the following grammar:

$$t, t_0, \ldots, t_{n-1} := \varepsilon \mid \bot_A \mid (t \parallel t) \mid \kappa^i_P(\vec{u}, t_0, \ldots, t_{n-1})$$

here, $\vec{u}$ ranges over tuples of extended individual terms, A ranges over constructive formulas and P is an inductive predicate for which we assume that the i-th production rule has n premisses and $|\vec{u}|$ input variables.

The *simple constructive terms* are generated by the following grammar:

$$t, t_0, \ldots, t_{n-1} := \varepsilon \mid \bot_A \mid \kappa^i_P(\vec{u}, t_0, \ldots, t_{n-1})$$

where $\vec{u}$ ranges over individual terms.

Definition 11. We extend the definition of the *value* $\mathsf{Val}(u)$ of an extended individual term u to define the value $\mathbf{Val}(t)$ of an arbitrary constructive term as follows. First, we define $\mathbf{Val}(u) = \{\mathsf{Val}(u)\}$ for an extended individual term u. The recursive definition is then:

- $\mathbf{Val}(\bot_A) := \emptyset$
- $\mathbf{Val}(\varepsilon) := \varepsilon$
- $\mathbf{Val}(s \parallel t) := \mathbf{Val}(s) \cup \mathbf{Val}(t)$
- $\mathbf{Val}(\kappa^i_P(\vec{s}, t_0, \ldots t_{n-1})) := \{\kappa^i_P(\mathsf{Val}(\vec{s}), t'_0, \ldots, t'_{n-1}) \mid t'_j \in \mathbf{Val}(t_j)\}$

Proposition 3. *For every constructive term t, the set $\mathbf{Val}(t)$ is a finite set of simple constructive terms.*

We can now define the notions of *subsumption* and *equivalence* of terms.

Definition 12. An n-ary *context* $C[z_0, \ldots, z_{n-1}]$ is a term in which $z_0, \ldots, z_{n-1}$ are n distinguished free (typed) variables. Given terms t, s of the same type, we say that t *subsumes* s, written $s \sqsubseteq t$, if for every unary context $C[z]$, whenever $C[s] \longrightarrow s'$ for some constructive term s' we have that $C[t] \longrightarrow t'$ for some constructive term t' such that $\mathbf{Val}(s') = \mathbf{Val}(t')$. We write $t \equiv s$ and say that t and s are equivalent if $t \sqsubseteq s$ and $s \sqsubseteq t$.

The following proposition is immediate:

Proposition 4. *If $t \longrightarrow s$ then $s \sqsubseteq t$.*

Next we define a notion of Herbrand expansion from constructive terms.

Proposition 5. *Every constructive term t is of type $[A]$ for some constructive formula A.*

Proof. Straightforward induction. □

Definition 13. We define the relation $W(M, V, t, A)$ (read: 't witnesses A in the model M relative to the assignment V') to be the smallest relation between models, assignments, simple constructive terms and formulas satisfying the following clauses:

- If $M, V \vDash A$ and A is a literal, then $W(M, V, \varepsilon, A)$.
- Suppose that the i-th production rule for the inductive predicate P is:

  $$\frac{B_1 \quad \cdots \quad B_k}{P(\vec{x}, \vec{t}(\vec{x}, \vec{y}))}$$

  with parameters $\vec{x}$ and input variables $\vec{y}$. If $w_1, \ldots, w_k$ are simple constructive terms such that:

  - $W(M, V, w_i, A_i[\vec{u}/\vec{x}, \vec{v}/\vec{y}])$ for each $i \in \{1, \ldots, k\}$, and
  - $[\![\vec{s}]\!]^V_M = [\![\vec{t}(\vec{u}, \vec{v})]\!]^V_M$,

  then $W(M, V, \kappa^i_P(\vec{v}, w_1, \ldots, w_k), P\vec{s})$.

Definition 14 (Herbrand expansion)**.** Given a constructive sequent S of the form $\Gamma \Rightarrow A$ and a constructive term $t : [A]$, we say that t is a *Herbrand expansion* of the sequent S if for every model M and assignment V such that $M, V \vDash \Gamma$, there is some term $u \in \mathbf{Val}(t)$ such that $W(M, V, u, A)$.

5.2 Clean terms and stacks

This section contains some technical definitions and observations concerning substitution stacks, which will facilitate our investigation of languages associated with proofs via Herband schemes later on.

Definition 15. The *bound variables* of a substitution stack σ and extended individual term $t \cdot \sigma$ is the set $\mathsf{BV}(\sigma)$ and $\mathsf{BV}(t \cdot \sigma)$ respectively, given by

$$\mathsf{BV}(\bot) = \emptyset \qquad \mathsf{BV}(\sigma[u/\alpha]) = \mathsf{BV}(\sigma) \cup \{\alpha\} \qquad \mathsf{BV}(t \cdot \sigma) = \mathsf{BV}(\sigma)$$

The *free variables* of an individual term t is the set $\mathsf{FV}(t)$ of eigenvariables occurring in t. For a substitution stack σ and extended individual term $t \cdot \sigma$, the free variables are defined by mutual recursion:

$$\begin{aligned} \mathsf{FV}(\bot) &= \emptyset \\ \mathsf{FV}(\sigma[u/\alpha]) &= (\mathsf{FV}(u) \setminus \mathsf{BV}(\sigma)) \cup \mathsf{FV}(\sigma) \\ \mathsf{FV}(t \cdot \sigma) &= (\mathsf{FV}(t) \setminus \mathsf{BV}(\sigma)) \cup \mathsf{FV}(\sigma) \end{aligned}$$

Definition 16. An extended term t is *clean* if every extended individual subterm $s \cdot \sigma$ of t is such that $\mathsf{FV}(s) \subseteq \mathsf{BV}(\sigma)$. A substitution stack σ is clean if every extended individual subterm of σ is clean.

A few basic observations about clean substitution stacks follow below.

Proposition 6. *If σ is a clean substitution stack then $FV(\sigma) = \emptyset$.*

Proposition 7. *Let π be a clean proof, let S be the start symbol of $\mathcal{H}(\pi)$ and let α be an eigenvariable that appears in the end sequent of a subproof π_0. If $\mathsf{S} \longrightarrow t$ and t has a subterm of the form $\mathsf{F}^{\pi_0}_i \sigma \vec{u}$, then $\alpha \in \mathsf{BV}(\sigma)$.*

Proof. Just check that all rewrites for rule applications in π preserve this property. $\square$

Proposition 8. *Let π be a clean proof and S the start symbol of $\mathcal{H}(\pi)$, and suppose $\mathsf{S} \longrightarrow t$. Then every extended individual term and every stack appearing in t is clean.*

Proof. See Appendix A. □

Our main task in this subsection is to establish a connection between regular substitution stacks and actual substitutions performed on regular derivations:

Proposition 9. *Let π be any clean proof and τ any substitution stack. Then:*

$$\mathsf{F}_i^{\pi} \sigma\tau\vec{u} \equiv \mathsf{F}_i^{\pi[\mathsf{Val}(\sigma)]}\tau\vec{u}$$

provided $\sigma\tau$ is a clean substitution stack.

Proof. See Appendix B. □

6 Example: pigeonhole principle

In this section we study an example, extracting witnesses for the pigeonhole principle. Herbrand schemes for first-order logic were used in [2] to study a proof of the pigeonhole principle in the form: 'for infinitely many pigeons and at most two pigeonholes, there must be two pigeons inhabiting the same pigeonhole'. The proof studied there used a cut on the infinite pigeonhole principle: under the same assumptions, infinitely many pigeons must inhabit the same pigeonhole.

Here, we approach the formula from a different perspective, instead proving by induction: 'for every natural number n, for infinitely many pigeons and at most n pigeonholes, there must be two pigeons inhabiting the same pigeonhole'. We then derive the statement for $n = 2$ via a cut on a canonical proof that 2 is a natural number. The non-wellfounded proof of the general statement also uses a cut, corresponding to what would be the induction step in a wellfounded proof of the same statment.

We work in a vocabulary with a constant 0, unary function symbols f, s and ordinary binary predicates $<, \leq, =$. The inductive predicates considered are the natural number predicate N, together with those inductive predicates needed to express quantifiers and connectives. We use the following abbreviations:

- $Bxy := \neg\exists z(x \leq z \wedge y < fz)$
- $F := \exists x \exists y(x < y \wedge fx = fy)$

The formula Bxy intuitively says that the function f is bounded by y above x. The formula F is the existential formula we want to extract witnesses for; if we think of fx as 'the pigeonhole that pigeon x inhabits' then it expresses that two pigeons inhabit the same pigeonhole. We let Γ be a set of Π_1-formulas (i.e. negations of constructive formulas built up from existential quantifiers, conjunction and disjunction) containing sufficiently many assumptions to prove some the expected basic laws for the non-logical predicates. For example, the sequent $\Gamma \Rightarrow x < y, y < x, x = y$ should be derivable.

Below we present the proof that we will analyze, where we have omitted some subproofs involving only propositional reasoning.

$$\dfrac{\dfrac{\dfrac{}{2.\ N0}}{1.\ Ns0} \qquad \dfrac{\overset{\pi'}{a.\ Bu0, \Gamma \Rightarrow F} \qquad \overset{\pi''}{4.\ N\alpha, Bu(s\alpha), \Gamma \Rightarrow F}}{3.\ Nv, Buv, \Gamma \Rightarrow F\ \dagger}}{0.\ Bus0, \Gamma \Rightarrow F}\,\mathsf{app}$$

π':

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{h.\ \Gamma \Rightarrow u \leq u \wedge 0 < fu, u \leq su \wedge 0 < fsu, u < su \wedge fu = fsu)}}{g.\ \Gamma \Rightarrow u \leq u \wedge 0 < fu, \exists zu \leq z \wedge 0 < z, fzu < su \wedge fu = fsu)}}{f.\ \Gamma \Rightarrow \exists zu \leq z \wedge 0 < fz, \exists zu \leq z \wedge 0 < fz, u < su \wedge fu = fsu)}}{e.\ Bu0, \Gamma \Rightarrow \exists zu \leq z \wedge 0 < fz, u < su \wedge fu = fsu)}}{d.\ Bu0, Bu0, \Gamma \Rightarrow u < su \wedge fu = fsu)}}{c.\ Bu0, \Gamma \Rightarrow u < su \wedge fu = fsu)}}{b.\ Bu0, \Gamma \Rightarrow \exists y(u < y \wedge fu = fy)}}{a.\ Bu0, \Gamma \Rightarrow F}$$

π'':

$$\dfrac{\dfrac{7.\ N\alpha, B\beta\alpha, \Gamma \Rightarrow F\ \dagger}{6.\ N\alpha, \exists yBy\alpha, \Gamma \Rightarrow F} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{17.\ s\gamma_0 \leq \gamma_1 \wedge \alpha < f\gamma_1, u \leq \gamma_0 \wedge \alpha < f\gamma_0, Bu(s\alpha), \Gamma \Rightarrow \gamma_0 < \gamma_1 \wedge f\gamma_0 = f\gamma_1}}{16.\ s\gamma_0 \leq \gamma_1 \wedge \alpha < f\gamma_1, u \leq \gamma_0 \wedge \alpha < f\gamma_0, Bu(s\alpha), \Gamma \Rightarrow \exists y(\gamma_0 < y \wedge f\gamma_0 = fy)}}{15.\ s\gamma_0 \leq \gamma_1 \wedge \alpha < f\gamma_1, u \leq \gamma_0 \wedge \alpha < f\gamma_0, Bu(s\alpha), \Gamma \Rightarrow F}}{14.\ \exists z(s\gamma_0 \leq z \wedge \alpha < fz), u \leq \gamma_0 \wedge \alpha < f\gamma_0, Bu(s\alpha), \Gamma \Rightarrow B(s\gamma_0)\alpha, F}}{13.\ u \leq \gamma_0 \wedge \alpha < f\gamma_0, Bu(s\alpha), \Gamma \Rightarrow B(s\gamma_0)\alpha, F}}{12.\ u \leq \gamma_0 \wedge \alpha < f\gamma_0, Bu(s\alpha), \Gamma \Rightarrow \exists yBy\alpha, F}}{11.\ \exists z(u \leq z \wedge \alpha < fz), Bu(s\alpha), \Gamma \Rightarrow \exists yBy\alpha, F}}{10.\ Bu(s\alpha), \Gamma \Rightarrow Bu\alpha, \exists yBy\alpha, F}}{9.\ Bu(s\alpha), \Gamma \Rightarrow \exists yBy\alpha, \exists yBy\alpha, F}}{8.\ Bu(s\alpha), \Gamma \Rightarrow \exists yBy\alpha, F}}{\dfrac{5.\ N\alpha, Bu(s\alpha), \Gamma, \Gamma \Rightarrow F, F}{4.\ N\alpha, Bu(s\alpha), \Gamma \Rightarrow F}}\,\mathsf{cut}$$

We now show how to derive a Herbrand set for F from the Herbrand scheme associated with this proof. For readability we will suppress subscripts indicating the types of extractor non-terminals, constant non-terminals and the Peirce operator. We assume that $|\Gamma| = l$, and we use the symbol ? to hide any terms that will not be relevant to the witness extraction. We start with the main calculation from the start symbol S:

$$\begin{aligned}
\mathsf{S} &\longrightarrow \mathsf{E}(\mathsf{F}^0_{l+1}\bot\mathsf{C}\vec{\mathsf{C}}) \\
&\longrightarrow \mathsf{E}(\mathsf{F}^3_{l+2}[s0/v](\mathsf{F}^1_0\bot)\mathsf{C}\vec{\mathsf{C}}) \\
&\longrightarrow \mathsf{E}(\mathsf{F}^3_{l+2}[s0/v](\kappa^1_N(0, \kappa^0_N))\mathsf{C}\vec{\mathsf{C}}) \\
&\longrightarrow \mathsf{E}(\mathsf{F}^4_{l+2}[0/\alpha][s0/v](\kappa^0_N)\mathsf{C}\vec{\mathsf{C}}) \\
&\longrightarrow \mathsf{E}(\mathsf{F}^5_{2l+2}[0/\alpha][s0/v](\kappa^0_N)\mathsf{C}\vec{\mathsf{C}}\vec{\mathsf{C}}\mathsf{C}) \parallel \mathsf{E}(\mathsf{F}^5_{2l+3}[0/\alpha][s0/v](\kappa^0_N)\mathsf{C}\vec{\mathsf{C}}\vec{\mathsf{C}}\mathsf{C})
\end{aligned}$$

Put $\sigma := [0/\alpha][s0/v]$. We continue with the left terms as Continuation 1:

$$\mathsf{E}(\mathsf{F}^5_{2l+2}\sigma(\kappa^0_N)\mathsf{C}\vec{\mathsf{C}}\vec{\mathsf{C}}\mathsf{C})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{F}^8_{l+1}\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{p}(\lambda z.\mathsf{F}^9_{l+1}\sigma\mathsf{C}\vec{\mathsf{C}}z\mathsf{C}) \parallel \mathsf{p}(\lambda z.\mathsf{F}^9_{l+2}\sigma\mathsf{C}\vec{\mathsf{C}}z\mathsf{C}))\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{p}(\lambda z.\mathsf{F}^9_{l+1}\sigma\mathsf{C}\vec{\mathsf{C}}z\mathsf{C}))\vec{\mathsf{C}}) \parallel \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{p}(\lambda z.\mathsf{F}^9_{l+2}\sigma\mathsf{C}\vec{\mathsf{C}}z\mathsf{C}))\vec{\mathsf{C}})$$

Continuation 1.a:

$$\mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{p}(\lambda z.\mathsf{F}^9_{l+1}\sigma\mathsf{C}\vec{\mathsf{C}}z\mathsf{C}))\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{F}^9_{l+1}\sigma\mathsf{C}\vec{\mathsf{C}}(\mathsf{F}^6_1\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\kappa_\exists(u\cdot\sigma, ?))\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^7_{l+2}[u\cdot\sigma/\beta]\sigma(\kappa^0_N)(?)\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^3_{l+2}[\beta/u][\alpha/v][u\cdot\sigma/\beta]\sigma(\kappa^0_N)(?)\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^a_{l+1}[\beta/u][\alpha/v][u\cdot\sigma/\beta]\sigma(?)\vec{\mathsf{C}})$$
$$\longrightarrow \kappa_\exists(u\cdot[\beta/u][\alpha/v][u\cdot\sigma/\beta]\sigma, \kappa_\exists(su\cdot[\beta/u][\alpha/v][u\cdot\sigma/\beta]\sigma, ?))$$
$$\equiv \kappa_\exists(u, \kappa_\exists(su, ?))$$

Continuation 1.b:

$$\mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{p}(\lambda z.\mathsf{F}^9_{l+2}\sigma\mathsf{C}\mathsf{C}\vec{\mathsf{C}}z\mathsf{C}))\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{F}^9_{l+2}\sigma\mathsf{C}\vec{\mathsf{C}}(\mathsf{F}^6_1\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{F}^9_{l+2}\sigma\mathsf{C}\vec{\mathsf{C}}(\delta_\exists(\lambda z.\mathsf{F}^7_1[z/\beta]\sigma(\kappa^0_N)\vec{y}\mathsf{C}))\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{F}^{10}_{l+2}\sigma\mathsf{C}\vec{\mathsf{C}}(\mathsf{F}^7_1[u\cdot\sigma/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{F}^{10}_{l+2}\sigma\mathsf{C}\vec{\mathsf{C}}(\mathsf{F}^3_1[\beta/u][\alpha/v][u\cdot\sigma/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{F}^{10}_{l+2}\sigma\mathsf{C}\vec{\mathsf{C}}(\mathsf{F}^a_0[\beta/u][\alpha/v][u\cdot\sigma/\beta]\sigma\vec{\mathsf{C}}\mathsf{C})\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{F}^{10}_{l+2}\sigma\mathsf{C}\vec{\mathsf{C}}(\kappa_\exists(u\cdot[\beta/u][\alpha/v][u\cdot\sigma/\beta]\sigma, ?) \parallel \kappa_\exists(su\cdot[\beta/u][\alpha/v][u\cdot\sigma/\beta]\sigma, ?)\mathsf{C})\vec{\mathsf{C}})$$
$$\equiv \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{F}^{10}_{l+2}\sigma\mathsf{C}\vec{\mathsf{C}}(\kappa_\exists(u, ?) \parallel \kappa_\exists(su, ?)\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{F}^{11}_{l+2}\sigma(\kappa_\exists(u, ?) \parallel \kappa_\exists(su, ?)\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\mathsf{F}^{12}_{l+2}[u/\gamma_0]\sigma?\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C} \parallel \mathsf{F}^{12}_{l+2}[su/\gamma_0]\sigma?\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\kappa_\exists(s\gamma_0\cdot[u/\gamma_0]\sigma, \mathsf{F}^{13}_{l+2}[u/\gamma_0]\sigma?\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C}) \parallel \kappa_\exists(s\gamma_0\cdot[su/\gamma_0]\sigma, \mathsf{F}^{13}_{l+2}[su/\gamma_0]\sigma?\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\kappa_\exists(s\gamma_0\cdot[u/\gamma_0]\sigma, \mathsf{F}^{13}_{l+2}[u/\gamma_0]\sigma?\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C}))\vec{\mathsf{C}}) \parallel \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\kappa_\exists(s\gamma_0\cdot[su/\gamma_0]\sigma, \mathsf{F}^{13}_{l+2}[su/\gamma_0]\sigma?\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C}))\vec{\mathsf{C}})$$
$$\equiv \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\kappa_\exists(su, \mathsf{F}^{13}_{l+2}[u/\gamma_0]\sigma?\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C}))\vec{\mathsf{C}}) \parallel \mathsf{E}(\mathsf{F}^6_{l+2}\sigma(\kappa^0_N)(\kappa_\exists(ssu, \mathsf{F}^{13}_{l+2}[su/\gamma_0]\sigma?\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C}))\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^7_{l+2}[su/\beta]\sigma(\kappa^0_N)(\mathsf{F}^{13}_{l+2}[u/\gamma_0]\sigma?\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C})\vec{\mathsf{C}}) \parallel \mathsf{E}(\mathsf{F}^7_{l+2}[ssu/\beta]\sigma(\kappa^0_N)(\mathsf{F}^{13}_{l+2}[su/\gamma_0]\sigma?\sigma\vec{\mathsf{C}}\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^3_{l+2}[\alpha/v][\beta/u][su/\beta]\sigma(\kappa^0_N)(\mathsf{F}^{13}_{l+2}[u/\gamma_0]\sigma?\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C})\vec{\mathsf{C}}) \parallel \mathsf{E}(\mathsf{F}^3_{l+2}[\alpha/v][\beta/u][ssu/\beta]\sigma(\kappa^0_N)(\mathsf{F}^{13}_{l+2}[su/\gamma_0]\sigma?\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \mathsf{E}(\mathsf{F}^a_{l+1}[\alpha/v][\beta/u][su/\beta]\sigma(\mathsf{F}^{13}_{l+2}[u/\gamma_0]\sigma?\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C})\vec{\mathsf{C}}) \parallel \mathsf{E}(\mathsf{F}^a_{l+1}[\alpha/v][\beta/u][ssu/\beta]\sigma(\mathsf{F}^{13}_{l+2}[su/\gamma_0]\sigma?\sigma\mathsf{C}\vec{\mathsf{C}}\mathsf{C})\vec{\mathsf{C}})$$
$$\longrightarrow \kappa_\exists(u\cdot[\alpha/v][\beta/u][su/\beta]\sigma, \kappa_\exists(su\cdot[\alpha/v][\beta/u][su/\beta]\sigma, ?)) \parallel \kappa_\exists(u\cdot[\alpha/v][\beta/u][ssu/\beta]\sigma, \kappa_\exists(su\cdot[\alpha/v][\beta/u][ssu/\beta]\sigma, ?))$$
$$\equiv \kappa_\exists(su, \kappa_\exists(ssu, ?)) \parallel \kappa_\exists(ssu, \kappa_\exists(sssu, ?))$$

Returning to the main calculation, we rewrite the right term as Continuation 2:

$$\begin{aligned}
&\mathsf{E}(\mathsf{F}^5_{2l+3}\sigma(\kappa^0_N)\mathsf{C}\vec{\mathsf{C}}\vec{\mathsf{C}}\mathsf{C})\\
&\qquad\longrightarrow \mathsf{E}(\mathsf{F}^8_{l+2}\sigma\mathsf{C}\vec{\mathsf{C}}(\mathsf{F}^6_1\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C}))\\
&\qquad\longrightarrow \mathsf{E}(\mathsf{F}^8_{l+2}\sigma x\vec{y}(\delta_\exists(\lambda w.\mathsf{F}^7_1[w/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\\
&\qquad\longrightarrow \mathsf{E}(\mathsf{F}^9_{l+3}\sigma\mathsf{C}\vec{\mathsf{C}}(\delta_\exists(\lambda w.\mathsf{F}^7_1[w/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})(\delta_\exists(\lambda w.\mathsf{F}^7_1[w/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\\
&\qquad\longrightarrow \mathsf{E}(\mathsf{F}^{10}_{l+3}\sigma\mathsf{C}\vec{\mathsf{C}}(\mathsf{F}^7_1[u\cdot\sigma/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})(\delta_\exists(\lambda w.\mathsf{F}^7_1[w/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\\
&\qquad\longrightarrow \mathsf{E}(\mathsf{F}^{11}_{l+3}\sigma(\mathsf{F}^7_1[u\cdot\sigma/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})\mathsf{C}\vec{\mathsf{C}}(\delta_\exists(\lambda w.\mathsf{F}^7_1[w/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\\
&\qquad\longrightarrow \mathsf{E}(\mathsf{F}^{11}_{l+3}\sigma(\mathsf{F}^3_1[\alpha/v][\beta/u][u\cdot\sigma/\beta]\sigma(\kappa^0_N)\vec{y}\mathsf{C})\mathsf{C}\vec{\mathsf{C}}(\delta_\exists(\lambda w.\mathsf{F}^7_1[w/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\\
&\qquad\longrightarrow \mathsf{E}(\mathsf{F}^{11}_{l+3}\sigma(\mathsf{F}^a_0[\alpha/v][\beta/u][u\cdot\sigma/\beta]\sigma\vec{\mathsf{C}}\mathsf{C})\mathsf{C}\vec{\mathsf{C}}(\delta_\exists(\lambda w.\mathsf{F}^7_1[w/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\\
&\qquad\longrightarrow \mathsf{E}(\mathsf{F}^{11}_{l+3}\sigma(\kappa_\exists(u\cdot[\alpha/v][\beta/u][u\cdot\sigma/\beta]\sigma,?)\parallel\kappa_\exists(su\cdot[\alpha/v][\beta/u][u\cdot\sigma/\beta]\sigma,?)\mathsf{C}\vec{\mathsf{C}}(\delta_\exists(\lambda w.\mathsf{F}^7_1[w/\alpha]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\\
&\qquad\equiv \mathsf{E}(\mathsf{F}^{11}_{l+3}\sigma(\kappa_\exists(u,?)\parallel\kappa_\exists(su,?))\mathsf{C}\vec{\mathsf{C}}(\delta_\exists(\lambda w.\mathsf{F}^7_1[w/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\\
&\qquad\longrightarrow \mathsf{E}(\mathsf{F}^{12}_{l+3}[u/\gamma_0]\sigma?\mathsf{C}\vec{\mathsf{C}}(\delta_\exists(\lambda w.\mathsf{F}^7_1[w/\alpha]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\parallel\mathsf{E}(\mathsf{F}^{12}_{l+3}[su/\gamma_0]\sigma?\mathsf{C}\vec{\mathsf{C}}(\delta_\exists(\lambda w.\mathsf{F}^7_1[w/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\\
&\qquad\longrightarrow \mathsf{E}(\mathsf{F}^{13}_{l+3}[u/\gamma_0]\sigma?\mathsf{C}\vec{y}(\mathsf{F}^7_1[s\gamma_0\cdot[u/\gamma_0]\sigma/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\parallel\mathsf{E}(\mathsf{F}^{13}_{l+3}[su/\gamma_0]\sigma?\mathsf{C}\vec{\mathsf{C}}(\mathsf{F}^7_1[s\gamma_0\cdot[su/\gamma_0]\sigma/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\\
&\qquad\equiv \mathsf{E}(\mathsf{F}^{13}_{l+3}[u/\gamma_0]\sigma?\mathsf{C}\vec{\mathsf{C}}(\mathsf{F}^7_1[su/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\parallel\mathsf{E}(\mathsf{F}^{13}_{l+3}[su/\gamma_0]\sigma?\mathsf{C}\vec{\mathsf{C}}(\mathsf{F}^7_1[ssu/\beta]\sigma(\kappa^0_N)\vec{\mathsf{C}}\mathsf{C})))\\
&\qquad\longrightarrow \mathsf{E}(\mathsf{F}^{13}_{l+3}[u/\gamma_0]\sigma?\mathsf{C}\vec{\mathsf{C}}(\kappa_\exists(u\cdot[\alpha/v][\beta/u][su/\beta]\sigma,?)\parallel\kappa_\exists(su\cdot[\alpha/v][\beta/u][su/\beta]\sigma,?))\\
&\qquad\qquad\parallel\mathsf{E}(\mathsf{F}^{13}_{l+3}[su/\gamma_0]\sigma?\mathsf{C}\vec{\mathsf{C}}(\kappa_\exists(u\cdot[\alpha/v][\beta/u][ssu/\beta]\sigma,?)\parallel\kappa_\exists(su\cdot[\alpha/v][\beta/u][ssu/\beta]\sigma,?))))\\
&\qquad\equiv \mathsf{E}(\mathsf{F}^{13}_{l+3}[u/\gamma_0]\sigma?\mathsf{C}\vec{\mathsf{C}}(\kappa_\exists(su,?)\parallel\kappa_\exists(ssu,?))\parallel\mathsf{E}(\mathsf{F}^{13}_{l+3}[su/\gamma_0]\sigma?\mathsf{C}\vec{\mathsf{C}}(\kappa_\exists(ssu,?)\parallel\kappa_\exists(sssu,?)))\\
&\qquad\longrightarrow \mathsf{E}(\mathsf{F}^{14}_{l+3}[u/\gamma_0]\sigma(\kappa_\exists(su,?)\parallel\kappa_\exists(ssu,?))?\mathsf{C}\vec{\mathsf{C}}\parallel\mathsf{E}(\mathsf{F}^{14}_{l+3}[su/\gamma_0]\sigma(\kappa_\exists(ssu,?)\parallel\kappa_\exists(sssu,?))?\mathsf{C}\vec{\mathsf{C}})\\
&\qquad\longrightarrow \mathsf{E}(\mathsf{F}^{15}_{l+3}[su/\gamma_1][u/\gamma_0]\sigma??\mathsf{C}\vec{\mathsf{C}}\parallel\mathsf{E}(\mathsf{F}^{15}_{l+3}[ssu/\gamma_1][u/\gamma_0]\sigma??\mathsf{C}\vec{\mathsf{C}}\\
&\qquad\qquad\parallel\mathsf{E}(\mathsf{F}^{15}_{l+3}[ssu/\gamma_1][su/\gamma_0]\sigma??\mathsf{C}\vec{\mathsf{C}}\parallel\mathsf{E}(\mathsf{F}^{15}_{l+3}[sssu/\gamma_1][su/\gamma_0]\sigma??\mathsf{C}\vec{\mathsf{C}}\\
&\qquad\longrightarrow \kappa_\exists(\gamma_0\cdot[su/\gamma_1][u/\gamma_0]\sigma,\kappa_\exists(\gamma_1\cdot[su/\gamma_1][u/\gamma_0]\sigma,?))\\
&\qquad\qquad\parallel\kappa_\exists(\gamma_0\cdot[ssu/\gamma_1][u/\gamma_0]\sigma,\kappa_\exists(\gamma_1\cdot[ssu/\gamma_1][u/\gamma_0]\sigma,?))\\
&\qquad\qquad\parallel\kappa_\exists(\gamma_0\cdot[ssu/\gamma_1][su/\gamma_0]\sigma,\kappa_\exists(\gamma_1\cdot[ssu/\gamma_1][su/\gamma_0]\sigma,?))\\
&\qquad\qquad\parallel\kappa_\exists(\gamma_0\cdot[sssu/\gamma_1][su/\gamma_0]\sigma,\kappa_\exists(\gamma_1\cdot[sssu/\gamma_1][su/\gamma_0]\sigma,?))\\
&\qquad\equiv \kappa_\exists(u,\kappa_\exists(su,?))\parallel\kappa_\exists(u,\kappa_\exists(ssu,?))\parallel\kappa_\exists(su,\kappa_\exists(ssu,?))\parallel\kappa_\exists(su,\kappa_\exists(sssu,?))
\end{aligned}$$

Gathering these results up we can now conclude that the computation has extracted the following five pairs of witnesses for the variables x, y:

	x	y
1	u	su
2	su	ssu
3	ssu	$sssu$
4	u	ssu
5	su	$sssu$

7 Multicut analysis

In this section we introduce the multicut rule, which will form the basis of the approach to cut-elimination we will follow.

7.1 Multicut instances

It will be convenient to regard a *formula occurrence* in a sequent formally as a pair (S, i) where S is a sequent and i is an index indicating the i-th formula occurrence in the sequent. We write $p(S, i) = 0$ if (S, i) is a left formula occurrence and $p(S, i) = 1$ if (S, i) is a right formula occurrence. Given a formula occurrence (S, i) we write $S[i]$ for the i-th formula in the sequent S. So we can think of formula occurrences as 'addresses' of formulas appearing in sequents, and the operation $S[-]$ retrieves the actual formula from its address. The *length* $|S|$ of a sequent $S = \Gamma \Rightarrow \Delta$ is $|\Gamma| + |\Delta|$.

Definition 17. A quasi-instance of multicut is a pair $(\mathbb{S}, \bigtriangledown)$ where $\mathbb{S}$ is a non-empty finite multi-set of sequents and $\bigtriangledown$ is a symmetric binary relation over the set of formula occurrences from sequents in $\mathbb{S}$. Given sequents S, U we write $S \bigtriangledown U$ if there are i, j such that $(S, i) \bigtriangledown (U, j)$. The (undirected) graph with vertices $\mathbb{S}$ and edges $\{S, S'\}$ such that $S \bigtriangledown S'$ is called the *cut graph* of the multicut quasi-instance, and denoted $G(\mathbb{S}, \bigtriangledown)$.

A multicut quasi-instance $(\mathbb{S}, \bigtriangledown)$ is called an *instance* of multicut if the following constraints are satisfied:

- The cut graph $G(\mathbb{S}, \bigtriangledown)$ is a tree, i.e. connected and acyclic.
- For each formula occurrence (S, i) there is at most one formula occurrence (U, j) such that $(S, i) \bigtriangledown (U, j)$.
- For each pair of sequents S, U, there is at most one formula occurrence (S, i) and at most one formula occurrence (U, j) such that $(S, i) \bigtriangledown (U, j)$.
- If $(S, i) \bigtriangledown (U, j)$ then:
  - $S[i] = U[j]$, and
  - $p(S, i) = 1 - p(U, j)$.

If there is some (U, j) with $(S, i) \bigtriangledown (U, j)$ then we call (U, j) the *cut companion* of (S, i). A formula occurrence (S, i) is said to be a *cut formula occurrence* of a multicut if it has a cut companion and a *side formula occurrence* otherwise.

If $(\mathbb{S}, \bigtriangledown)$ is a multicut instance, we say that the inference with sequents $\mathbb{S}$ as premises and the associated cut relation $\bigtriangledown$, written in short-hand as:

$$\frac{(\mathbb{S}, \bigtriangledown)}{U}$$

where U is some sequent, is *valid*, if there is a one-to-one map f from formula occurrences (U, i) for $i < l(U)$ to side formula occurrences in the multicut instance. We often speak rather informally of the formula occurrence corresponding to (U, i) in the multicut, by which we mean $f(U, i)$ for some map f which we usually leave implicit.

Definition 18. We say that a formula occurrence (S, i) *directly depends on* a formula occurrence (U, j), written $(U, j) \prec (S, i)$, if there is some $k < l(S)$ with $k \neq i$ and $(U, j) \bigtriangledown (S, k)$. We say that (S, i) depends on (U, j) if $(U, j) \prec^+ (S, i)$, where $\prec^+$ denotes the transitive closure of $\prec$.

Intuitively, (S, i) directly depends on (U, j) if an output for the latter is needed as input for the calculation of an output for the former. As a direct consequence of the definition of a multicut instance, the dependency relation is well-founded.

An alternative formulation of dependency may help to clarify it. Given a multicut instance $(\mathbb{S}, \bigtriangledown)$ and two distinct sequents $S, U \in \mathbb{S}$, there must be a unique sequence of formula occurrences:

$$(S, i_0) \bigtriangledown (V_1, j_n) \neq (V_1, i_n) \bigtriangledown \ldots \bigtriangledown (V_{n-1}, i_{n-1}) \neq (V_{n-1}, j_{n-1}) \bigtriangledown (U, i_n)$$

We write $\mathsf{link}(S, U)$ for the index i_0 and, conversely, $\mathsf{link}(U, S)$ for the index i_n.

The following proposition is proved in the companion paper [1].

Proposition 10. *Given two formula occurrences (S, i) and (U, j) with $S \neq U$, we have $(S, i) \prec^+ (U, j)$ if and only if $i = \mathsf{link}(S, U)$ and $j \neq \mathsf{link}(U, S)$.*

An *extended multicut instance* $(\mathbb{S}, \bigtriangledown, \pi)$ is an instance of multicut together with an assignment π of a multicut-free (but not necessarily cut-free) proof π_S to each sequent $S \in \mathbb{S}$. Given an extended multicut instance we define the *canonical input term* $I(S, i)$ and the *canonical output term* $O(S, i)$ associated with a formula occurrence (S, i) by well-founded induction on $\prec$:

- If (S, i) is a left side formula occurrence of the multicut, then $I(S, i) = \mathsf{C}_{S[i]}$.
- If (S, i) is a right side formula occurrence of the multicut, then $I(S, i) = \mathsf{C}_{\neg S[i]}$.
- If (S, i) has cut companion (U, j) then $I(S, i) = O(U, j)$.
- $O(S, i) = F_i^{\pi_S} \bot I(S, 0) \ldots I(S, i-1) I(S, i+1) \ldots I(S, l(S) - 1)$

Note that this definition is indeed by well-founded induction, as each term $I(S, j)$ for $j \neq i$ is either a constant non-terminal or given recursively as $O(U, k)$ for $(U, k) \bigtriangledown (S, j)$ and hence $(U, k) \prec (S, i)$. We extend the notation π_S for $S \in \mathbb{S}$ by writing also π_{S_0} for the subtree of π_S generated by S_0, if S_0 is a premise of S in π_S.

7.2 Multicut guarded proofs

Definition 19. Let π be any regular proof. The *end piece* of π is the smallest sub-tree π_{ep} of π containing the end sequent and closed under premises of all rules except cut and multicut. Given a sequent S in π_{ep}, define $\pi_{ep}|S$ to be the part of π_{ep} containing all sequents from the end sequent of π up to and including S. We say that π is *multicut guarded* if each leaf of π_{ep} is either an instance of the axiom id, or the conclusion of a nullary inference rule for an inductive predicate, or the conclusion of a multicut rule.

Definition 20. Let π be a multicut guarded regular proof of a constructive end sequent. For each sequent S belonging to π_{ep}, and each index $i < l(S)$, we define the *target term* $T(S, i, \pi)$ by well-founded induction on the descendant relation in the tree π_{ep}. For the base case, where S is a leaf in π_{ep}, there are two possible cases: either S is an axiom, or the conclusion of a nullary right inference rule, or S is the conclusion of a multicut. If S is an axiom then it consists of two occurrences of the same formula, which is a literal. We set $T(S, i, \pi) = \varepsilon$ for $i \in \{0, 1\}$. If S is the conclusion of a nullary inference rule for an inductive predicate P, then since the end sequent is constructive this inference rule must be the right rule associated with some nullary production rule r for the predicate and the sequent contains only the principal formula of the rule. So we set $T(S, 0, \pi) = \kappa_P^r$.

The remaining case is where S is the conclusion of a multicut. We can define an extended multicut instance $(\mathbb{S}, \bigtriangledown, \pi')$ by letting $\mathbb{S}$ consist of the premises of the multicut, with the relation $\bigtriangledown$ given as specified by the multicut, and letting

π' assign to each premise U the corresponding generated sub-proof of π. We define $T(S, i, \pi) = O(S', i')$, where (S', i') is the side formula occurrence of the multicut corresponding to (S, i).

Now suppose S has at least one premiss belonging to π_{ep}. Then S is the conclusion of a rule for an inductive predicate, or a structural rule other than cut. Given that S has n premisses, let $S_0, \ldots, S_{n-1}$ be the premisses of S, with S_0 thus denoting the unique premise if the rule used was unary and S_0, S_1 being the left and right premisses of a binary rule. We define $T(S, i, \pi)$ by a case distinction on the rule used:

Weakening: Put $T(S, *, \pi) = \bot_A$ for the principal index, where the principal formula is A. For other indices we put $T(S, i, \pi) = T(S', i', \pi)$ where (S', i') is the formula occurrence associated with (S, i) via the weakening inference.

Contraction: Put $T(S, *, \pi) = T(S_0, *, \pi) \parallel T(S_0, * + 1, \pi)$ for the principal index. For a non-principal index put $T(S, i, \pi) = T(S_0, i', \pi)$ where i' is the corresponding index in the premise.

Exchange: Assume the exchanged formulas have indices i, j, put $T(S, i, \pi) = T_(S_0, j, \pi)$, $T(S, j, \pi) = T(S_0, i, \pi)$ and $T(S, k, \pi) = T(S_0, k, \pi)$ for $k \notin \{i, j\}$.

Substitution rule: If S is the conclusion of the substitution rule with substitution σ we put $T(S, i, \pi) = T(S_0, i, \pi[\sigma])$.

Left or right ¬-rule: In each case we set $T(S, i, \pi) = T(S_0, i', \pi)$ where (S_0, i') is either the side formula occurrence associated with (S, i), or (S, i) is the principal formula and (S_0, i') the minor formula of the rule application.

Right IP-rule: Assuming the right rule is associated with production rule r for predicate P, for the principal index put:

$$T(S, *, \pi) = \kappa_P(\vec{t}, T(S_0, *_0, \pi), \ldots, T(S_{n-1}, *_{n-1}, \pi))$$

where $\vec{t}$ are the witnesses for the input variables used in the rule application and $*_i$ is the index of the minor formula of the i-th premiss. For a non-principal index i put $T(S, i, \pi) = T(S_j, i', \pi)$ where (S_j, i') is the formula occurrence associated with (S, i).

The case of a left rule for an inductive predicate cannot occur since the end sequent was assumed to be constructive.

For a proof π of a constructive sequent S, which we recall is of the form $\Gamma \Rightarrow A$ with a single constructive formula on the right (and negated constructive formulas on the left), we abbreviate $T(S, i, \pi)$ by $T(\pi)$, where i the index of the right-most formula. Note that target terms are always constructive terms.

Definition 21. The *guarded version* of a regular proof π is the proof π' obtained by replacing every bottom-most cut in π by a multicut with the same premises. We say that the proof π reduces to the proof π^* via multicut reductions if its guarded version does.

Proposition 11. *Let π be a proof with constructive end-sequent S of the form $\Gamma \Rightarrow A$, and let π' be its guarded version. Then:*

$$\mathsf{S}_\pi \sqsupseteq T(\pi')$$

Definition 22. A proof is said to be *essential-cut-free* if every cut formula in a cut or multicut appearing in the proof is a literal.

Note that any essential-cut-free proof is multicut guarded.

Proposition 12. *Let π be an essential-cut-free proof of a constructive end-sequent S. Then $T(\pi)$ is a Herbrand expansion of S.*

Proof. The crucial thing to note is that an essential-cut-free proof of a constructive end-sequent must be a finite (wellfounded) proof, since an infinite branch could satisfy the trace condition required for validity. With this observation in place, the argument proceeds by a straightforward induction on the size of the proof π. □

7.3 Cut reductions and permutations

In this section we introduce reduction and permutation rules for multicuts, and show how these can be simulated by Herbrand schemes. For the structural rules, in particular the non-trivial cases of reduction rules for contraction and weakening on cut formulas, we refer the reader to the companion paper [1] where these cases have already been dealt with. The treatment in the case of non-wellfounded proofs is precisely the same. (Binary cuts are handled by a reduction that simply merges these with a multicut.)

The rules for negation are rather trivial and are omitted. The remaining cases that we need to cover are: a reduction rule for cuts in which some premiss is a conclusion of the substitution rule, a reduction rule for cuts on principal formulas of rules for inductive predicates, and finally, left and right permutation rules for inductive predicate rules with principal formulas occurring as side formulas to the cut.

7.3.1 Reduction of substitution rule

We first consider a reduction rule for mulicuts in which one of the premisses is the conclusion of the substitution rule. Here is a visualization of the reduction:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Sigma \Rightarrow \Pi}}{\Sigma[\tau] \Rightarrow \Pi[\tau]}\,\tau \qquad \cdots}{\Gamma \Rightarrow \Delta}\,\triangledown \quad \Rightarrow \quad \dfrac{\dfrac{\vdots[\tau]}{\Sigma[\tau] \Rightarrow \Pi[\tau]} \qquad \cdots}{\Gamma \Rightarrow \Delta}\,\triangledown$$

For the precise definition, suppose $(\mathbb{S}, \triangledown, \pi)$ is an extended multicut instance in which the sequent S is the conclusion of an instance of $\mathsf{Sub}(\tau)$ for some substitution τ and premiss S_0. We define the reduced multicut instance $(\mathbb{S}^*, \triangledown^*, \pi^*)$ as follows: we set $\mathbb{S}^* = \mathbb{S}$, $\triangledown^* = \triangledown$ and define: $\pi^*_S = \pi_0[\sigma]$ where π_0 is the subproof of π_S generated by S_0. For $U \in \mathbb{S} \setminus \{S\}$ we set $\pi^*_U = \pi_U$.

Given a formula occurrence (U, i) with $U \in \mathbb{S}$ and a formula occurrence (V, j) with $V \in \mathbb{S}^*$, let us denote the canonical output term of (U, i) regarded as a formula occurrence of the multicut instance $(\mathbb{S}, \triangledown, \pi)$ by $O(U, i)$ and the canonical outout term of (V, j) regarded as a formula occurrence of the reduced multicut instance $(\mathbb{S}^*, \triangledown^*, \pi^*)$ by $O^*(V, j)$. We shall continue to follow this notational convention for the other cases below.

Proposition 13. *For each formula occurrence (U, i) with $U \in \mathbb{S}$, we have $O(U, i) \sqsupseteq O^*(U, i)$.*

Proof. This is a fairly straightforward argument, in which we handle the case where $U = S$ by combining the rewrite rule associated with the substitution rule with Proposition 9. □

7.3.2 Reduction on IP-rules

We first define the reduction. Suppose the extended multi-cut instance to be reduced is $(\mathbb{S}, \triangledown, \pi)$, and suppose the principal formula occurrences are (L_0, l) and (R_0, r), where (L_0, l) is a left formula occurrence, (R_0, r) is a right formula occurrence, and $L[l] = R[r] = P(\vec{u}, \vec{t}(\vec{u}, \vec{v}))$. For simplicity we assume that $r = |R_0|$ and $l = 0$. Suppose R_0 is the conclusion of the right rule associated with the p-th production rule for P and the premisses of R_0 are $R'_0, \dots, R'_{m-1}$ with $R_j[r] = A_j[\vec{u}/\vec{x}, \vec{v}/\vec{\alpha}]$ for $j < m$. Suppose the premiss of L_0 associated with the same production rule is L_1 with $L_1[j] = A_j[\vec{u}/\vec{x}]$ for $j < m$ and $L_1[j + m] = L_0[j + 1]$ for $j < r$. We define the reduced extended multi-cut instance $(\mathbb{S}^*, \triangledown^*, \pi^*)$ as follows:

- Replace R_0 by $R'_0, \dots, R'_{m-1}$ and L_0 by L_1. For $j < m$ we set $\pi^*_{R'_j}$ to be the subproof of π_{R_0} generated by R'_j, and $\pi^*_{L_1}$ is obtained by uniformly substituting $\vec{v}$ for $\vec{\alpha}$ in the subproof of L_0 generated by L_1.
- All other sequents in $\mathbb{S}$ are carried over as they are to $\mathbb{S}^*$.
- The relation $\triangledown^*$ is defined as the smallest symmetric relation satisfying the following conditions:
  - $(L_1, j) \triangledown^* (R'_j, r)$ for $j < m$.
  - $(S, i) \triangledown^* (U, j)$ for all formula occurrences $(S, i) \triangledown (U, j)$ with $S, U \notin \{L_0, R_0\}$.
  - If $(S, i) \triangledown (L_0, j)$ and $S \neq R_0$ then $(S, i) \triangledown^* (L_1, m + j)$.
  - If $(S, i) \triangledown (R_0, j)$ and $S \neq L_0$ then $(S, i) \triangledown^* (R_{i'}, j')$, where $(R_{i'}, j')$ is the formula occurrence associated with the side formula occurrence (R_0, j) in the right rule application.

This definition corresponds to a reduction rule for a proof ending with a multi-cut:

$$\frac{(\mathbb{S}, \triangledown, \pi)}{\Gamma \Rightarrow \Delta} \quad \Rightarrow \quad \frac{(\mathbb{S}^*, \triangledown^*, \pi^*)}{\Gamma \Rightarrow \Delta}$$

Visualization:

$$\dfrac{\cdots \quad \dfrac{\overset{\vdots}{\Sigma_0 \Rightarrow \Pi_0, A_0[\vec{u}/\vec{x}, \vec{v}/\vec{\alpha}]} \quad \cdots \quad \overset{\vdots}{\Sigma_{k-1} \Rightarrow \Pi_{m-1}, A_{m-1}[\vec{u}/\vec{x}, \vec{v}/\vec{\alpha}]}}{\Sigma_0, \dots, \Sigma_{m-1} \Rightarrow \Pi_0, \dots, \Pi_{m-1}, P(\vec{u}, \vec{t}(\vec{u}, \vec{v}))} \quad \dfrac{\cdots \quad \overset{\vdots}{A_0[\vec{u}/\vec{x}], \dots, A_{m-1}[\vec{u}/\vec{x}], \Sigma' \Rightarrow \Pi'}}{P(\vec{u}, \vec{t}(\vec{u}, \vec{v})), \Sigma' \Rightarrow \Pi'}}{\Gamma \Rightarrow \Delta} \triangledown$$

$$\Downarrow$$

$$\dfrac{\cdots \quad \begin{matrix}\vdots \\ \Sigma_0 \Rightarrow \Pi_0, A_0[\vec{u}/\vec{x}, \vec{v}/\vec{\alpha}]\end{matrix} \quad \cdots \quad \begin{matrix}\vdots \\ \Sigma_{m-1} \Rightarrow \Pi_{m-1}, A_{m-1}\end{matrix} \quad \begin{matrix}\vdots\ [\vec{v}/\vec{\alpha}] \\ A_0[\vec{u}/\vec{x}, \vec{v}/\vec{\alpha}], \ldots, A_{m-1}[\vec{u}/\vec{x}, \vec{v}/\vec{\alpha}], \Sigma' \Rightarrow \Pi'\end{matrix}}{\Gamma \Rightarrow \Delta}\ \triangledown^*$$

Proposition 14. *Let (S, i) be any side formula occurrence with $S \in \mathbb{S}$.*

1. *If $S = L_0$ then $O(L_0, i) \sqsupset O^*(L_1, m + i)$.*
2. *If $S = R_0$ then $O(R_0, i) \sqsupset O^*(R'_j, i')$ where (R'_j, i') is the formula occurrence associated with (R_0, i) in the right rule application.*
3. *If $S \notin \{L_0, R_0\}$ then $O(S, i) \sqsupset O^*(S, i)$.*

Proof. See Appendix C. □

7.3.3 Right rule permutation

Suppose we are given the extended multi-cut instance $(\mathbb{S}, \triangledown, \pi)$, and suppose the principal formula occurrence of a right rule for predicate P, corresponding to the k-premiss production rule r, is the side formula occurrence (S, p), where the premisses of S are $S_0, \ldots, S_{k-1}$. Suppose S contains m cut formula occurrences $(S, i_0), \ldots, (S, i_{m-1})$. For each $i \in \{i_0, \ldots, i_{m-1}\}$, let $\mathbb{S}_i$ denote:

$$\{V \in \mathbb{S} \mid \mathsf{link}(V, S) = i\}$$

Note that $\mathbb{S}_i$ is disjoint with $\mathbb{S}_j$ whenever $i \neq j$.

For each $j \in \{0, \ldots, k-1\}$ we define an extended multi-cut instance $(\mathbb{S}^*_j, \triangledown^*_j, \pi^*_j)$

- $\mathbb{S}^*_j$ the union of $\{S_j\}$ and all sets $\mathbb{S}_i$ where $i \in \{i_0, \ldots, i_{m-1}\}$ and the formula occurrence (S, i) is associated with some formula occurrence in S_j. Note that in the case where S_j contains no formula occurrences associated with any cut formula occurrences, $\mathbb{S}^*_j = \{S_j\}$.
- Set $\triangledown^*_j$ to be the smallest symmetric relation such that:
  - If $(S, i) \triangledown (U, i')$ and (S, i) is associated with the formula occurrence (S_j, i'') then $(S_j, i'') \triangledown^*_j (U, i')$.
  - If $U, V \in \mathbb{S}_i$ for some i and $(U, i') \triangledown (V, i'')$ then $(U, i') \triangledown^*_j (V, i'')$.
- π^*_j is defined by taking generated subproofs as expected.

This construction corresponds to a cut permutation rule of the following shape: we make the conclusion of the multi-cut instead the conclusion of the right rule of which S was the conclusion, and we make the j-th premiss of this rule the conclusion of the extended multi-cut instance $(\mathbb{S}^*_j, \triangledown^*_j, \pi^*_j)$. Here is an illustration of the cut permutation rule for a concrete case involving the predicate for lists of natural numbers:

$$\dfrac{\dfrac{\begin{matrix}\vdots \\ \Gamma \Rightarrow A, Nu\end{matrix} \quad \begin{matrix}\vdots \\ \Pi \Rightarrow B, Lv\end{matrix}}{\Gamma, \Pi \Rightarrow A, B, L(u^\frown v)} \quad \begin{matrix}\vdots \\ A \Rightarrow \Delta\end{matrix} \quad \begin{matrix}\vdots \\ B \Rightarrow \Theta\end{matrix}}{\Gamma, \Pi \Rightarrow \Delta, \Theta, L(u^\frown v)}\ \triangledown$$

$$\Downarrow$$

$$\dfrac{\dfrac{\overset{\vdots}{\Gamma \Rightarrow A, Nu} \quad \overset{\vdots}{A \Rightarrow \Delta}}{\Gamma \Rightarrow \Delta, Nu}\triangledown \quad \dfrac{\overset{\vdots}{\Pi \Rightarrow B, Lv} \quad \overset{\vdots}{B \Rightarrow \Theta}}{\Pi \Rightarrow \Theta, Lv}\triangledown}{\Gamma, \Pi \Rightarrow \Delta, \Theta, L(u^\frown v)}$$

A second illustration in which the cut formulas are distributed differently among the premisses of the right rule:

$$\dfrac{\dfrac{\overset{\vdots}{\Gamma \Rightarrow Nu} \quad \overset{\vdots}{\Pi \Rightarrow A, B, Lv}}{\Gamma, \Pi \Rightarrow A, B, L(u^\frown v)} \quad \overset{\vdots}{A \Rightarrow \Delta} \quad \overset{\vdots}{B \Rightarrow \Theta}}{\Gamma, \Pi \Rightarrow \Delta, \Theta, L(u^\frown v)}\triangledown$$

$$\Downarrow$$

$$\dfrac{\dfrac{\overset{\vdots}{\Gamma \Rightarrow Nu}}{\Gamma \Rightarrow Nu}\triangledown \quad \dfrac{\overset{\vdots}{\Pi \Rightarrow A, B, Lv} \quad \overset{\vdots}{A \Rightarrow \Delta} \quad \overset{\vdots}{B \Rightarrow \Theta}}{\Pi \Rightarrow \Delta, \Theta, Lv}\triangledown}{\Gamma, \Pi \Rightarrow \Delta, \Theta, L(u^\frown v)}$$

Note how in this case, the sequent $\Gamma \Rightarrow Nu$ becomes the conclusion of a trivial instance of the multi-cut rule with no cut formulas.

Proposition 15. *Given the extended multi-cut instance $(\mathbb{S}, \triangledown, \pi)$ as described above, in which the principal formula occurrence of a right rule for predicate P corresponding to the k-premiss production rule r is the side formula occurrence (S, p), the following holds for any side formula occurrence (U, i):*

- *For $(U, i) = (S, p)$ we have:*

  $$O(S, p) \sqsupseteq \kappa^r_P(\vec{t} \cdot \bot, O^*(S_0, p_0), \ldots, O^*(S_{k-1}, p_{k-1}))$$

  where p_j is the minor formula of premiss S_j and $\vec{t}$ are the witnesses used in the right rule application.

- *For $U = S$ and $i \neq p$ we have $O(S, i) \sqsupseteq O^*(S_j, i')$ where (S_j, i') is the formula occurrence associated with (S, i) via the right rule application.*

- *In all other cases, $O(U, i) \sqsupseteq O^*(U, i)$.*

Proof. See Appendix D. □

8 Main results

In this section we gather our main technical results. In particular, we show that the Herbrand scheme associated with any proof of a constructive end sequent derives a Herbrand expansion for that sequent.

First, we shall use our analysis of cut reductions and permutations to prove the following key result:

Theorem 1 (Simulation). *Let π be any clean, regular and multi-cut guarded proof of a constructive end sequent, and suppose π reduces to π^* via the multi-cut reduction and permutation rules. Then $T(\pi^*) \sqsubseteq T(\pi)$.*

Proof. Suppose π^* is obtained from π by applying any of the multicut reduction or permutation rules. Then π^* is also a multicut guarded regular proof. Furthermore, for any sequent S appearing in π_{ep}, S is in π^*_{ep} also. We show that for any sequent S in π^*_{ep} and for all $i < |S|$ we have $T(S, i, \pi) \sqsupseteq T(S, i, \pi^*)$. The result follows as the special case where S is the end-sequent and i is the index of its unique right-hand formula.

The statement is proved by a leaf-to-root induction in the finite tree π_{ep} (this has to be finite since the end-sequent is constructive). The only non-trivial part of the induction is for the base case, where S is a leaf in π_{ep} and is therefore either an axiom or the conclusion of a multicut. In the former case, $T(S, i, \pi) = T(S, i, \pi^*)$ for each $i < l(S)$. In the latter case, we have an extended multicut instance $(\mathbb{S}, \bigtriangledown, \pi)$ where $\mathbb{S}$ consists of the premisses of S, and we overload the notation to let π denote the assignment to each premise of the multicut the corresponding generated subproof of the proof π. We make a case distinction on what sort of cut reduction was applied to the multicut to obtain π^*, ignoring the trivial case in which some other multicut was reduced.

Many of the reduction and permutation rules have been handled in the companion paper [1], and are treated exactly the same way here. The remaining cases are the reduction rule for an inductive predicate, and the permutation rule for a right sequent rule associated with an inductive predicate. Propositions 14 and 15 are tailor-made to handle these cases, so we leave out the details. □

To obtain Herbrand expansions from this result, we need a cut elimination theorem. Luckily, here we can build on the pre-existing literature on non-wellfounded proof theory.

Theorem 2. *Let π be a proof of a constructive end-sequent. Then π reduces to an essential cut-free proof.*

To prove the cut elimination theorem we show that there is a terminating multicut reduction strategy by reducing to cut elimination of the system $\mu\mathsf{LK}$ of (propositional) classical logic with explicit fixed points linear logic, for which a cut elimination theorem is already known. The formulas in $\mu\mathsf{LK}$ are generated by the grammar

$$\phi, \psi \coloneqq P \mid P^\perp \mid X \mid \phi \vee \psi \mid \phi \wedge \psi \mid \mu X\, \phi \mid \nu X\, \phi$$

Given a formula ϕ we denote by $\phi^\perp$ the simulated negation. Central to theorem 2 is the following corollary of Saurin [20, Theorem 44]:

Theorem. *The multicut reductions for non-wellfounded proofs in $\mu\mathsf{LK}$ are weakly normalising.*

The non-wellfounded proof system for $\mu\mathsf{LK}$ referred to here is a one-sided sequent system with validity of proof trees defined via a trace condition, as we do here. We refer to [20] for the details.

Saurin's cut-elimination argument interprets a propositional proof as a non-wellfounded proof in linear logic and shows that such proofs are closed under fair multicut elimination. The resultant proof will be well-founded (i.e., finite) by the assumption on the end-sequent: Theorem 2 is stated for proofs with constructive end-sequents, which via our translation to $\mu\mathsf{LK}$ corresponds to sequents consisting of formulas in which all fixed-point operators are μ-operators.

A cut-free proof of such a sequent must be wellfounded, since an infinite branch couldn't possibly contain an infinite thread in which a ν-operator is unfolded infinitely many times.

Proof of Theorem 2. Let $\pi \vdash \Gamma \Rightarrow \Delta$ be a proof of a Σ_1 sequent. Stripping π of all first-order content we envisage π as a proof π_P in $\mu\mathsf{LK}$ of a one-sided sequent $\Gamma_P^\perp, \Delta_P$ via a translation that removes all terms for formulas and interprets IPs as fixed points. More precisely, for each formula B and set of inductive variables V is associated a $\mu\mathsf{LK}$ formula B_V defined recursively as follows. For $X \in V$, $(X\vec{s})_V = X$. Otherwise, $(X\vec{s})_V = \mu X \bigvee_{r \in \mathsf{Rules}(X)} \theta_{r,V\cup\{X\}}$ where for each $r \in \mathsf{Rules}(X)$ we define:

$$\theta_{r,V} \coloneqq (B_0)_V \wedge \cdots \wedge (B_{k-1})_V \quad \text{assuming} \quad r = \frac{B_0 \quad \cdots \quad B_{k-1}}{X\vec{w}}$$

The translation is completed by adding the clauses:

$$(P\vec{t})_V = P \quad \text{for } P \text{ an ordinary predicate}$$
$$(\neg A)_V = (A_V)^\perp$$

That is, π_P is the proof in classical propositional logic with fixed-points in which every formula occurrence A in π has been replaced by A_P and instances of left and right rules for inductive predicates are changed appropriately.

It is clear that every multicut reduction $\pi \Rrightarrow \pi^*$ corresponds to a multicut reduction $\pi_P \Rrightarrow^* \pi_P^*$ where $\pi_P^* = (\pi^*)_P$ is the result of applying the above translation to π^*.

All that remains is to observe that if $\pi \Rrightarrow \pi_0 \Rrightarrow \cdots$ is a fair multicut reduction sequence, then the simulating reduction $\pi_P \Rrightarrow^* (\pi_0)_P \Rrightarrow^* \cdots$ can be chosen to be fair. Saurin's theorem implies that the reduction sequence necessarily terminates in a cut-free proof. □

We can now state our main corollary, which shows that we can always derive Herbrand expansions from Herbrand schemes associated with proofs of constructive end sequents.

Corollary. *Let π be a proof of a constructive end sequent $\Gamma \Rightarrow A$ and let $\mathscr{H}(\pi)$ be its Herbrand scheme with start symbol S_π. Then $\mathsf{S}_\pi \longrightarrow t$ for some constructive term t which is a Herbrand expansion of the end sequent.*

Proof. Let π' be the multi-cut guarded version of π. By Proposition 11, $\mathsf{S}_\pi \longrightarrow T(\pi')$. By Theorem 2, π' reduces to an essential-cut-free proof π^*, and by Theorem 1 $T(\pi^*) \sqsubseteq T(\pi')$. By Proposition 12, $T(\pi^*)$ is a Herbrand expansion of the end sequent $\Gamma \Rightarrow A$. Since $T(\pi^*) \sqsubseteq T(\pi')$, $T(\pi') \longrightarrow u$ for some constructive term u with $\mathbf{Val}(u) = \mathbf{Val}(T(\pi^*))$, hence u is also a Herbrand expansion of $\Gamma \Rightarrow A$. Since $\mathsf{S}_\pi \longrightarrow T(\pi') \longrightarrow u$, we are done. □

9 Conclusion

We have presented Herbrand schemes as a computational interpretation of cyclic proofs for classical theories of inductively defined predicates. This interpretation is compositional, in the sense that when proofs are composed via cuts, the corresponding Herbrand schemes are composed by function application. In the special case where the end-sequent of a proof is constructive, Herbrand schemes can be used to directly compute Herbrand expansions. We illustrated this approach with an analysis of a cyclic proof of the pigeonhole principle.

For future work, the connection with cyclic proofs for Gödel's T as presented by Das in [11] is of particular interest as a potential generalisation to the cyclic setting of Gerhardy and Kohlenbach's ([15]) computational interpretation of classical predicate logic via a functional interpretation. Another avenue is to exploit more fully the framework of higher-order recursion schemes. In this paper, we consider languages of recursion schemes to consist of well-founded terms only. In the literature on higher-order recursion schemes, however, the value of a recursion scheme is an infinite non-wellfounded tree, i.e., an infinite term or 'co-term'. Since every inductive predicate P corresponds to a co-inductive predicate $\neg P$, it raises the question of whether one can define a notion of language for Herbrand schemes that allows us to compute languages including infinitary co-terms as witnesses for co-inductive formulas, representing co-inductive datatypes like streams or infinite trees. In the companion paper [1] we sketched an interpretation of Herbrand schemes as strategies in two-player games which we conjecture extends to the infinitary/cyclic setting.

Acknowledgments

This work was supported by the Knut and Alice Wallenberg Foundation [2020.0199], Swedish Research Council [2020-01873, 2017-05111], and Dutch Research Council [OCENW.M20.048].

A Proof of Proposition 8

One needs to check that this invariant is preserved by all rewrites, i.e. we prove the statement by induction on the length of a rewrite sequence from the start symbol. We focus on the key observations here. First, we note that the principal rewrite of a subterm $\mathsf{F}_i^{\pi_0}\sigma\vec{u}$ where π_0 ends with a right rule introduces an extended individual term of the form $t \cdot \sigma$. If an eigenvariable α occurs in t then, by Proposition 7, we then have $\alpha \in \mathsf{BV}(\sigma)$. This shows that $\mathsf{FV}(t) \subseteq \mathsf{BV}(\sigma)$, so the newly introduced term is clean. Non-principal rewrites for right inductive predicate rules are unproblematic since they neither introduce new terms nor modify any stacks.

Non-principal rewrites for left rules of inductive predicates modify the stack, but the only extended terms added to the stack in this way come from the argument of the proof non-terminal corresponding to the principal formula occurrence. So the terms added to the stack are, by the induction hypothesis, clean. Hence such rewrites also preserve the invariant. The principal rewrite for a left inductive predicate rule introduces a subterm of the form $\lambda\vec{z}\mathsf{F}_j^i[\vec{z}/\vec{\alpha}]\sigma\vec{s}$, and since the terms $\vec{z}$ do not contain any eigenvariables the stack $[\vec{z}/\vec{\alpha}]\sigma$ is clean.

Finally, we consider a rewrite corresponding to an instance of the substitution rule:

$$\frac{\pi_1 : \Gamma \Rightarrow \Delta}{\pi_0 : \Gamma[\tau] \Rightarrow \Delta[\tau]}$$

Here we have $\mathsf{F}_i^{\pi_0}\sigma\vec{u} \longrightarrow \mathsf{F}_i^{\pi_1}\sigma[\tau(x_0)\cdot\sigma/x_0]\ldots[\tau(x_{k-1})\cdot\sigma/x_{k-1}]\vec{u}$. By Proposition 7 we get $\mathsf{FV}(\tau(x_j)) \subseteq \mathsf{BV}(\sigma)$ so the new stack is still clean.

B Proof of Proposition 9

We restate proposition 9:

Proposition. *Let π be any regular proof and $\sigma\rho$ a regular substitution stack. Then*

$$\mathsf{F}_i^{\pi}\sigma\rho \equiv \mathsf{F}_i^{\pi[\mathsf{Val}(\sigma)]}\rho.$$

In the companion paper [1] the analogous statement was proved by induction on the height of a proofs. In the present setting of non-wellfounded proofs that approach is not available to us. Instead, we have to work directly with induction on the length of rewrite sequences.

We shall prove the following statements inductively, for all k: given a context $C[z_0,\ldots,z_n]$ where each z_j has the same type as $\mathsf{F}_{i_j}^{\pi_j}\sigma_j$:

Left-to-right If a term of the form $C[\mathsf{F}_{i_0}^{\pi_0}\sigma_0\rho_0,\ldots,\mathsf{F}_{i_n}^{\pi_n}\sigma_n\rho_n]$ rewrites to a constructive term t in k steps then there is a constructive term t' such that $C[\mathsf{F}_{i_0}^{\pi_0[\mathsf{Val}(\sigma_0)]}\rho_0,\ldots,\mathsf{F}_{i_n}^{\pi_n[\mathsf{Val}(\sigma_n)]}\rho_n] \longrightarrow t'$ and $\mathbf{Val}(t) = \mathbf{Val}(t')$.

Right-to-left If a term of the form $C[\mathsf{F}_{i_0}^{\pi_0[\mathsf{Val}(\sigma_0)]}\rho_0,\ldots,\mathsf{F}_{i_n}^{\pi_n[\mathsf{Val}(\sigma_n)]}\rho_n]$ rewrites to a constructive term t in k steps then there is a constructive term t' such that $C[\mathsf{F}_{i_0}^{\pi_0}\sigma_0\rho_0,\ldots,\mathsf{F}_{i_n}^{\pi_n}\sigma_n\rho_n] \longrightarrow t'$ and $\mathbf{Val}(t) = \mathbf{Val}(t')$.

Here, the stacks $\sigma_j\rho_j$ are assumed to be clean. The strategy to handle the induction step of the induction runs as follows: for the left-to-right direction, suppose a term of the form $C[\mathsf{F}_{i_0}^{\pi_0}\sigma_0\rho_0,\ldots,\mathsf{F}_{i_n}^{\pi_n}\sigma_n\rho_n]$ rewrites to a constructive term t in

$k+1$ steps, where the first rewrite of this sequence is $C[\mathsf{F}_{i_0}^{\pi_0}\sigma_0, \ldots, \mathsf{F}_{i_n}^{\pi_n}\sigma_n] \longrightarrow t_0$. We need to show by a case-by-case inspection that t_0 is on the form:

$$D[\mathsf{F}_{i'_{0,0}}^{\pi'_{0,0}}\tau_0\sigma_0\rho_0, \ldots, \mathsf{F}_{i'_{0,m_0}}^{\pi'_{0,m_0}}\tau_0\sigma_0\rho_0, \ldots, \mathsf{F}_{i'_{n,0}}^{\pi'_{n,0}}\tau_n\sigma_n\rho_n, \ldots, \mathsf{F}_{i'_{n,m_n}}^{\pi'_{n,m_n}}\tau_n\sigma_n\rho_n]$$

for some context D, such that the term $C[\mathsf{F}_{i_0}^{\pi_0[\mathsf{Val}(\sigma_0)]}\rho_0, \ldots, \mathsf{F}_{i_n}^{\pi_n[\mathsf{Val}(\sigma_n)]}\rho_n]$ can be rewritten to:

$$D[\mathsf{F}_{i'_{0,0}}^{\pi'_{0,0}[\mathsf{Val}(\sigma_0)]}\tau_0\rho_0, \ldots, \mathsf{F}_{i'_{0,m_0}}^{\pi'_{0,m_0}[\mathsf{Val}(\sigma_0)]}\tau_0\rho_0, \ldots, \mathsf{F}_{i'_{n,0}}^{\pi'_{n,0}[\mathsf{Val}(\sigma_n)]}\tau_n\rho_n, \ldots, \mathsf{F}_{i'_{n,m_n}}^{\pi'_{n,m_n}[\mathsf{Val}(\sigma_n)]}\tau_n\rho_n]$$

It follows directly by the induction hypothesis on k that the latter term rewrites to some t' with $\mathbf{Val}(t) = \mathbf{Val}(t')$, hence this holds for $C[\mathsf{F}_{i_0}^{\pi_0[\mathsf{Val}(\sigma_0)]}\rho_0, \ldots, \mathsf{F}_{i_n}^{\pi_n[\mathsf{Val}(\sigma_n)]}\rho_n]$ as well. The reasoning to prove the right-to-left direction is similar. The proof of the analogous proposition in the companion paper [1] can be rephrased as an induction along these lines, thus avoiding any induction on the height of proof trees, without any substantial change in the reasoning. Furthermore, the proof of Proposition 9 is then essentially the same (although far more tedious). We omit the details.

C Proof of Proposition 14

Let (S, i) be any formula occurrence, either a side formula or a cut formula. We prove the following items by a simultaneous well-founded induction on the dependency relation $\prec$:

1. If $S = L_0$ and $i \neq l$ then $O(L_0, i) \sqsupseteq O^*(L_1, m+i)$.
2. If $S = R_0$ and $i \neq r$ then $O(R_0, i) \sqsupseteq O^*(R'_j, i')$ where (R'_j, i') is the formula occurrence associated with (R_0, i) in the right rule application.
3. If $S \notin \{L_0, R_0\}$ then $O(S, i) \sqsupseteq O^*(S, i)$.

The proposition follows immediately from these statements.

For item (1), we write $O(L_0, i) = \mathsf{F}_i^{\pi_{L_0}}\bot(\mathsf{F}_r^{\pi_{R_0}}\bot\vec{a}_0 \ldots \vec{a}_{m-1}\vec{b}_0 \ldots \vec{b}_{m-1})\vec{c}$. The input for each index $j \neq l$ is either of the form $\mathsf{C}_{[L_0[j]]}$ for a left index j, or of the form $\mathsf{C}_{\langle L_0[j]\rangle}$ for a right index j, or of the form $O(U, k)$ where $(L_0, j) \bigtriangledown (U, k)$. In the latter case, $U \notin \{L_0, R_0\}$ and item (3) of the induction hypothesis gives $O(U, k) \sqsupseteq O^*(U, k)$. Similarly, in the term $\mathsf{F}_r^{\pi_{R_0}}\bot\vec{a}_0 \ldots \vec{a}_{m-1}\vec{b}_0 \ldots \vec{b}_{m-1}$, each input for an index $j \neq r$ is either of the form $\mathsf{C}_{[R_0[j]]}$ for a left index j, or of the form $\mathsf{C}_{\langle R_0[j]\rangle}$ for a right index j, or of the form $O(U, k)$ for some $(U, k) \bigtriangledown (R_0, j)$. In the latter case we have $U \notin \{L_0, R_0\}$ and item (3) of the induction hypothesis gives $O(U, k) \sqsupseteq O^*(U, k)$. Let $\vec{a}^*_0, \ldots, \vec{a}^*_{m-1}, \vec{b}^*_0, \ldots, \vec{b}^*_{m-1}, \vec{c}^*$ be the result of replacing each of these terms of the form $O(U, k)$ by $O^*(U, k)$. We then have:

$$\begin{aligned}
O(L_0, i) &= \mathsf{F}_i^{\pi_{L_0}} \bot (\mathsf{F}_r^{\pi_{R_0}} \bot \vec{a}_0 \dots \vec{a}_{m-1} \vec{b}_0 \dots \vec{b}_{m-1}) \vec{c} \\
&\sqsupseteq \mathsf{F}_i^{\pi_{L_0}} \bot (\kappa_P^p(\vec{v} \cdot \bot, \mathsf{F}_r^{\pi_{R'_0}} \bot \vec{a}_0 \vec{b}_0, \dots, \mathsf{F}_r^{\pi_{R'_{m-1}}} \bot \vec{a}_{m-1} \vec{b}_{m-1})) \vec{c} \\
&\sqsupseteq \mathsf{F}_i^{\pi_{L_1}} [\vec{v} \cdot \bot / \vec{\alpha}] (\mathsf{F}_r^{\pi_{R'_0}} \bot \vec{a}_0 \vec{b}_0) \dots (\mathsf{F}_r^{\pi_{R'_{m-1}}} \bot \vec{a}_{m-1} \vec{b}_{m-1}) \vec{c} \\
&\equiv \mathsf{F}_i^{\pi_{L_1} [\vec{v}/\vec{\alpha}]} \bot (\mathsf{F}_r^{\pi_{R'_0}} \bot \vec{a}_0 \vec{b}_0) \dots (\mathsf{F}_r^{\pi_{R'_{m-1}}} \bot \vec{a}_{m-1} \vec{b}_{m-1}) \vec{c} && \text{Prop. 9} \\
&\sqsupseteq \mathsf{F}_i^{\pi_{L_1} [\vec{v}/\vec{\alpha}]} \bot (\mathsf{F}_r^{\pi_{R'_0}} \bot \vec{a}_0^* \vec{b}_0^*) \dots (\mathsf{F}_r^{\pi_{R'_{m-1}}} \bot \vec{a}_{m-1}^* \vec{b}_{m-1}^*) \vec{c}^* \\
&= O^*(L_1, i)
\end{aligned}$$

as required.

For item (2), we write $O(R_0, i) = \mathsf{F}_i^{\pi_{R_0}} \bot \vec{a}_0 \vec{b}_0 \dots \vec{a}_{m-1} \vec{b}_{m-1} (\mathsf{F}_l^{\pi_{L_0}} \bot \vec{c})$ and let (R'_j, i') be the formula occurrence corresponding to (R_0, i) in the right rule application. The input for each index $j' \neq r$ is either of the form $\mathsf{C}_{\langle R_0[j'] \rangle}$ for a right index, or $\mathsf{C}_{[R_0[j']]}$ for a left index, or of the form $O(U, k)$ where $(R_0, j') \bigtriangledown (U, k)$. In the latter case, $U \notin \{L_0, R_0\}$ and item (3) of the induction hypothesis gives $O(U, k) \sqsupseteq O^*(U, k)$. Similarly, in the term $\mathsf{F}_l^{\pi_{L_0}} \bot \vec{c}$, each input for an index $j' \neq l$ is either of the form $\mathsf{C}_{\langle L_0[j'] \rangle}$ for a right index, or $\mathsf{C}_{[L_0[j']]}$ for a left index, or of the form $O(U, k)$ where $(L_0, j') \bigtriangledown (U, k)$. In the latter case we have $U \notin \{L_0, R_0\}$ and item (3) of the induction hypothesis gives $O(U, k) \sqsupseteq O^*(U, k)$. Let $\vec{a}_0^*, \dots, \vec{a}_{m-1}^*, \vec{b}_0^*, \dots, \vec{b}_{m-1}^*, \vec{c}^*$ be the result of replacing each of these terms of the form $O(U, k)$ by $O^*(U, k)$. We then have:

$$\begin{aligned}
O(R_0, i) &= \mathsf{F}_i^{\pi_{R_0}} \bot \vec{a}_0 \vec{b}_0 \dots \vec{a}_{m-1} \vec{b}_{m-1} (\mathsf{F}_l^{\pi_{L_0}} \bot \vec{c}) \\
&\sqsupseteq \mathsf{F}_i^{\pi_{R_0}} \bot \vec{a}_0 \vec{b}_0 \dots \vec{a}_{m-1} \vec{b}_{m-1} (\delta_P((d_{p',j'})_{p' \in \mathcal{R}(P) \;\&\; j' < \mathsf{ar}(p')})) \\
&\sqsupseteq \mathsf{F}_{i'}^{\pi_{R'_j}} \bot \vec{a}_j \vec{b}_j (d_{p,j}(\vec{v} \cdot \bot, \mathsf{F}_r^{\pi_{R'_0}} \bot \vec{a}_0 \vec{b}_0, \dots, \mathsf{F}_r^{\pi_{R'_{j-1}}} \bot \vec{a}_{j-1} \vec{b}_{j-1}, \\
&\qquad\qquad \mathsf{F}_r^{\pi_{R'_{j+1}}} \bot \vec{a}_{j+1} \vec{b}_{j+1}, \dots, \mathsf{F}_r^{\pi_{R'_{m-1}}} \bot \vec{a}_{m-1} \vec{b}_{m-1}))
\end{aligned}$$

where $d_{p,j} = \lambda \vec{z} \lambda y_0 \dots y_{j-1} y_{j+1} \dots y_{m-1} \mathsf{F}_j^{\pi_{L_1}} [\vec{z}/\vec{\alpha}] y_0 \dots y_{j-1} y_{j+1} \dots y_{m-1} \vec{c}$, so we can continue:

$$\begin{aligned}
&\mathsf{F}_{i'}^{\pi_{R'_j}} \bot \vec{a}_j \vec{b}_j (d_{p,j}(\vec{v} \cdot \bot, \mathsf{F}_r^{\pi_{R'_0}} \bot \vec{a}_0 \vec{b}_0, \dots, \mathsf{F}_r^{\pi_{R'_{j-1}}} \bot \vec{a}_{j-1} \vec{b}_{j-1}, \\
&\qquad\qquad \mathsf{F}_r^{\pi_{R'_{j+1}}} \bot \vec{a}_{j+1} \vec{b}_{j+1}, \dots, \mathsf{F}_r^{\pi_{R'_{m-1}}} \bot \vec{a}_{m-1} \vec{b}_{m-1})) \\
&\quad \sqsupseteq \mathsf{F}_{i'}^{\pi_{R'_j}} \bot \vec{a}_j \vec{b}_j \big(\mathsf{F}_j^{\pi_{L_1}} [\vec{v} \cdot \bot / \vec{\alpha}] (\mathsf{F}_r^{\pi_{R'_0}} \bot \vec{a}_0 \vec{b}_0) \dots (\mathsf{F}_r^{\pi_{R'_{j-1}}} \bot \vec{a}_{j-1} \vec{b}_{j-1}) \\
&\qquad\qquad\qquad (\mathsf{F}_r^{\pi_{R'_{j+1}}} \bot \vec{a}_{j+1} \vec{b}_{j+1}) \dots (\mathsf{F}_r^{\pi_{R'_{m-1}}} \bot \vec{a}_{m-1} \vec{b}_{m-1}) \vec{c}\big) \\
&\quad \equiv \mathsf{F}_{i'}^{\pi_{R'_j}} \bot \vec{a}_j \vec{b}_j (\mathsf{F}_j^{\pi_{L_1} [\vec{v}/\vec{\alpha}]} \bot (\mathsf{F}_r^{\pi_{R'_0}} \bot \vec{a}_0 \vec{b}_0) \dots (\mathsf{F}_r^{\pi_{R'_{j-1}}} \bot \vec{a}_{j-1} \vec{b}_{j-1}) \\
&\qquad\qquad\qquad (\mathsf{F}_r^{\pi_{R'_{j+1}}} \bot \vec{a}_{j+1} \vec{b}_{j+1}) \dots (\mathsf{F}_r^{\pi_{R'_{m-1}}} \bot \vec{a}_{m-1} \vec{b}_{m-1}) \vec{c}) \\
&\quad \sqsupseteq \mathsf{F}_{i'}^{\pi_{R'_j}} \bot \vec{a}_j^* \vec{b}_j^* (\mathsf{F}_j^{\pi_{L_1} [\vec{v}/\vec{\alpha}]} \bot (\mathsf{F}_r^{\pi_{R'_0}} \bot \vec{a}_0^* \vec{b}_0^*) \dots (\mathsf{F}_r^{\pi_{R'_{j-1}}} \bot \vec{a}_{j-1}^* \vec{b}_{j-1}^*) \\
&\qquad\qquad\qquad (\mathsf{F}_r^{\pi_{R'_{j+1}}} \bot \vec{a}_{j+1}^* \vec{b}_{j+1}^*) \dots (\mathsf{F}_r^{\pi_{R'_{m-1}}} \bot \vec{a}_{m-1}^* \vec{b}_{m-1}^*) \vec{c}^*) \\
&\quad = \mathsf{F}_{i'}^{\pi_{R'_j}} \bot \vec{a}_j^* \vec{b}_j^* (O^*(L_1, j)) \\
&\quad = O^*(R'_j, i')
\end{aligned}$$

The equivalence on the third row is by Proposition 9.

For item (3), suppose $S \notin \{L_0, R_0\}$ and write $O(S,i) = F_i^{\pi_S} \bot \vec{a}$. Each input for index $i' \neq i$ is either of the form $\mathsf{C}_{\langle S[i'] \rangle}$ for a right index, or $\mathsf{C}_{[S[i']]}$ for a left index, or of the form $O(U,k)$ where $(S,i') \bigtriangledown (U,k)$. In the latter case, there are three possibilities. If $U \notin \{L_0, R_0\}$ then item (3) of the induction hypothesis gives $O(U,k) \sqsupseteq O^*(U,k)$. If $U = L_0$ then we must have $k \neq l$ since otherwise we would have $(U,k) \bigtriangledown (S,i')$, hence $(S,i') = (R_0, r)$, contradicting our assumption that $S \notin \{L_0, R_0\}$. Item (1) of the induction hypothesis gives $O(L_0,k) \sqsupseteq O^*(L_1,k')$ where k' is the index associated with k via the left rule application. Similarly, if $U = R_0$ then we must have $k \neq r$ and item (2) of the induction hypothesis gives $O(R_0,k) \sqsupseteq O^*(R'_j,k')$ where (R'_j,k') is the formula occurrence associated with (R_0,k) via the right rule application. Let $\vec{a}^*$ be the result of replacing each input term (U,k) for $U \notin \{L_0, R_0\}$ by $O^*(U,k)$, each input term of the form (L_0,k) by $O^*(L_1,k')$ and each input term of the form (R_0,k) by $O^*(R'_j,k')$. Then we have:

$$\begin{aligned} O(S,i) &= \mathsf{F}_i^{\pi_S} \bot \vec{a} \\ &\sqsupseteq \mathsf{F}_i^{\pi_S} \bot \vec{a}^* \\ &= O^*(S,i) \end{aligned}$$

as required.

D Proof of Proposition 15

In order to keep notation readable, we carry out the proof for the special case in which the inference rule to be permuted is the right rule corresponding to one of the production rules of the list predicate. The general case is not different in any interesting respect. We recall that the list predicate has the following two production rules which we, for the moment, refer to by their index in the enumeration:

$$\frac{}{L\epsilon}\,0 \qquad \frac{Nx \quad Ly}{L(x^\frown y)}\,1$$

Now let $(\mathbb{S}, \bigtriangledown, \pi)$ be an extended multi-cut instance, and suppose the side formula occurrence (S,p) is the principal formula occurrence of the right rule associated with production rule (1). This rule has two premisses, S_l, S_r, and we denote the minor formula occurrences of these premisses by (S_l, p_l) and (S_r, p_r) respectively. So $S[p]$ is of the form $L(s^\frown t)$ where $S_l[p_l] = Ns$ and $S_r[p_r] = Lt$. Let $(\mathbb{S}^*_l, \bigtriangledown^*_l, \pi^*_l)$ and $(\mathbb{S}^*_r, \bigtriangledown^*_r, \pi^*_r)$ be the associated reduced extended multi-cut instances. We need to prove the following:

1. $O(S,p) \sqsupseteq \kappa^1_L(s \cdot \bot, O^*(S_l,p_l), O^*(S_r,p_r))$.
2. For $U = S$ and $i \neq p$ we have $O(S,i) \sqsupseteq O^*(S_j,i')$ where (S_j,i') is the formula occurrence associated with (S,i) via the right rule application (so $j \in \{l,r\}$).
3. In all other cases, $O(U,i) \sqsupseteq O^*(U,i)$.

As before we prove this by wellfounded induction on $\prec$. Item (1) is almost immediate from the principal rewrite rule for right IP rules. To ease notation suppose that $O(S,p)$ is of the form $\mathsf{F}^{\pi_S}_p \bot \vec{u}\vec{v}$ where the arguments $\vec{u}$ correspond to formula occurrences associated with the left premiss S_l and arguments $\vec{v}$ are similarly associated with the right premiss S_r. Letting π_l, π_r denote the subproofs of π_S generated by S_l, S_r respectively, we then have:

$$O(s,p) \longrightarrow \kappa^1_L(s \cdot \bot, \mathsf{F}^{\pi_l}_{p_l} \bot \vec{u}, \mathsf{F}^{\pi_r}_{p_r} \bot \vec{v})$$

and it suffices to prove that $\mathsf{F}^{\pi_l}_{p_l} \bot \vec{u} \sqsupseteq O^*(S_l, p_l)$ and $\mathsf{F}^{\pi_r}_{p_r} \bot \vec{v} \sqsupseteq O^*(S_r, p_r)$. For this it suffices in turn to note that $O^(S_l, p_l)$ and $O^*(S_r, p_r)$ are respectively of the form $\mathsf{F}^{\pi_l}_{p_l} \bot \vec{u}^*$ and $\mathsf{F}^{\pi_r}_{p_r} \bot \vec{v}^*$ where the arguments $\vec{u}^*, \vec{v}^*$ are point-wise subsumed by $\vec{u}, \vec{v}$. This is proved by an easy application of the induction hypothesis.

For item (2), i is the index of a non-principal formula and $O(S,i)$ is of the form $\mathsf{F}^{\pi_S}_i \bot \vec{u}\vec{v}(I(S,p))$ where we assume the arguments $\vec{u}$ are associated with the left premiss and arguments $\vec{v}$ are associated with the right premiss. Since we assumed that (S,p) was a side formula occurrence of the multi-cut, $I(S,p) = \mathsf{C}_{\neg L(s \frown t)}$. We now apply the rewrite rule:

$$\mathsf{C}_{\neg L(s \frown t)} \longrightarrow \delta_L(\lambda xy.\lambda z_r.\mathsf{C}_{\neg N s}, \lambda xy.\lambda z_l.\mathsf{C}_{\neg L t})$$

Here, the variables have types: $x, y : \iota$, $z_r : [Lt]$, $z_l : [Ns]$. We can now apply the pattern-matching non-principal rewrite rules; if i is associated with a formula occurrence (S_l, i') in the left premiss we get:

$$\begin{aligned} O(S,i) &= \mathsf{F}^{\pi_S}_i \bot \vec{u}\vec{v}\mathsf{C}_{\neg L(s \frown t)} \\ &\sqsupseteq \mathsf{F}^{\pi_S}_i \bot \vec{u}\vec{v}(\delta_L(\lambda xy.\lambda z_r.\mathsf{C}_{\neg N s}, \lambda xy.\lambda z_l.\mathsf{C}_{\neg L t})) \\ &\sqsupseteq \mathsf{F}^{\pi_l}_{i'} \vec{u}\mathsf{C}_{\neg N s} \end{aligned}$$

Using a similar argument as before, appealing to the induction hypothesis, we can show that $\mathsf{F}^{\pi_l}_{i'} \vec{u}\mathsf{C}_{\neg N s} \sqsupseteq O^*(S_l, i')$. (In particular, note that $\mathsf{C}_{\neg N s} = I(S_l, p_l)$.) The argument for the case where i is associated with a formula occurrence in the right premiss is analogous.

Item (3) is straightforward and therefore left to the reader.

References

[1] Bahareh Afshari, Sebastian Enqvist, and Graham Leigh. Herbrand schemes for first-order logic. Preprint: `https://eprints.illc.uva.nl/id/eprint/2286.`, 2023.

[2] Bahareh Afshari, Stefan Hetzl, and Graham E. Leigh. Herbrand's theorem as higher order recursion. *Ann. Pure Appl. Log.*, 171(6):102792, 2020.

[3] David Baelde, Amina Doumane, Denis Kuperberg, and Alexis Saurin. Bouncing threads for circular and non-wellfounded proofs: Towards compositionality with circular proofs. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–13, 2022.

[4] David Baelde, Amina Doumane, and Alexis Saurin. Infinitary proof theory: the multiplicative additive case. *Computer Science Logic 2016*, 2016.

[5] Stefano Berardi and Makoto Tatsuta. Classical system of Martin-Löf's inductive definitions is not equivalent to cyclic proof system. In *International Conference on Foundations of Software Science and Computation Structures*, pages 301–317. Springer, 2017.

[6] James Brotherston. Cyclic proofs for first-order logic with inductive definitions. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 78–92. Springer, 2005.

[7] James Brotherston and Alex Simpson. Complete sequent calculi for induction and infinite descent. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 51–62. IEEE, 2007.

[8] James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation*, 21(6):1177–1216, 2011.

[9] Robert Constable and Chet Murthy. *Finding computational content in classical proofs*, pages 341–362. Cambridge University Press, USA, 1991.

[10] Thierry Coquand. Computational content of classical logic. In Andrew M. Pitts and P. Editors Dybjer, editors, *Semantics and logics of computation*, Publications of the Newton Institute, pages 33–78. Cambridge University Press, 1997.

[11] Anupam Das. A circular version of Gödel's T and its abstraction complexity. *arXiv preprint arXiv:2012.14421*, 2020.

[12] Anupam Das. On the logical complexity of cyclic arithmetic. *Logical Methods in Computer Science*, 16, 2020.

[13] Paul Downen and Zena M. Ariola. A tutorial on computational classical logic and the sequent calculus. *Journal of Functional Programming*, 28:e3, 2018.

[14] Jérôme Fortier and Luigi Santocanale. Cuts for circular proofs: semantics and cut-elimination. In *Computer Science Logic 2013 (CSL 2013)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[15] Philipp Gerhardy and Ulrich Kohlenbach. Extracting herbrand disjunctions by functional interpretation. *Arch. Math. Log.*, 44(5):633–644, 2005.

[16] Matthew Hague, Andrzej S Murawski, C-HL Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 452–461. IEEE, 2008.

[17] Ulrich Kohlenbach. *Applied Proof Theory: Proof interpretations and their Use in mathematics*. Springer, Heidelberg, 2008.

[18] Thomas Powell. Computational interpretations of classical reasoning: From the Epsilon Calculus to stateful programs. In Stefania Centrone, Sara Negri, Deniz Sarikaya, and Peter M. Schuster, editors, *Mathesis Universalis, Computability and Proof*, pages 255–290. Springer International Publishing, Cham, 2019.

[19] Luigi Santocanale. A calculus of circular proofs and its categorical semantics. In *International Conference on Foundations of Software Science and Computation Structures*, pages 357–371. Springer, 2002.

[20] Alexis Saurin. A linear perspective on cut-elimination for non-wellfounded sequent calculi with least and greatest fixed-points. In Revantha Ramanayake and Josef Urban, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 203–222, Cham, 2023. Springer Nature Switzerland.

[21] Alex Simpson. Cyclic arithmetic is equivalent to peano arithmetic. In *International Conference on Foundations of Software Science and Computation Structures*, pages 283–300. Springer, 2017.

[22] Christian Urban. *Classical logic and computation.* PhD thesis, University of Cambridge, 2000.