On the Computational Representation of Classical Logical Connectives

Jayshan Raghunandan and Alexander J. Summers

Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2RH, UK

Abstract

Many programming calculi have been designed to have a Curry-Howard correspondence with a classical logic. We investigate the effect that different choices of logical connective have on such calculi, and the resulting computational content. We identify two connectives 'if-and-only-if' and 'exclusive or' whose computational content is not well known, and whose cut elimination rules are non-trivial to define. In the case of the former, we define a term calculus and show that the computational content of several other connectives can be simulated. We show this is possible even for connectives not logically expressible with 'if-and-only-if'.

1 Introduction

There are many programming calculi which have been designed to have a Curry-Howard correspondence with a logical proof system. In recent years such calculi have been designed to explore the computational content of Classical Logic (e.g. [2,4,6,8,11,12,14]). Different authors have chosen different sets of logical connectives to treat as primitive in their logic, and designed the syntax and reduction rules of their calculi accordingly. Implication is the most popular choice of connective, since it is well-understood that its computational behaviour is related to function abstraction and application. There are calculi which do not use implication, for example that of Wadler [14]. Calculi exist which employ conjunction, disjunction, negation, and even more esoteric connectives such as difference [1,2] and constants for truth and falsity.

We consider logics with different primitive connectives and discuss general approaches to the design of corresponding term calculi. We restrict our attention to *propositional* logical connectives; an investigation of various approaches to employing quantifiers has been studied in [10].

We work in the logical context of the sequent calculus, a brief introduction to which is given in Section 2. The style of our term calculi is based on that of the $\mathcal{X}$-calculus [12],

[1] Email: jr200@doc.ic.ac.uk, ajs300m@doc.ic.ac.uk

which has a Curry-Howard correspondence with a classical sequent calculus for implication. The $\mathcal{X}$-calculus is presented in Section 3. In Section 4, we generalise the design of $\mathcal{X}$ to that of analogous term calculi based on sequent calculi with different logical connectives. Section 5 identifies a class of logical connectives to investigate, and for each, shows how to derive suitable term representations and associated reduction rules. We identify that the computational content of the 'if-and-only-if' ($\leftrightarrow$) and 'exclusive or' ($\otimes$) connectives are not well understood. Section 6 is a study of the 'if-and-only-if' connective, whose reduction rules turn out to be non-trivial to define. We define a term calculus based only on this connective, which we call $\mathcal{X}^{\leftrightarrow}$, and investigate its computational expressivity. As a surprising result we show that this new calculus can, given certain restrictions, simulate the reductions of several well-known logical connectives which are not themselves logically expressible in terms of $\leftrightarrow$. As an example, we give an interpretation of the $\mathcal{X}$-calculus into $\mathcal{X}^{\leftrightarrow}$.

2 Sequent Calculi

In recent years, various programming calculi have been proposed which are based on a Curry-Howard correspondence with sequent calculus proof systems, rather than natural deduction systems. In such a proof system for classical logic, one deals with sequents of the form $A_1, \ldots, A_m \vdash B_1, \ldots, B_n$, which should be read as "if all of $A_1, \ldots, A_m$ are all true, then (at least) one of $B_1, \ldots, B_n$ is true". Proof rules are defined for introducing a logical connective on both the left and right of a sequent (elimination rules are not used, in contrast to Natural Deduction systems). In this paper we treat the collections of formulas on the left and right of a sequent as sets, and allow arbitrary extra formulas to be included at the leaves (axioms) of a derivation, in the style of Kleene [5]. This avoids the need for the structural rules used in the original sequent calculi [3], which allows the proof system to focus on the structure of the formulas themselves.

A special rule called the *cut* is used in sequent calculi to connect two proofs together. Gentzen showed for his sequent calculi that although the cut rule might be useful for brevity, it is redundant, in the sense that any proof containing an instance of the cut rule can be transformed into a cut-free proof of the same end sequent. Gentzen defined a set of cut-elimination rules, which are non-confluent, and normalising but not strongly normalising.

An example of a sequent calculus for a logic with the implication connective only is specified by Figure 1.

$$\frac{}{\Gamma, A \vdash A, \Delta}\,(Ax) \qquad \frac{\Gamma \vdash \Delta, A \quad A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta}\,(cut)$$

$$\frac{\Gamma \vdash A, \Delta \quad B, \Gamma \vdash \Delta}{\Gamma, A{\to}B \vdash \Delta}\,({\to}_L) \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A{\to}B, \Delta}\,({\to}_R)$$

Fig. 1. A sequent-calculus for implication

In fact, this particular sequent calculus is the basis of the $\mathcal{X}$-calculus, which is described in the following section.

3 The $\mathcal{X}$-Calculus

Our work is based on the $\mathcal{X}$-calculus [12]; an untyped term annotation for classical implicative sequent calculus. We recall here the basic definitions.

Definition 3.1 [$\mathcal{X}$-Terms] The terms of the $\mathcal{X}$-calculus are defined by the following syntax, where x, y range over the infinite set of *sockets* and α, β over the infinite set of *plugs* (sockets and plugs together form the set of *connectors*).

$$\begin{array}{rllllll} P, Q ::= & \langle x.\alpha\rangle & \mid \widehat{y}P\widehat{\beta}\cdot\alpha & \mid P\widehat{\beta}\,[y]\,\widehat{x}Q & \mid P\widehat{\alpha}\dagger\widehat{x}Q & \mid P\widehat{\alpha}\not\dagger\widehat{x}Q & \mid P\widehat{\alpha}\,\backslash\!\!\dagger\,\widehat{x}Q \\ & \textit{capsule} & \textit{export} & \textit{mediator} & \textit{cut} & \textit{left-cut} & \textit{right-cut} \end{array}$$

The $\widehat{\cdot}$ symbolises that the connector underneath is bound in the attached subterm—a bound socket is written as a prefix to the term, whereas a bound plug is written as a suffix. For example in the mediator $P\widehat{\beta}\,[y]\,\widehat{x}Q$, occurrences of β are bound in the subterm P and occurrences of x are bound in Q. A connector which does not occur under a binder is said to be free. We will use *fp*(P) to denote the free plugs of P, and similarly *fs*(P) for free sockets. We work modulo α-conversion (issues regarding α-conversion have been studied in [13]). The reduction rules are specified below.

Definition 3.2 [Logical Rules] The logical rules are presented by:

$$\begin{array}{lrcll} (\textit{cap}): & \langle y.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}\langle x.\beta\rangle & \to & \langle y.\beta\rangle & \\ (\textit{exp}): & (\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha}\dagger\widehat{x}\langle x.\gamma\rangle & \to & \widehat{y}P\widehat{\beta}\cdot\gamma & \alpha\notin \textit{fp}(P) \\ (\textit{med}): & \langle y.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}(P\widehat{\beta}\,[x]\,\widehat{z}Q) & \to & P\widehat{\beta}\,[y]\,\widehat{z}Q & x\notin \textit{fs}(P,Q) \\ (\textit{exp-med}): & (\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\gamma}\,[x]\,\widehat{z}R) & \to & \left\{\begin{array}{l} Q\widehat{\gamma}\dagger\widehat{y}(P\widehat{\beta}\dagger\widehat{z}R) \\ (Q\widehat{\gamma}\dagger\widehat{y}P)\widehat{\beta}\dagger\widehat{z}R \end{array}\right\} & \begin{array}{l}\alpha\notin \textit{fp}(P), \\ x\notin \textit{fs}(Q,R)\end{array} \end{array}$$

The first three logical rules above specify a renaming (reconnecting) pro cedure, whereas the last rule specifies the basic computational step: it allows the body of the function from the export to be inserted between the two subterms of the mediator (the resulting cuts may be bracketed either way, as shown).

Definition 3.3 [Activation Rules] We define two *cut-activation* rules.

$$\begin{array}{ll} (\textit{act-}\mathrm{L}): & P\widehat{\alpha}\dagger\widehat{x}Q \to P\widehat{\alpha}\not\dagger\widehat{x}Q \;\; \textit{if } P \textit{ does not introduce } \alpha \\ (\textit{act-}\mathrm{R}): & P\widehat{\alpha}\dagger\widehat{x}Q \to P\widehat{\alpha}\,\backslash\!\!\dagger\,\widehat{x}Q \;\; \textit{if } Q \textit{ does not introduce } x \end{array}$$

where: P *introduces* x**:** Either $P = Q\widehat{\beta}\,[x]\,\widehat{y}R$ and $x\notin$ *fs*(Q,R), or $P = \langle x.\alpha\rangle$
P *introduces* α**:** Either $P = \widehat{x}Q\widehat{\beta}\cdot\alpha$ and $\alpha\notin$ *fp*(Q), or $P = \langle x.\alpha\rangle$

An activated cut is processed by 'pushing' it systematically through the syntactic structure of the circuit in the direction indicated by the tilting of the dagger. Whenever an active cut meets a circuit exhibiting the connector it is trying to communicate with, a new (inactive) cut is 'deposited', representing an attempt to communicate at this level. The pushing of the active cut continues until the level of capsules is reached, where it is either deactivated or destroyed. Once again, the inactive cut can reduce via a logical rule, or pushing can continue in the other direction. This behaviour is expressed by the following propagation rules.

Definition 3.4 [Propagation Rules] *Left Propagation*:

$$\begin{array}{rrcll}
(\not\dagger): & \langle y.\alpha\rangle\widehat{\alpha}\not\dagger\widehat{x}P & \to & \langle y.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}P & \\
(\not\dagger\mathit{cap}): & \langle y.\beta\rangle\widehat{\alpha}\not\dagger\widehat{x}P & \to & \langle y.\beta\rangle & \beta\neq\alpha \\
(\not\dagger\mathit{exp\text{-}outs}): & (\widehat{y}Q\widehat{\beta}\cdot\alpha)\widehat{\alpha}\not\dagger\widehat{x}P & \to & (\widehat{y}(Q\widehat{\alpha}\not\dagger\widehat{x}P)\widehat{\beta}\cdot\gamma)\widehat{\gamma}\dagger\widehat{x}P, & \gamma\ \mathit{fresh} \\
(\not\dagger\mathit{exp\text{-}ins}): & (\widehat{y}Q\widehat{\beta}\cdot\gamma)\widehat{\alpha}\not\dagger\widehat{x}P & \to & \widehat{y}(Q\widehat{\alpha}\not\dagger\widehat{x}P)\widehat{\beta}\cdot\gamma, & \gamma\neq\alpha \\
(\not\dagger\mathit{med}): & (Q\widehat{\beta}\,[z]\,\widehat{y}R)\widehat{\alpha}\not\dagger\widehat{x}P & \to & (Q\widehat{\alpha}\not\dagger\widehat{x}P)\widehat{\beta}\,[z]\,\widehat{y}(R\widehat{\alpha}\not\dagger\widehat{x}P) & \\
(\not\dagger\mathit{cut\text{-}cap}): & (Q\widehat{\beta}\dagger\widehat{y}\langle y.\alpha\rangle)\widehat{\alpha}\not\dagger\widehat{x}P & \to & (Q\widehat{\alpha}\not\dagger\widehat{x}P)\widehat{\beta}\dagger\widehat{x}P & \\
(\not\dagger\mathit{cut}): & (Q\widehat{\beta}\dagger\widehat{y}R)\widehat{\alpha}\not\dagger\widehat{x}P & \to & (Q\widehat{\alpha}\not\dagger\widehat{x}P)\widehat{\beta}\dagger\widehat{y}(R\widehat{\alpha}\not\dagger\widehat{x}P), & R\neq\langle y.\alpha\rangle
\end{array}$$

Right Propagation:

$$\begin{array}{rrcll}
(\backslash\!\!\!\dagger\dagger): & P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}\langle x.\beta\rangle & \to & P\widehat{\alpha}\dagger\widehat{x}\langle x.\beta\rangle & \\
(\backslash\!\!\!\dagger\mathit{cap}): & P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}\langle y.\beta\rangle & \to & \langle y.\beta\rangle, & y\neq x \\
(\backslash\!\!\!\dagger\mathit{exp}): & P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}(\widehat{y}Q\widehat{\beta}\cdot\gamma) & \to & \widehat{y}(P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}Q)\widehat{\beta}\cdot\gamma & \\
(\backslash\!\!\!\dagger\mathit{med\text{-}outs}): & P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}(Q\widehat{\beta}\,[x]\,\widehat{y}R) & \to & P\widehat{\alpha}\dagger\widehat{z}((P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}R)), & \\
 & & & & z\ \mathit{fresh} \\
(\backslash\!\!\!\dagger\mathit{med\text{-}ins}): & P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}(Q\widehat{\beta}\,[z]\,\widehat{y}R) & \to & (P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}R), & z\neq x \\
(\backslash\!\!\!\dagger\mathit{cut\text{-}cap}): & P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}(\langle x.\beta\rangle\widehat{\beta}\dagger\widehat{y}R) & \to & P\widehat{\alpha}\dagger\widehat{y}(P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}R) & \\
(\backslash\!\!\!\dagger\mathit{cut}): & P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}(Q\widehat{\beta}\dagger\widehat{y}R) & \to & (P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}Q)\widehat{\beta}\dagger\widehat{y}(P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}R), & Q\neq\langle x.\beta\rangle
\end{array}$$

We write $\to$ for the reduction relation generated by the logical, propagation and activation rules. The following are admissible rules (see [12,13]).

Lemma 3.5 (Garbage Collection and Renaming)

$$\begin{array}{rlrl}
(\mathit{gc}\text{-}\mathrm{L}): & P\widehat{\alpha}\not\dagger\widehat{x}Q\to P,\ \mathit{if}\ \alpha\notin \mathit{fp}(P) & (\mathit{ren}\text{-}\mathrm{L}): & P\widehat{\delta}\dagger\widehat{z}\langle z.\alpha\rangle,\to P[\alpha/\delta] \\
(\mathit{gc}\text{-}\mathrm{R}): & P\widehat{\alpha}\backslash\!\!\!\dagger\widehat{x}Q\to Q,\ \mathit{if}\ x\notin \mathit{fs}(Q) & (\mathit{ren}\text{-}\mathrm{R}): & \langle z.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}P,\to P[z/x]
\end{array}$$

4 The Computational Representation of a Connective

In this section, we outline some of the techniques used in the rest of the paper for deriving suitable proof rules, corresponding syntax representations and reduction rules to represent the inclusion of a particular logical connective.

We use $A, B, \ldots$ as propositional variables and $\neg$ to represent logical negation, which binds tighter than any other connective. We use $\circ$ and $\bullet$ to represent arbitrary binary connectives (logical connectives which take two arguments). For formulas F_1 and F_2 we write

$F_1 {\equiv} F_2$ (and say the formulas are *logically equivalent*) if for all assignments of truth values to propositional variables, F_1 and F_2 have the same truth value as each other.

4.1 Sequent Rules

We will assume the rules for the axiom (c.f. capsule in $\mathcal{X}$) and the cut are present and unchanged in all the systems we discuss (see Figure 1). For each logical connective of interest, suitable proof rules must be provided (or derived) for introducing the connective on the left and right-hand sides of a sequent (c.f. $\rightarrow_L$ and $\rightarrow_R$ of Figure 1).

One important point is that for various logical connectives one has a choice of how many proof rules to incorporate. This is most easily seen in the different treatments of *pairing*, typically relating to the $\wedge$ connective (although this notion is generalised in Section 5). A term is usually provided to construct a pair, but there are different approaches to the problem of dealing with pairs (making use of their individual components). One approach is to provide two *projections*, which reduce a pair to one or other of its component elements. Another is sometimes termed a 'pattern-matching' approach, in which both components are substituted in to some receiving term. These two approaches can be shown to be inter-derivable in our framework, and the decision of which to use is largely a matter of taste. As an example, for the $\wedge$ connective (conjunction), the left introduction could be specified in either of the following two ways:

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\,(\wedge_L) \quad \textit{or} \quad \left\{ \frac{\Gamma, A \vdash \Delta}{\Gamma, (A \wedge B) \vdash \Delta}\,(\wedge_{L1}) \quad \textit{and} \quad \frac{\Gamma, B \vdash \Delta}{\Gamma, (A \wedge B) \vdash \Delta}\,(\wedge_{L2}) \right\}$$

In this paper we choose the 'pattern-matching' style; that is, we will always choose to have exactly one left and right introduction rule for a binary connective.

It may not always be the case that a set of suitable sequent proof rules for a particular connective are obvious. In this case, one can proceed as follows. To derive suitable sequent rules for a binary connective $\circ$, say, choose a formula F such that $F{\equiv}A{\circ}B$ and F uses connectives for which one already knows suitable proof rules. Now, try to construct what a 'general' sequent derivation which introduces this formula on the left and right of the sequent might be. Once all of the connectives in F have been introduced, it will not be possible to proceed further in the derivation. All remaining sub-derivations to be completed translate to sub-proofs in the derived rule, while the formula F is replaced by $A{\circ}B$ for the end sequent. This process will give suitable proof rules for the connective $\circ$. This technique will be further illustrated and exploited in Sections 5 and 6.

4.2 Term Syntax

We work in the style of the $\mathcal{X}$-calculus, since this gives a simple and symmetric treatment of the inputs and outputs present within the syntax. When deriving the syntax to represent a particular proof rule, formulas which occur on the left of a sequent will become inputs (sockets) $x, y, z, \ldots$ while formulas on the right will be outputs (plugs) $\alpha, \beta, \gamma, \ldots$. Any subproofs present in the rule will be represented as subterms of the syntax. Formulas which disappear from such subproofs by application of the proof rule (formulas which are *bound* by the rule) will correspond to bound connectors on the subterms, while a new formula which is introduced by the rule corresponds to a free connector of the appropriate kind.

With these ideas in mind, it should be clear to see that the term representation of the sequent calculus in Figure 1 could well be chosen to be the syntax of $\mathcal{X}$ (see Definition 3.1). For example, the $\rightarrow_R$ rule has one subproof, which corresponds to a sub-term P, say. It binds two formulas in this subproof, one on the left of the sequent and one on the right, therefore a socket and a plug of P should be bound (say x and α respectively). The rule introduces a new formula on the right of the sequent, which leads to a free plug being present in the term representation (say β). One can easily see that the *export* of the $\mathcal{X}$-calculus, written $\widehat{x}P\widehat{\alpha}\cdot\beta$ is such a representation, with the $\cdot$ being inactive syntax, designed to make the terms easier to parse.

As a further example, consider the $\rightarrow_L$ rule. This has two subproofs, which become subterms P and Q. Each has a single formula bound, on the right and left of the sequents respectively. Thus a plug α is bound in P and a socket y is bound in Q. Finally, a free socket x should be introduced. The notation $P\widehat{\alpha}\,[x]\,\widehat{y}Q$ is chosen, with the x occurring between the two terms simply to provide a better intuition for how this term behaves; it acts as a 'hole' between the two subterms, into which a further term can be inserted to 'mediate' between P and Q (this behaviour is seen in the *exp-med* rule).

4.3 Reduction Rules

Whichever logical connectives are employed, we will always keep the following $\mathcal{X}$ reduction rules (which deal with cuts and capsules) in place:

$$\mathit{cap}, \mathit{act}\text{-L}, \mathit{act}\text{-R}, \not\nearrow\dagger, \not\nearrow\mathit{cap}, \not\nearrow\mathit{cut\text{-}cap}, \not\nearrow\mathit{cut}, \not\nwarrow\dagger, \not\nwarrow\mathit{cap}, \not\nwarrow\mathit{cut\text{-}cap}, \not\nwarrow\mathit{cut}$$

The notion of a plug or socket being *introduced* can be generalised to say P introduces x (respectively, α) iff x is free in P but not in any of its proper subterms.

Propagation rules must be defined for propagating left and right cuts through each syntactic construct. If a new syntax construct corresponds to a left-introduction rule (i.e. its free connector is a socket), two rules must be given for propagating a right-cut over it (depending on whether the free connector is that which the cut is attempting to connect to), and one for propagating a left-cut (c.f. $\not\nwarrow$*med-outs*, $\not\nwarrow$*med-ins*, $\not\nearrow$*med*). The general approach is to push copies of the cut into the subterms, leaving a copy on the outside if an occurrence of the desired connector was present at this level (c.f. $\not\nwarrow$*med-outs*). The appropriate rules for propagation over a construct which introduces a plug may be derived symmetrically (two for left cuts, one for right cuts).

This leaves the appropriate extra logical reduction rules to be defined. Each new syntax construct warrants a logical rule to specify a renaming of its introduced connector, via a cut with a capsule (see the rules *exp* and *med*, for example). Finally, for each logical connective employed, a logical rule must be defined to show how a cut between the right and left introduction of the connective may be reduced (c.f. the rule *exp-med*). We call this the *principal logical rule* for the connective, since it is the rule which specifies how these structures may be removed from a proof, creating new cuts between their subterms and simplifying the task of cut-elimination. The principal rule is the only one which cannot be methodically derived independent of the particular connective concerned. For this reason, when investigating the representation of a particular connective, as far as reduction rules are concerned we will only concern ourselves with the principal logical rule for the connective.

5 Comparing Logical Connectives

In this section, we compare various logical connectives, focusing on relationships between them and how this affects their inclusion in a term calculus. For each connective we are interested in the following three questions:

(i) What is a suitable term representation of its proof rules?

(ii) What is its principal reduction rule?

(iii) What computational content is gained by its inclusion?

5.1 Enumerating the connectives

There are an infinite number of possible logical connectives, since a connective may apply to an arbitrary (but usually fixed) number of arguments (hereon its *arity*). It is extremely rare in practice for authors to employ connectives with arity greater than two (although for an example, see [7]). To decide on a set of connectives for our study, we found the following three questions of interest:

(a) How many logical connectives are there of arity n ($n \geq 0$)?

(b) How many of these depend on all n inputs (we say these have *true arity* n)?

(c) How many of these *always* depend on all n inputs?

To explain the second question, take for example the binary connective which has inputs A and B and always evaluates to A (ignoring B). In a sense one could see this as a unary connective, since it only makes use of one input. This gives a way of identifying those connectives of arity n which we regard as degenerate cases.

The third question regards a stronger notion; that the value of a connective should, in every input state, depend on all of its inputs. As a non-example, the evaluation of a conjunction ($\wedge$) may be 'short-cut'; if its first argument turns out to be false then the second need not be considered. Thus conjunction does not satisfy the criterion outlined in the third question.

The answer to each of these questions is given by the following result:

Theorem 5.1 (Enumerating Logical Connectives) *For any integer $n \geq 0$:*

(*a*) *There are 2^{2^n} logical connectives of arity n.*

(*b*) *The number of these which depend on all n inputs (those of true arity n), $t(n)$ is given by the following formula: $t(n) = 2^{2^n} - \sum_{i=0}^{n-1} \binom{n}{i} t(i)$.*

(*c*) *There are exactly two connectives of arity n which always depend on all n inputs; these are the parity function (which is true exactly when an even number of its arguments are), and its negation.*

Proof.

(a) Each connective is exactly specified by a 'truth-table'; defining whether it evaluates to true or false in each of the 2^n possible input states. The result follows by counting all such truth tables.

(b) By counting; start with all connectives of arity n, and subtract off those which depend on strictly fewer inputs. Since each of these may depend on a different subset of the actual inputs available, one must count them for each appropriate subset (hence the $\binom{n}{i}$).

(c) Let f be some such connective of arity n, which we represent as a function of n inputs, $f(i_1, i_2, \ldots, i_n)$. We write 0 for a false input, 1 for true, and $\overline{i}$ for the negation on these inputs (i.e. $\overline{1} = 0$ etc.). Our condition on f states that given a set of input values $i_1, \ldots, i_n$, the value of $f(i_1, i_2, \ldots, i_n)$ depends on all of $i_1, \ldots, i_n$, or equivalently, if we change (negate) any one of the inputs, the value of $f(i_1, i_2, \ldots, i_n)$ must change. Now consider setting all inputs to 0, and say $f(0, \ldots, 0) = a$ where $a = 0$ or $a = 1$. By our condition on f, if we now negate any one of the inputs, the value of $f(i_1, i_2, \ldots, i_n)$ must be $\overline{a}$. In general, let j be the number of true inputs, i.e. $j = |\{i_k \mid 1 \leq k \leq n \text{ and } i_k = 1\}|$. A straightforward induction on j shows that:

$$f(i_1, i_2, \ldots, i_n) = \left\{ \begin{array}{l} a \text{ if } j \text{ is even} \\ \\ \overline{a} \text{ if } j \text{ is odd} \end{array} \right\}$$

Thus f is exactly specified by the choice of a. Therefore there are exactly two such functions, the parity function and its negation.

□

Examining the second part of this result, we see that $t(0) = 2$. These two connectives are the logical constants $\top$ and $\bot$, which can be seen as connectives of arity 0 (they can be seen as the parity and not-parity connectives of arity 0). It is easy to see that $t(1) = 2$ also, and these are the identity connective (which returns whatever input it receives unchanged) and negation ($\neg$). Furthermore, we see $t(2) = 10$, i.e. there are 10 different logical connectives of true arity 2. These connectives are listed in the next section, although the reader might find it interesting to try to name them all first!

We will henceforth only interest ourselves in connectives of arity 2 (and below). As commented, this choice is common in the literature. On a practical note, since the reader may verify that $t(3) = 218$, an exhaustive analysis of all possible connectives of any greater arity would be too cumbersome.

5.2 The Binary Connectives

In this section, we will give the complete set of possible binary connectives, and provide an analysis of them with respect to the three questions outlined at the start of Section 5. We are interested in possible relationships between these connectives, and how these are reflected by their computational counterparts. For example, *duality* is a well-known concept relating binary logical connectives, and it will be seen that this relationship carries over into their computational behaviour (this is related to the results of [2,14]).

We make use of the following relationships between connectives:

Definition 5.2 [Relating connectives] For any two binary connectives $\circ$,$\bullet$:

Duality We say $\bullet$ is the *dual* of $\circ$ iff $A \bullet B \equiv \neg(\neg A \circ \neg B)$.

Negation We say $\bullet$ is the *negation* of $\circ$ iff $A \bullet B \equiv \neg(A \circ B)$.

Reversal We say $\bullet$ is the *reverse* of $\circ$ iff $A \bullet B \equiv B \circ A$.

$$\begin{tikzcd}
A\wedge B \arrow[loop left, "S"] \arrow[r, leftrightarrow, "D"] \arrow[d, leftrightarrow, "N"] & A\vee B \arrow[loop right, "S"] \arrow[d, leftrightarrow, "N"] \\
A\uparrow B \arrow[loop left, "S"] \arrow[r, leftrightarrow, "D"] & A\downarrow B \arrow[loop right, "S"]
\end{tikzcd}
\qquad
\begin{tikzcd}
B-A \arrow[r, leftrightarrow, "D"] \arrow[d, leftrightarrow, "N"] \arrow[dr, leftrightarrow, "S"] & A\rightarrow B \arrow[d, leftrightarrow, "N"] \arrow[dl, leftrightarrow] \\
B\rightarrow A \arrow[r, leftrightarrow, "D"] & A-B
\end{tikzcd}$$

$$\begin{tikzcd}
\top \arrow[loop above, "S"] \arrow[d, leftrightarrow, bend right, "N"'] \arrow[d, leftrightarrow, bend left, "D"] \\
\bot \arrow[loop below, "S"]
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[loop left, "D"] \arrow[r, leftrightarrow, "S"] \arrow[d, leftrightarrow, "N"] & B \arrow[loop right, "D"] \arrow[d, leftrightarrow, "N"] \\
\neg A \arrow[loop left, "D"] \arrow[r, leftrightarrow, "S"] & \neg B \arrow[loop right, "D"]
\end{tikzcd}
\qquad
\begin{tikzcd}
A\leftrightarrow B \arrow[loop above, "S"] \arrow[d, leftrightarrow, bend right, "N"'] \arrow[d, leftrightarrow, bend left, "D"] \\
A\otimes B \arrow[loop below, "S"]
\end{tikzcd}$$

Note: ↑ = nand, ↓ = nor, ⊗ = xor

Fig. 2. Binary Connectives

$$\frac{\Gamma \vdash B, \Delta \quad \Gamma, A \vdash \Delta}{\Gamma, A{\leftarrow}B \vdash \Delta}\ (\leftarrow_L) \qquad\qquad \frac{\Gamma, B \vdash A, \Delta}{\Gamma \vdash A{\leftarrow}B, \Delta}\ (\leftarrow_R)$$

Fig. 3. Sequent Rules for reverse implication $\leftarrow$

Flipping inputs We say $\bullet$ is obtained from $\circ$ by *flipping an input* if either $A \bullet B \equiv \neg A \circ B$ or $A \bullet B \equiv A \circ \neg B$.

In all but the last case, these concepts describe self-inverse functions (e.g. the dual of the dual of a connective is the connective itself).

The binary connectives include conjunction ($\wedge$), disjunction ($\vee$), nand ($\uparrow$) and nor ($\downarrow$). There is implication ($\rightarrow$) and its reverse ($\leftarrow$), and the so-called 'difference' operator ($-$), where $A{-}B \equiv A{\wedge}\neg B$. For the reverse connective of $-$ (for which there is no standard symbol) we tend to simply use $-$ and swap the arguments, but will shortly be able to dispense with this slight abuse of notation. As well as these, there is 'if-and-only-if' ($\leftrightarrow$), 'exclusive or' ($\otimes$), and the degenerate cases ($\top$,$\bot$ and the identity and negation on each argument, which we will write ID A, $\neg A$, ID B and $\neg B$). Although we call these degenerate cases *binary* connectives, we will treat them as having their true arities (e.g. when using negation we only mention the input which it uses). All these connectives are illustrated in Figure 2, along with arrows to represent duality (D), negation (N) and reversal (R) of connectives.

Firstly, we wish to examine the effect of the 'reversal' of a connective with respect to our questions of interest. For example, consider the connective $\leftarrow$. A sensible pair of sequent rules for this connective is shown in Figure 3. Deriving the syntax needed to represent these rules, we find that we can use exactly the same as that for implication. This is because the

$$\begin{tikzcd}
\top \arrow[loop above, "F"] \arrow[d, bend right, leftrightarrow, "N"'] \arrow[d, bend left, leftrightarrow, "D"] & \mathrm{ID} \arrow[loop above, "\substack{D\\F}"] \arrow[d, leftrightarrow, "N"] & \wedge \arrow[r, leftrightarrow, "D"] \arrow[rr, bend left, leftrightarrow, "F"] \arrow[d, leftrightarrow, "N"] & \vee \arrow[d, leftrightarrow, "N"] \arrow[dr, leftrightarrow, "F"] & - \arrow[d, leftrightarrow, "{N, D}"] \arrow[dl, leftrightarrow] & \leftrightarrow \arrow[d, bend right, leftrightarrow, "N"'] \arrow[d, leftrightarrow, "F"] \arrow[d, bend left, leftrightarrow, "D"] \\
\bot \arrow[loop below, "F"] & \neg \arrow[loop below, "\substack{D\\F}"] & \uparrow \arrow[r, leftrightarrow, "D"] \arrow[rr, bend right, leftrightarrow, "F"'] & \downarrow & \rightarrow & \otimes
\end{tikzcd}$$

Note: $\uparrow$ = nand, $\downarrow$ = nor, $\otimes$ = xor

Fig. 4. Binary Connectives Modulo Reversals

same inputs and outputs are bound and introduced in the rules; the only difference with the rules for implication is in the positioning of A and B, which is irrelevant once the types are removed. Similarly, the reduction rules required to represent this connective will be exactly the same as those for implication, and therefore so will the computational content obtained. These ideas generalise to any connective and its reverse.

As a result of this observation, we choose to examine the connectives in question modulo reversals. Since most of the connectives in Figure 2 are symmetrical (remain the same when reversed), this actually only reduces the number of connectives in question by four. Our notation becomes rather less cumbersome, in that we need not write formulas to define any of the connectives (e.g. $B{-}A$ was used to write the reverse of $A{-}B$); we can now write an unambiguous symbol for each. This is shown in Figure 4, where the arrows indicating duality (D), negation (N) and flipping inputs (F) group together related connectives. It remains for us to explain the significance of these three relationships.

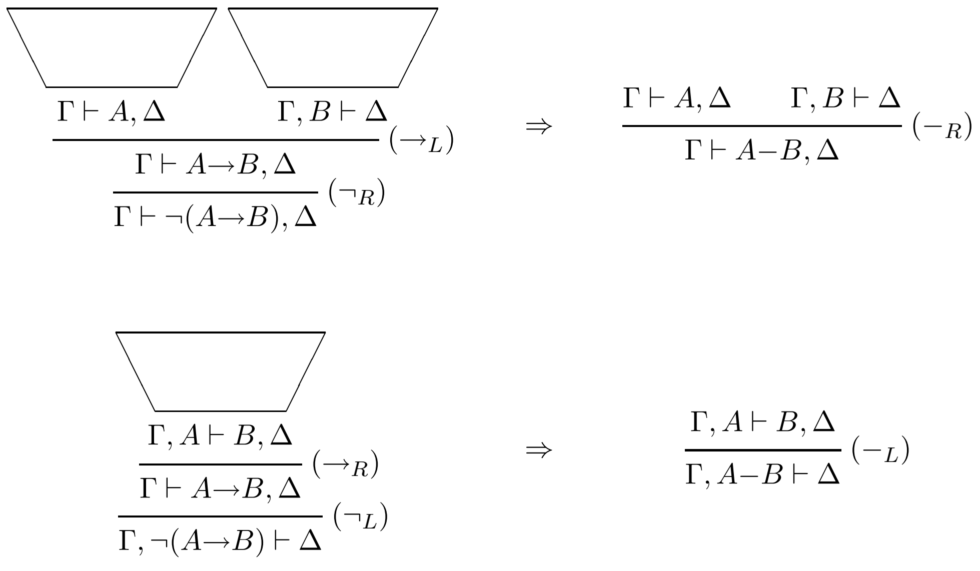
Before examining the effect of negating a connective, it is useful to examine the negation connective itself. The sequent rules for negation are as follows:

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\,(\neg_L) \qquad\qquad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\,(\neg_R)$$

The first rule binds a formula on the left of the sequent and produces a new one on the right, while the second does the opposite. The syntax we choose to use for negation reflects this swapping of inputs for outputs in the simplest way possible; we write $x{\cdot}P\widehat{\alpha}$ and $\widehat{y}Q{\cdot}\beta$ for the left and right terms respectively. The principal reduction rule for negation is as follows:

$$(\widehat{x}P{\cdot}\alpha)\widehat{\alpha} \dagger \widehat{y}(y{\cdot}Q\widehat{\beta}) \;\rightarrow\; Q\widehat{\beta} \dagger \widehat{x}P \qquad \alpha \notin \mathit{fp}(P), y \notin \mathit{fs}(Q)$$

Given the sequent rules for any connective it is straightforward to derive suitable sequent rules for the negation of the connective. For example, the negation of implication ($\rightarrow$) is the 'difference' connective ($-$), and by seeking suitable derivations for the formula $\neg(A{\rightarrow}B)$ on both the left and the right of a sequent, one can derive the appropriate rules for 'difference', as shown in Figure 5. Notice that appropriate syntax to represent 'difference' will have the same subterms, inputs and outputs as for implication, except that the free connector introduced appears on the opposite side of the sequent (due to the negation). For example, the syntax added in the case of the difference connective might be $P\widehat{\beta}\,[\alpha]\,\widehat{x}Q$ for the right-introduction rule and $x{\cdot}\widehat{y}P\widehat{\beta}$ for the left. This generalises to any connective and its negation; the term representations will be identical for each, but with the left and right terms exchanged. Furthermore, in defining a cut-elimination rule, one can see that the



$$\dfrac{\dfrac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma \vdash A{\rightarrow}B, \Delta}\,(\rightarrow_L)}{\Gamma \vdash \neg(A{\rightarrow}B), \Delta}\,(\neg_R) \quad \Rightarrow \quad \dfrac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma \vdash A{-}B, \Delta}\,(-_R)$$

$$\dfrac{\dfrac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A{\rightarrow}B, \Delta}\,(\rightarrow_R)}{\Gamma, \neg(A{\rightarrow}B) \vdash \Delta}\,(\neg_L) \quad \Rightarrow \quad \dfrac{\Gamma, A \vdash B, \Delta}{\Gamma, A{-}B \vdash \Delta}\,(-_L)$$

Fig. 5. Deriving the sequent rules for the difference connective $-$

reduct of the key logical rule will be the same in the cases of $\rightarrow$ and $-$, and in general for a connective and its negation.

The relationship between a connective and its dual, in terms of its computational representation, can also be seen to induce a relationship between their term representations. In this case, as well as the introduced formula 'swapping sides', the formulas which are bound in the proof rules also do so. For example, compare the rules for $\wedge$ and $\vee$:

$$\dfrac{\Gamma,\ A,\ B \vdash \Delta}{\Gamma,\ A{\wedge}B \vdash \Delta}\,(\wedge_L) \qquad \dfrac{\Gamma \vdash A,\ \Delta \qquad \Gamma \vdash B,\ \Delta}{\Gamma \vdash A{\wedge}B,\ \Delta}\,(\wedge_R)$$

$$\dfrac{\Gamma \vdash A,\ B,\ \Delta}{\Gamma \vdash A{\vee}B,\ \Delta}\,(\vee_R) \qquad \dfrac{\Gamma,\ A \vdash \Delta \qquad \Gamma,\ B \vdash \Delta}{\Gamma, A{\vee}B \vdash \Delta}\,(\vee_L)$$

One can see a striking similarity here. In this sequent calculus setting, it is reasonable to view disjunction as another kind of 'pairing'; the left rule is a pair of two proofs (binding a formula on the left of each), whereas the right rule provides the facility to interact with the members of such a pair.

The effect of flipping an input is to negate only one of the inputs to a connective, which in turn corresponds to the bound occurrences of one of the formulas swapping sides in the rules. For example, implication can be obtained from disjunction by flipping the first input ($A{\rightarrow}B \equiv \neg A{\vee}B$). One can see this also by comparing the sequent rules. In this sense, it is possible in the sequent calculus to see even implication as a kind of pairing. Examining the syntax of $\mathcal{X}$ (for brevity, compared to dealing in the proof rules), one can regard the mediator $Q\widehat{\alpha}\,[x]\,\widehat{y}R$ as a pair of two terms Q and R, binding an output of one and an input of another. The export $\widehat{z}P\widehat{\beta}{\cdot}\gamma$ is the term which can 'deal with the pair'; providing connectors to connect to both elements of the pair, analogously with (for example) the term corresponding to $\wedge_L$.

From the discussions above, it can be seen that once one knows the sequent rules (and hence, an appropriate term representation) for a particular connective, one can easily derive them for the negation and dual of the connective, and any connective which is obtained by flipping an input. In particular, the six connectives which are joined to each other by various arrows in Figure 4 (including $\wedge$,$\vee$ and $\rightarrow$) all have related sequent rules. Each can in fact be regarded as a kind of pairing connective; the differences lie in whether inputs or outputs are bound in the two subterms which make up the pair, and whether the pair is made available on an introduced input or output. We will sometimes refer to these six connectives as the *pairing connectives*.

As can be seen from Figure 4, the remaining connectives come in related groups of two. The syntax and main rule for the negation connective have already been discussed, while the identity connective can be seen to have a very trivial computational content (at best it provides a kind of *aliasing*, where a connector is bound within a subterm and then immediately exported again with a new name).

The $\top$ and $\bot$ connectives are rather unusual, since it turns out they each have no sensible proof rule for introducing the connective on one side of the sequent (in fact a rule can be added but it amounts to a special case of weakening). In the case of $\top$, there is only a sensible rule for introduction on the right, and symmetrically $\bot$ only has an introduction rule on the left. These rules are given below:

$$\frac{}{\Gamma, \vdash \top, \Delta}\,(\top_R) \qquad\qquad \frac{}{\Gamma, \bot \vdash \Delta}\,(\bot_L)$$

Since these rules introduce a new formula without binding any existing ones, they can be seen to be inhabited by terms which make available an output (respectively input) which isn't connected to anything. As far as reduction rules are concerned, it is impossible to add the usual principal logical rule, since there is no pair of left and right terms to connect. When one considers a cut between (for example) a $\top_R$ rule on the left and some other term in the right, it is clear that the connector bound on the other side of the cut must be introduced by weakening (if the cut is typeable). In this way the terms to represent $\top$ and $\bot$ can be used to provide 'dead-end' cuts, which when evaluated simply disappear (c.f. Lemma 3.5). As an example of the kind of computational content expressible, if one adds the syntax for the $\bot_L$ rule to the existing $\mathcal{X}$-calculus, then one can express direct manipulation of continuations (since with $\rightarrow$ and $\bot$ one can express negation).

As a separate point, it should be noted that if one employs more than one logical connective in a term calculus, it will be possible to create (untypeable) cuts between their respective syntax representations to which no reduction rule applies. For example, if one were to cut the term representation of the $\top_R$ rule with a mediator, there would be no sensible way to evaluate the cut. Therefore, when more than one logical connective is employed, the notion of normal form is extended; in particular it will be possible to have (untypeable) normal forms which contain cuts.

There remain only two binary connectives to discuss; being $\leftrightarrow$ ('if-and-only-if') and $\otimes$ ('exclusive or'). These two are related in the diagram; in fact the two connectives are related by negation, duality and may each be obtained from the other by flipping either input. In a sense, the (similar) operations they describe are difficult to relate directly to any of the other connectives; there are no 'simple' equivalent formulas which express these connectives in terms of the others. It is of course possible to encode these connectives using others, but as the following result shows, they must be expressed in a more complicated way.

Theorem 5.3 (Expressing $\leftrightarrow$, $\otimes$) *Let S be the set of binary boolean connectives without $\leftrightarrow$ and $\otimes$. There is no formula F expressible using only the connectives in S such that* *both:*

(a) *F is logically equivalent to either $A{\leftrightarrow}B$ or $A{\otimes}B$.*

(b) *A and B occur in F only once.*

Remark 5.4 In contrast, all of the connectives in S can be expressed in terms of other connectives in S using A and B only once; in a sense they can be expressed more directly than the two connectives in question.

The following technical lemma allows us to show the theorem:

Lemma 5.5 (Removing $\top$, $\bot$ and ID) *If F is a formula constructed using the binary connectives, and the propositional variables A and B, then there exists a formula G such that:*

(a) *$G \equiv F$.*

(b) *A and B each occur in G no more times than they do in F.*

(c) *No other propositional variables occur in G.*

(d) *G does not use the ID connective.*

(e) *Either G does not mention $\top$ and $\bot$, or either $G = \top$ or $G = \bot$.*

Proof. Just the idea of the proof is given here. Firstly, it is clear that any uses of the ID connective can be simply removed while maintaining an equivalent formula. One can then define a rewrite system (using equivalences) to eliminate all occurrences of $\top$ and $\bot$ which are underneath another connective. For example, one rewrites $A{\wedge}\top$ to A, and $A{\rightarrow}\bot$ to $\neg A$. It is easy to show the rewrite system is strongly normalising, and that its normal forms satisfy the criteria listed. $\square$

Proof. [of Theorem 5.3] Suppose that such a formula F exists and seek a contradiction. Clearly F cannot be equivalent to $\top$ or $\bot$. Hence, by Lemma 5.5 there exists a formula $G \equiv F$ which doesn't mention $\top$,$\bot$ or ID, and mentions A and B at most once. Note that $\leftrightarrow$ and $\otimes$ are respectively the parity function on two arguments, and its negation. Since the truth value of $A{\leftrightarrow}B$ depends on the truth values of both arguments, G must mention A and B exactly once. Now remove any double-negations which may occur, to obtain a formula G'. Without loss of generality G' is of the form $\circ_1((\circ_2 A) \bullet (\circ_3 B))$, where $\bullet$ is one of $\wedge, \vee, \uparrow, \downarrow, \rightarrow, -$, while $\circ_1, \circ_2, \circ_3$ are positions in which $\neg$ may or may not occur. Without loss of generality again, by assumption $G' \equiv A{\leftrightarrow}B$ (the case for $\otimes$ is identical). $A{\leftrightarrow}B$ always depends on the values of both A and B to evaluate its result, whereas by Theorem 5.1(3), $\bullet$ does not always depend on both the values of its arguments, therefore G' does not always depend on the values of both A and B. Contradiction. $\square$

This theorem suggests that the two connectives $\leftrightarrow$ and $\otimes$ may have some interesting complexity which the other binary connectives do not. It seems natural to investigate the computational content of these two connectives, which appears not to have been attempted so far in the literature. In particular, no cut-elimination rule (or analogously, proof reduction rule in a Natural Deduction setting) seems to have been defined for these connectives. It is these concerns which motivate the next section.

6 Interpreting if-and-only-if

In this section we study the computational behaviour of the logical connective ‘if-and-only-if’ (‘iff’ for short) that evaluates to true exactly when its two arguments have the same truth value. We could equally have chosen to study the negation of this connective ‘exclusive-or’, whose $\mathcal{X}$-style term representations will be almost the same except that the free connector that is introduced in each term will be of the opposite kind (input versus output).

We are able to determine the form of the left and right introduction rules for the iff connective via the equivalence $A{\leftrightarrow}B \equiv \neg(A{\vee}B){\vee}(A{\wedge}B)$ for example. From this, we can construct derivations whose conclusions introduce this compound formula on the left and right of a sequent. (Detailed proofs are given in Appendix A).

Condensing these derivations gives us the $({\leftrightarrow}_L)$ and $({\leftrightarrow}_R)$ introduction rules shown in Figure 6, which we can inhabit with $\mathcal{X}$-style terms in the usual way. We write the corresponding ‘iff-left’ and ‘iff-right’ terms as $[M\widehat{\mu}\widehat{\sigma}\,[y]\,\widehat{i}\widehat{j}N]$ and $[\widehat{x}P\widehat{\alpha},\widehat{z}Q\widehat{\delta}].\gamma$ respectively.

$$\frac{\Gamma \vdash A, B, \Delta \qquad \Gamma, A, B \vdash \Delta}{\Gamma, (A{\leftrightarrow}B) \vdash \Delta}\,({\leftrightarrow}_L) \qquad \frac{\Gamma, A \vdash B, \Delta \qquad \Gamma, B \vdash A, \Delta}{\Gamma \vdash (A{\leftrightarrow}B), \Delta}\,({\leftrightarrow}_R)$$

Fig. 6. ${\leftrightarrow}_L$ and ${\leftrightarrow}_R$ introduction rules

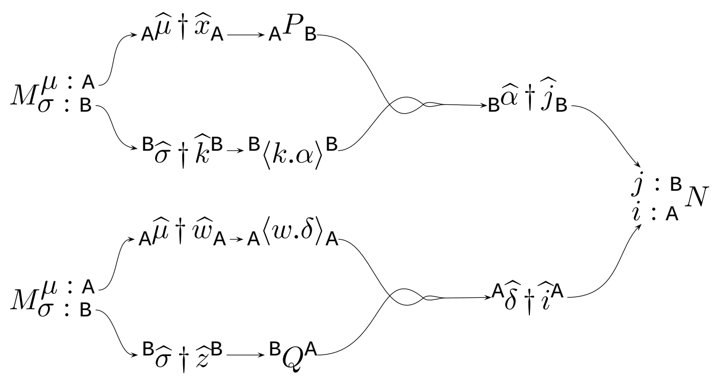
The principal logical rule for iff should transform a proof that cuts together an $({\leftrightarrow}_R)$ formula with an $({\leftrightarrow}_L)$ formula, or in $\mathcal{X}$ notation,

$$([\widehat{x}P\widehat{\alpha},\widehat{z}Q\widehat{\delta}].\gamma)\widehat{\gamma} \dagger \widehat{y}([M\widehat{\mu}\widehat{\sigma}\,[y]\,\widehat{i}\widehat{j}N]) \quad ,\gamma, y \textit{ are introduced.}$$

The reduct is not straightforward to determine. The rules for the iff connective each bind *two* inputs and *two* outputs, and each rule has two subterms. We observe a striking resemblance between these terms and those used to represent the implication connective (i.e. the syntax of $\mathcal{X}$, Definition 3.1). The iff-right term is reminiscent of an export term, except two ‘functions’ are available over the same interface rather than one (n.b. $A{\leftrightarrow}B{\equiv}(A{\rightarrow}B){\wedge}(B{\rightarrow}A)$). The iff-left term is reminiscent of a mediator with two binders over each of its subterms instead of one.

In the case of a mediator, $R\widehat{\psi}\,[l]\,\widehat{k}S$, we seek to connect the terms R and S together via the provided connectors. In general, connecting ψ to k directly would result in the restriction that our ‘implications’ must be of the form $A{\rightarrow}A$; instead we allow the body of an export to be inserted to ‘mediate’ between these two subterms. If we think of the iff-left term as a kind of mediator, the problem we must solve is again that of connecting outputs and inputs between the terms M and N. However, even in the general case, M and N have bound connectors with types in common; it would seem that we have everything we need to connect these terms together directly. $M\widehat{\mu}$ appears to connect well with $\widehat{i}N$ and $M\widehat{\sigma}$ appears to connect well with $\widehat{j}N$.

In general this cannot be done, since the underlying proof sequents interpret the types of the two inputs as formulas that are read conjunctively, and the types of the two outputs as formulas that are read disjunctively. In this context, M offers a value of type A *or* a value of type B (loosely a value of type $A{\vee}B$) while N requires both a value of type A and a value of type B (loosely, requires a value of type $A{\wedge}B$). Therefore, the problem we must solve in trying to join these two proofs is essentially that of determining how we can convert from a value of type $A{\vee}B$ to a value of type $A{\wedge}B$, i.e. we intuitively need



M^{μ}_{σ} : A : B
$_{\mathsf{A}}\widehat{\mu}\dagger\widehat{x}_{\mathsf{A}}$
$_{\mathsf{A}}P_{\mathsf{B}}$
$^{\mathsf{B}}\widehat{\sigma}\dagger\widehat{k}^{\mathsf{B}}$
$^{\mathsf{B}}\langle k.\alpha\rangle^{\mathsf{B}}$
$_{\mathsf{B}}\widehat{\alpha}\dagger\widehat{j}_{\mathsf{B}}$
j : B N
i : A
M^{μ}_{σ} : A : B
$_{\mathsf{A}}\widehat{\mu}\dagger\widehat{w}_{\mathsf{A}}$
$_{\mathsf{A}}\langle w.\delta\rangle_{\mathsf{A}}$
$^{\mathsf{A}}\widehat{\delta}\dagger\widehat{i}^{\mathsf{A}}$
$^{\mathsf{B}}\widehat{\sigma}\dagger\widehat{z}^{\mathsf{B}}$
$^{\mathsf{B}}Q^{\mathsf{A}}$

Fig. 7. Connection diagram for the reduct of $([\widehat{x}P\widehat{\alpha},\widehat{z}Q\widehat{\delta}].\gamma)\widehat{\gamma}\dagger\widehat{y}([M\widehat{\mu}\widehat{\sigma}\;[y]\;\widehat{ij}N])$, i.e. $((M\widehat{\mu}\dagger\widehat{x}P)\widehat{\sigma}\dagger\widehat{k}\langle k.\alpha\rangle)\widehat{\alpha}\dagger\widehat{j}(((M\widehat{\sigma}\dagger\widehat{z}Q)\widehat{\mu}\dagger\widehat{w}\langle w.\delta\rangle)\widehat{\delta}\dagger\widehat{i}N)$

something of type $(A\vee B)\to(A\wedge B)$. Note that this 'intuitive' formula is actually logically equivalent to $A\leftrightarrow B$, which is the kind of functionality provided on γ by the iff-right term.

We return to the previous method of determining the principal logical rule as detailed in Section 5, i.e. that of considering how one would reduce a cut between derivations that introduce a formula logically equivalent to $A\leftrightarrow B$. We cut together the proofs that derive $\neg(A\vee B)\vee(A\wedge B)$ on the left and right of the sequent and reduce them using the cut-elimination rules for negation, disjunction and conjunction. A possible reduction sequence is given in Appendix B. Condensing, then annotating the resulting proof yields the reduct:

$$((M\widehat{\mu}\dagger\widehat{x}P)\widehat{\sigma}\dagger\widehat{k}\langle k.\alpha\rangle)\widehat{\alpha}\dagger\widehat{j}(((M\widehat{\sigma}\dagger\widehat{z}Q)\widehat{\mu}\dagger\widehat{w}\langle w.\delta\rangle)\widehat{\delta}\dagger\widehat{i}N)$$

This is better understood in the diagrammatic form of Figure 7. The twisting of wires represents an (implicit) contraction in the proof, which 'merges' two connections (occurrences of the same formula) into one. We use P to convert the type of one of the outputs of M, so that both end up with the same type. The cut with a capsule is used to rename the other output (to α, the same name as the output of P) so that they can be contracted into one. In this way, we can connect the two outputs of M to a single input of N via a cut. Making a copy of the term M allows us to simultaneously connect each output to each input of N; without two copies, it is difficult to construct cuts that make all of these connections.

An alternative (and symmetrical) reduction path to that shown in Appendix B yields the following reduct.

$$(M\widehat{\mu}\dagger\widehat{x}(\langle x.\pi\rangle\widehat{\pi}\dagger\widehat{i}(P\widehat{\alpha}\dagger\widehat{j}N)))\widehat{\sigma}\dagger\widehat{z}(\langle z.\tau\rangle\widehat{\tau}\dagger\widehat{j}(Q\widehat{\delta}\dagger\widehat{i}N))$$

One can see that in this alternative two copies of N (rather than M) are made and inputs are renamed rather than outputs. We are able to condense the connection diagram of Figure 7 into a form which focuses on the direct connections made via each cut (see Figure 8). We show this for both the reducts mentioned above.

These can be interpreted as $\mathcal{X}$-style terms, leading us to the following definition.

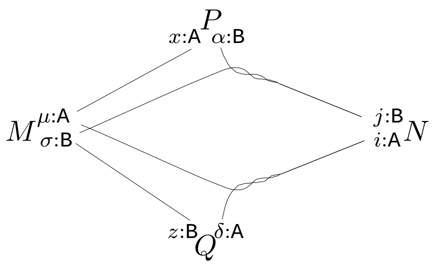
Definition 6.1 [Principal iff-reduction rule with copying] The term

$$([\widehat{x}P\widehat{\alpha},\widehat{z}Q\widehat{\delta}].\gamma)\widehat{\gamma}\dagger\widehat{y}([M\widehat{\mu}\widehat{\sigma}\;[y]\;\widehat{ij}N])$$

where, γ, y are *introduced* reduces to one of the following variants (with k,w,π,τ *fresh*).

$$(a)\;\;((M\widehat{\mu}\dagger\widehat{x}P)\widehat{\sigma}\dagger\widehat{k}\langle k.\alpha\rangle)\widehat{\alpha}\dagger\widehat{j}(((M\widehat{\sigma}\dagger\widehat{z}Q)\widehat{\mu}\dagger\widehat{w}\langle w.\delta\rangle)\widehat{\delta}\dagger\widehat{i}N)$$

$$(b)\;\;(M\widehat{\mu}\dagger\widehat{x}(\langle x.\pi\rangle\widehat{\pi}\dagger\widehat{i}(P\widehat{\alpha}\dagger\widehat{j}N)))\widehat{\sigma}\dagger\widehat{z}(\langle z.\tau\rangle\widehat{\tau}\dagger\widehat{j}(Q\widehat{\delta}\dagger\widehat{i}N))$$



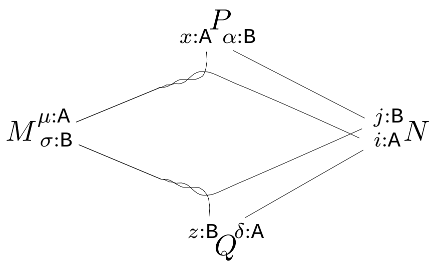
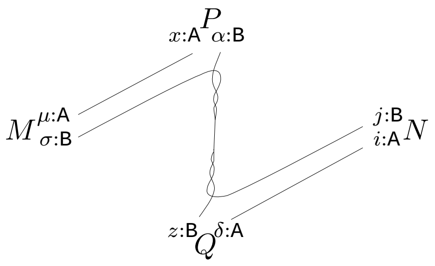


Fig. 8. Simplified connection diagrams for the reducts of Definition 6.1

As mentioned previously, a copy of either M or N is used to facilitate the connection of each output of M to each input of N. The question arises of whether this copying is necessary. One of the graphs shown in Figure 8 renames both outputs of M while the other renames both inputs of N. We sought to explore other ways in which M and N could be connected and more specifically, whether it would be possible to obtain a reduct for the principal logical rule for $\leftrightarrow$ which did not require copying. We sought to distribute the connections in a more symmetrical fashion because we believed that the copying was only necessary due to the large number of connections being made with one term or the other. We discovered a solution where we rename *one* output in M and *one* input in N. This leads to the diagrams shown in Figure 9. The reader can verify that a path exists from each output of M to to each input of N.



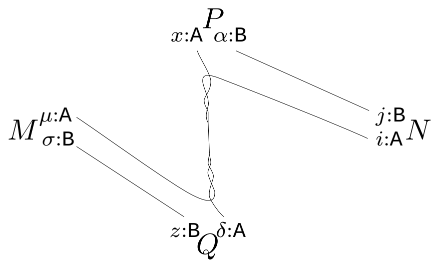


Fig. 9. Simplified Connection Diagrams for the Reducts of Definition 6.2

This leads us to a simpler definition for the principal logical rule.

Definition 6.2 [Simplified Principal iff-reduction Rule] The term

$$([\widehat{x}P\widehat{\alpha}, \widehat{z}Q\widehat{\delta}].\gamma)\widehat{\gamma} \dagger \widehat{y}([M\widehat{\mu}\widehat{\sigma}\ [y]\ \widehat{i}\widehat{j}N])$$

where, γ, y *introduced* and k, π *fresh* reduces to one of the following variants.

$$(a)\ \ ((M\widehat{\mu} \dagger \widehat{x}P)\widehat{\sigma} \dagger \widehat{k}\langle k.\alpha\rangle)\widehat{\alpha} \dagger \widehat{z}(\langle z.\pi\rangle\widehat{\pi} \dagger \widehat{j}(Q\widehat{\delta} \dagger \widehat{i}N))$$

$$(b)\ \ ((M\widehat{\sigma} \dagger \widehat{z}Q)\widehat{\mu} \dagger \widehat{k}\langle k.\delta\rangle)\widehat{\delta} \dagger \widehat{x}(\langle x.\pi\rangle\widehat{\pi} \dagger \widehat{i}(P\widehat{\alpha} \dagger \widehat{j}N))$$

These reducts will be significantly cheaper to evaluate than those given in Definition 6.1 since an extra copy of M (or N) is not required and fewer cuts are needed to represent all the necessary connections. From now on, we will use this version of the principal logical rule for iff.

6.1 *Simulating other connectives*

If a logical connective is able to express another connective, then it is straightforward to simulate the computational content of the latter connective in a term-calculus corresponding to the former. The only logical connectives expressible by ($\leftrightarrow$) are ($\top$) and (ID), which might lead us to believe its simulation capabilities in this sense are limited. However, we find this is not the case; in fact we are able to simulate the reductions associated with several other connectives, i.e. we can encode the syntax for these other connectives in such a way that reductions are preserved. When this is the case, we say we can *computationally express* the connective (which may or may not be expressible in a logical sense).

If we look at the iff-terms themselves, we find they provide a wealth of input and output connectors arranged in different combinations over a number of subterms. We also observe that the principal logical rule (see Definition 6.2) offers a number of interactions between these different subterms, giving scope for modelling a variety of computational behaviour, some of which may be new.

As an example of a connective which can be computationally expressed (but not logically expressed) by iff, we show how to express the syntax and reduction behaviour of the $\mathcal{X}$-calculus (based on the implication connective) in a term calculus based on the iff connective (which we call $\mathcal{X}^{\leftrightarrow}$). We give the definition of this new calculus below. For brevity we omit the activated cuts, which should be treated analogously.

Definition 6.3 [Syntax for the calculus, $\mathcal{X}^{\leftrightarrow}$]

$$\begin{array}{rcccccccc} M, N ::= & \langle x.\alpha \rangle & \mid & [M\widehat{\mu}\widehat{\sigma}\,[z]\,\widehat{i}\widehat{j}N] & \mid & [\widehat{x}M\widehat{\alpha}, \widehat{z}N\widehat{\delta}].\gamma & \mid & M\widehat{\alpha} \dagger \widehat{x}N \\ & \textit{axiom} & & \textit{iff-left} & & \textit{iff-right} & & \textit{cut} \end{array}$$

The typing rules for terms of the $\mathcal{X}^{\leftrightarrow}$-calculus are given below.

Definition 6.4 [Typing rules for $\mathcal{X}^{\leftrightarrow}$]

$$\frac{}{\langle x.\alpha \rangle :\cdot \Gamma, x{:}A \vdash \alpha{:}A, \Delta}\,(Ax) \qquad \frac{M :\cdot \Gamma \vdash \alpha{:}A, \Delta \qquad N :\cdot \Gamma, x{:}A \vdash \Delta}{M\widehat{\alpha} \dagger \widehat{x}N :\cdot \Gamma \vdash \Delta}\,(Cut)$$

$$\frac{M :\cdot \Gamma \vdash \mu{:}A, \alpha{:}B, \Delta \qquad N :\cdot \Gamma, i{:}A, j{:}B \vdash \Delta}{[M\widehat{\mu}\widehat{\sigma}\,[z]\,\widehat{i}\widehat{j}N] :\cdot \Gamma, z{:}(A{\leftrightarrow}B) \vdash \Delta}\,(\leftrightarrow_L)$$

$$\frac{M :\cdot \Gamma, x{:}A \vdash \alpha{:}B, \Delta \qquad N :\cdot \Gamma, z{:}B \vdash \delta{:}A, \Delta}{[\widehat{x}M\widehat{\alpha}, \widehat{z}N\widehat{\delta}].\gamma :\cdot \Gamma \vdash \gamma{:}(A{\leftrightarrow}B), \Delta}\,(\leftrightarrow_R)$$

As remarked earlier, the iff-left term is reminiscent of a mediator with two binders over each of its subterms rather than one, and the iff-right term is reminiscent of an export, except that two ‘functions’ are available over the same interface rather than one. With this observation in mind, we move towards an encoding of the $\mathcal{X}$-calculus in $\mathcal{X}^{\leftrightarrow}$.

We can sensibly assume that when encoding the export term into an iff-right term $[\widehat{x}P\widehat{\alpha}, \widehat{z}Q\widehat{\delta}].\gamma$, we require only one of the two subterms, say P. This leaves the question of what we should do with Q. By making Q the capsule $\langle y.\delta \rangle$, we can give an encoding that is sound (no undesired reductions are possible) providing that we restrict the reduction to always use the first variant of the principal logical rule given in Definition 6.2. This does not seem a severe restriction; one might view this as a strategy on the reduction (one always has the choice of which variant of the principal iff rule to use). Our encoding is as follows.

Definition 6.5 [Interpretation of $\mathcal{X}$ into $\mathcal{X}^{\leftrightarrow}$]

$$\begin{aligned}
[\![\langle x.\alpha\rangle]\!]^{\leftrightarrow} &= \langle x.\alpha\rangle \\
[\![\widehat{x}P\widehat{\alpha}\cdot\gamma]\!]^{\leftrightarrow} &= [\widehat{x}[\![P]\!]^{\leftrightarrow}\widehat{\alpha}, \widehat{z}\langle y.\delta\rangle\widehat{\delta}].\gamma \qquad z, y, \delta \textit{ fresh} \\
[\![M\widehat{\alpha}\,[y]\,\widehat{x}N]\!]^{\leftrightarrow} &= [[\![M]\!]^{\leftrightarrow}\widehat{\alpha}\widehat{\beta}\,[y]\,\widehat{z}\widehat{x}[\![N]\!]^{\leftrightarrow}] \qquad \beta, z \textit{ fresh} \\
[\![M\widehat{\alpha}\dagger\widehat{x}N]\!]^{\leftrightarrow} &= [\![M]\!]^{\leftrightarrow}\widehat{\alpha}\dagger\widehat{x}[\![N]\!]^{\leftrightarrow}
\end{aligned}$$

Notice that in the interpretation of $\widehat{x}P\widehat{\alpha}\cdot\gamma$, had we chosen Q (the right-hand subterm) to be $\langle z.\delta\rangle$, this would have forced the types for z and δ, and therefore x and α to be the same. As a result, our encoding would not preserve typeability, since in the original term x and α need not have had the same type.

We have the following result for our encoding:

Theorem 6.6 (Preservation of typeability) *For any $\mathcal{X}$-term P, P is typeable iff $[\![P]\!]^{\leftrightarrow}$ is typeable.*

In fact, the type derivations in the two systems are closely related; one can define a further encoding from a type-derivation for P in the usual $\mathcal{X}$ system to a type-derivation for $[\![P]\!]^{\leftrightarrow}$ in the corresponding $\mathcal{X}^{\leftrightarrow}$ system. Such details are omitted here.

To show that our encoding is sensible, we must also check that we can simulate the reductions of $\mathcal{X}$. As pointed out in Section 4.3, the mechanism provided by the propagation and renaming rules is generic to any $\mathcal{X}$-style term calculus; it performs the same basic task of pushing cuts through subterms and renaming connectors regardless of the syntax employed. To show that such rules are simulated is straightforward, and we therefore only concern ourselves with the rule *exp-med* given in Definition 3.2.

The following (abbreviated) reduction confirms that we can simulate the first variant of the *exp-med* rule. The $\mathcal{X}^{\leftrightarrow}$ calculus can be extended with rules for garbage collection and renaming similar to those of Lemma 3.5.

$$\begin{aligned}
& [\![(\widehat{x}P\widehat{\alpha}\cdot\gamma)\widehat{\gamma}\dagger\widehat{y}(M\widehat{\mu}\,[y]\,\widehat{j}N)]\!]^{\leftrightarrow} \\
= \ & ([\widehat{x}[\![P]\!]^{\leftrightarrow}\widehat{\alpha}, \widehat{z}\langle c.\delta\rangle\widehat{\delta}].\gamma)\widehat{\gamma}\dagger\widehat{y}([[\![M]\!]^{\leftrightarrow}\widehat{\mu}\widehat{\sigma}\,[y]\,\widehat{i}\widehat{j}[\![N]\!]^{\leftrightarrow}]) \qquad (z, c, \delta, \sigma, i \textit{ fresh}) \\
\rightarrow \ & (([\![M]\!]^{\leftrightarrow}\widehat{\mu}\dagger\widehat{x}[\![P]\!]^{\leftrightarrow})\widehat{\sigma}\dagger\widehat{k}\langle k.\alpha\rangle)\widehat{\alpha}\dagger\widehat{z}(\langle z.\pi\rangle\widehat{\pi}\dagger\widehat{j}(\langle c.\delta\rangle\widehat{\delta}\dagger\widehat{i}[\![N]\!]^{\leftrightarrow})) \ \text{(Def. 6.2(a))} \\
\rightarrow \ & ([\![M]\!]^{\leftrightarrow}\widehat{\mu}\dagger\widehat{x}[\![P]\!]^{\leftrightarrow})\widehat{\alpha}\dagger\widehat{z}[\![N[z/j]]\!]^{\leftrightarrow} \qquad (\textit{act}\text{-R}, \textit{gc}\text{-R}, \textit{act}\text{-R}, \textit{ren}\text{-R}, \textit{act}\text{-L}, \textit{gc}\text{-L}) \\
= \ & [\![M\widehat{\mu}\dagger\widehat{x}(P\widehat{\alpha}\dagger\widehat{j}N)]\!]^{\leftrightarrow} \qquad \text{(modulo } \alpha\text{-conversion)}
\end{aligned}$$

In fact, our encoding is *only* able to simulate this variant of the *exp-med* rule. The differently-bracketed alternatives of the *exp-med* rule do not reduce to each other and also do not always share the same normal forms. However, it is understood that the set of normal forms reachable from the two variants of *exp-med* differ only in some special cases, and even then only by permutations of structure within the terms which do not affect their computational behaviour. If one were to employ a suitable notion of proof-nets for classical logic (see for example [9]), then these terms could be identified formally. In this sense, our encoding captures all of the essential computations that can be performed within $\mathcal{X}$.

The principal logical rule for iff manipulates four subterms, while the principal logical rule for any pairing connective involves three. We encoded implication by choosing one of the four subterms to be a suitable capsule. Since the iff-terms bind many combinations of inputs and outputs, we can suitably restrict them to computationally express other pairing connectives in a similar way. We are able to do this for the logical connectives $\wedge$ and $\uparrow$ up to the same limitations as discussed above for implication. Additionally, this can be achieved for the negation connective without limitations.

While the iff connective is unable to logically express the connectives $\rightarrow$, $\wedge$, $\uparrow$, $\neg$, we are able to simulate the significant computational behaviour of their corresponding term calculi. Similarly, the $\otimes$ connective is able to simulate the computational behaviour for the dual pairing connectives $-$, $\vee$, $\downarrow$ and again for the connective $\neg$.

7 Conclusions and Future Work

This work has provided an analysis of the issues involved in deriving term calculi to correspond with arbitrary choices of logical connective. We have shown various general techniques for deriving suitable syntax, reduction rules and (to some extent) computational content corresponding with the inclusion of a logical connective of interest.

The analysis of logical connectives purely in terms of the movement of their inputs and outputs seems to yield interesting results, and this should be looked at more closely. For example, we hypothesise that a term calculus can express non-terminating terms if and only if it contains a connective which can ‘swap’ an input for an output.

Our investigation into the $\leftrightarrow$ connective has shown that much more can be expressed than we first thought, and this directly relates to the inputs and outputs present. A more general investigation of the computational content of this connective (in particular, any examples which are not neatly expressed with other connectives) is the subject of future work. Our simulation result for $\mathcal{X}$ would also be strengthened by the formalisation of a suitable notion of equivalence on $\mathcal{X}$-terms, which is likely to relate to Kleene permutations and/or proof-nets.

8 Acknowledgements

We would like to thank Steffen van Bakel, Luca Cardelli, Dorian Gaertner, David Gross, Pierre Lescanne and Dragiša Žunić for many interesting discussions on the subject of this paper.

References

[1] Tristan Crolard. A formulae-as-types interpretation of subtractive logic. *Journal of Logic and Computation*, 14(4):529–570, 2004.

[2] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proc. ICFP'00*, pages 233–243. ACM, 2000.

[3] Gerhard Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1934.

[4] Hugo Herbelin. A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. In *Proc. CSL '94*, volume 933 of *LNCS*, pages 61–75. Springer, 1994.

[5] S.C. Kleene. *Introduction to Metamathematics*. North-Holland, 1952.

[6] Stéphane Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In *ENTCS*, volume 86 of *entcs*. Elsevier, 2003.

[7] P.B. Levy. Jumbo lambda-calculus. In *Proc. ICALP'06*, LNCS. Springer-Verlag, 2006.

[8] M. Parigot. An algorithmic interpretation of classical natural deduction. In *Proc. LPAR'92*, volume 624 of *LNCS*, pages 190–201. Springer-Verlag, 1992.

[9] Edmund Robinson. Proof nets for classical logic. *J. Logic Comput.*, 13(5):777–797, 2003. Special issue: Semantic foundations of proof-search.

[10] Alexander J. Summers and Steffen van Bakel. Approaches to polymorphism in classical sequent calculus. In *ESOP'06*, pages 84–99, 2006.

[11] Christian Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge, 2000.

[12] S. van Bakel, S. Lengrand, and P. Lescanne. The language $\mathcal{X}$: circuits, computations and classical logic. In *Proc. ICTCS'05*, 2005.

[13] Steffen van Bakel and Jayshan Raghunandan. Explicit alpha conversion and garbage collection in $\mathcal{X}$. Unpublished, April 2006.

[14] Philip Wadler. Call-by-value is dual to call-by-name. In *ICFP'03*, pages 189–201. ACM Press, 2003.

A Deriving iff rules using $A{\leftrightarrow}B \equiv \neg(A{\vee}B) \vee (A{\wedge}B)$

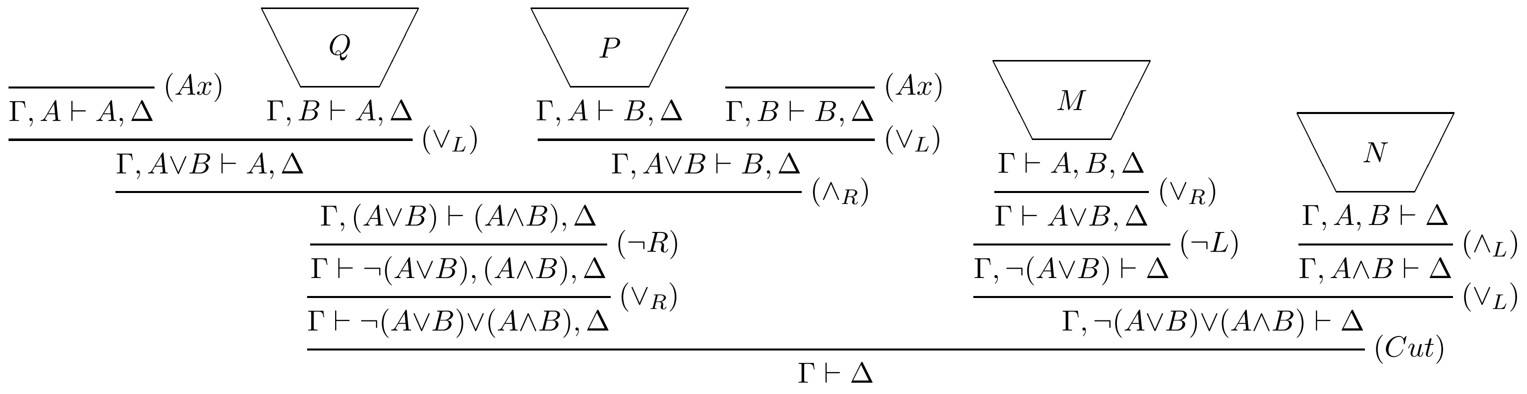
Proof of $\Gamma, \neg(A{\vee}B) \vee (A{\wedge}B) \vdash \Delta$:

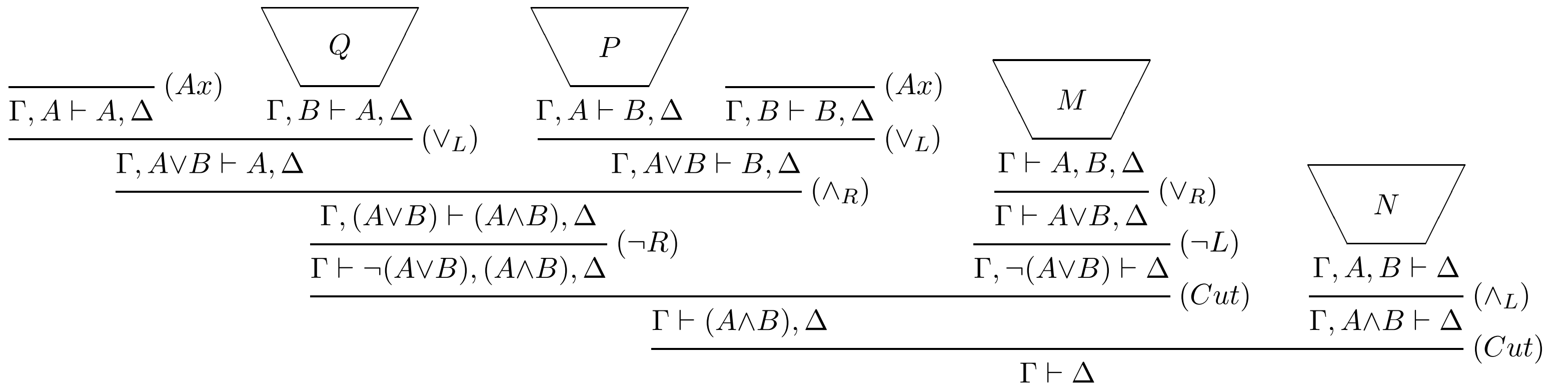
$$\dfrac{\dfrac{\dfrac{\overset{\bigtriangledown}{\Gamma \vdash A, B, \Delta}}{\Gamma \vdash A{\vee}B, \Delta}\,(\vee_R)}{\Gamma, \neg(A{\vee}B) \vdash \Delta}\,(\neg_L) \qquad \dfrac{\overset{\bigtriangledown}{\Gamma, A, B \vdash \Delta}}{\Gamma, A{\wedge}B \vdash \Delta}\,(\wedge_L)}{\Gamma, \neg(A{\vee}B) \vee (A{\wedge}B) \vdash \Delta}\,(\vee_L)$$

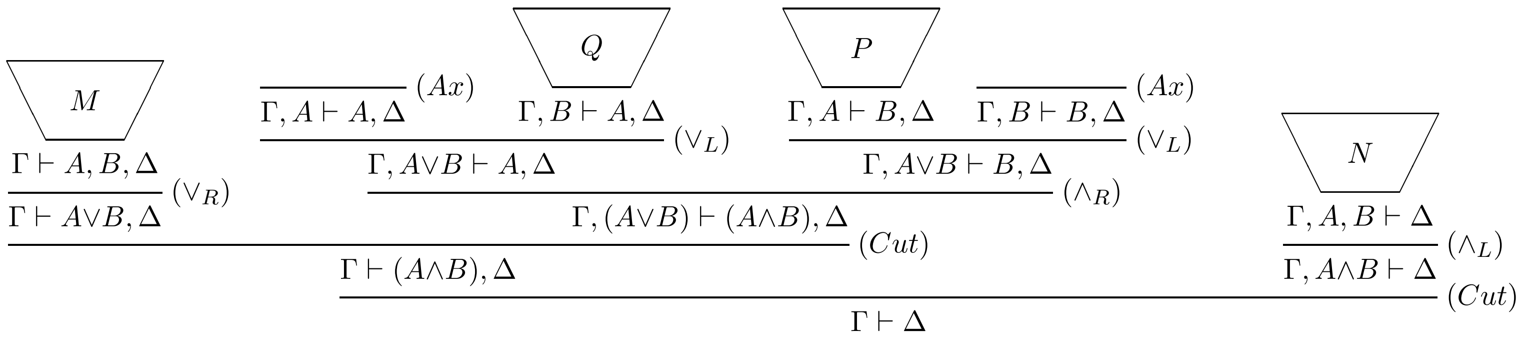
Proof of $\Gamma \vdash \neg(A{\vee}B) \vee (A{\wedge}B), \Delta$:

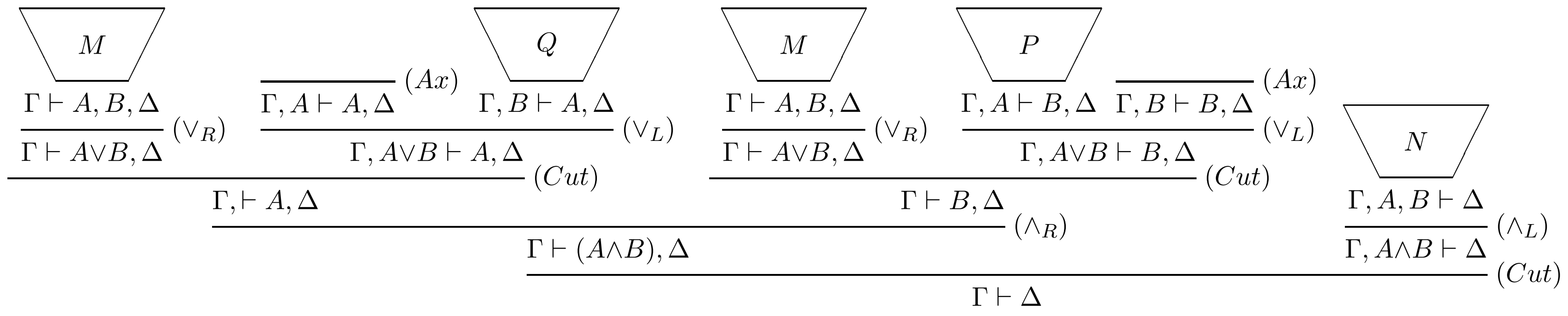
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, A \vdash A, \Delta}\,(Ax) \qquad \overset{\bigtriangledown}{\Gamma, B \vdash A, \Delta}}{\Gamma, A{\vee}B \vdash A, \Delta}\,(\vee_L) \qquad \dfrac{\overset{\bigtriangledown}{\Gamma, A \vdash B, \Delta} \qquad \dfrac{}{\Gamma, B \vdash B, \Delta}\,(Ax)}{\Gamma, A{\vee}B \vdash B, \Delta}\,(\vee_L)}{\Gamma, (A{\vee}B) \vdash (A{\wedge}B), \Delta}\,(\wedge_R)}{\Gamma \vdash \neg(A{\vee}B), (A{\wedge}B), \Delta}\,(\neg_R)}{\Gamma \vdash \neg(A{\vee}B) \vee (A{\wedge}B), \Delta}\,(\vee_R)$$

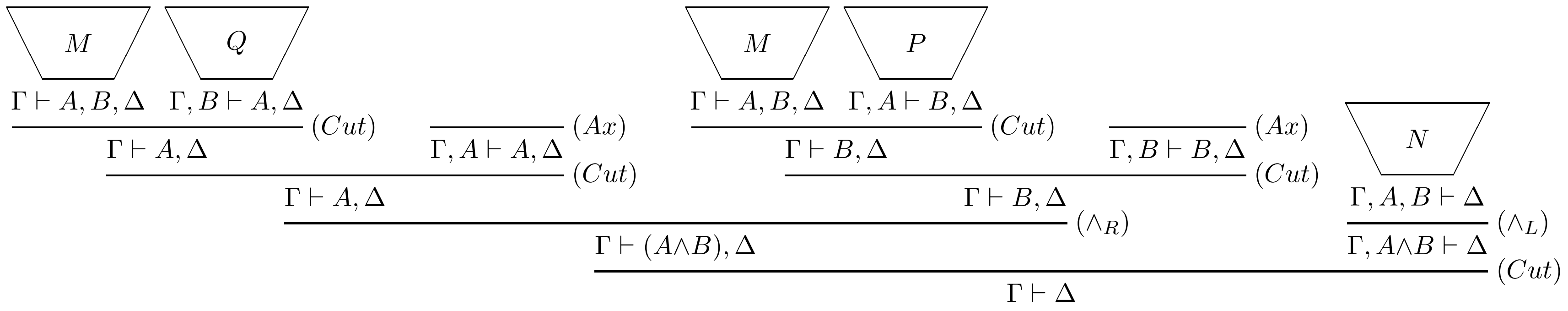
B Deriving Cut Elimination Rules for iff

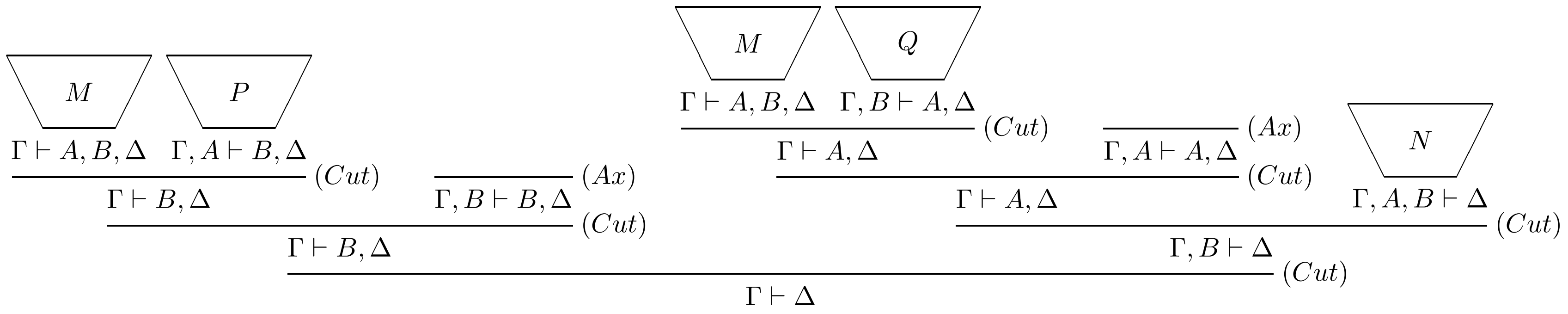

Q
P
M
N
$\Gamma, A \vdash A, \Delta$ (Ax)
$\Gamma, B \vdash A, \Delta$
$\Gamma, A \vdash B, \Delta$
$\Gamma, B \vdash B, \Delta$ (Ax)
$\Gamma, A\vee B \vdash A, \Delta$ $(\vee_L)$
$\Gamma, A\vee B \vdash B, \Delta$ $(\vee_L)$
$\Gamma, (A\vee B) \vdash (A\wedge B), \Delta$ $(\wedge_R)$
$\Gamma \vdash \neg(A\vee B), (A\wedge B), \Delta$ $(\neg R)$
$\Gamma \vdash \neg(A\vee B)\vee(A\wedge B), \Delta$ $(\vee_R)$
$\Gamma \vdash A, B, \Delta$
$\Gamma \vdash A\vee B, \Delta$ $(\vee_R)$
$\Gamma, \neg(A\vee B) \vdash \Delta$ $(\neg L)$
$\Gamma, A, B \vdash \Delta$
$\Gamma, A\wedge B \vdash \Delta$ $(\wedge_L)$
$\Gamma, \neg(A\vee B)\vee(A\wedge B) \vdash \Delta$ $(\vee_L)$
$\Gamma \vdash \Delta$ (Cut)


Q
P
M
N
$\Gamma, A \vdash A, \Delta$ (Ax)
$\Gamma, B \vdash A, \Delta$
$\Gamma, A \vdash B, \Delta$
$\Gamma, B \vdash B, \Delta$ (Ax)
$\Gamma, A\vee B \vdash A, \Delta$ $(\vee_L)$
$\Gamma, A\vee B \vdash B, \Delta$ $(\vee_L)$
$\Gamma, (A\vee B) \vdash (A\wedge B), \Delta$ $(\wedge_R)$
$\Gamma \vdash \neg(A\vee B), (A\wedge B), \Delta$ $(\neg R)$
$\Gamma \vdash A, B, \Delta$
$\Gamma \vdash A\vee B, \Delta$ $(\vee_R)$
$\Gamma, \neg(A\vee B) \vdash \Delta$ $(\neg L)$
$\Gamma \vdash (A\wedge B), \Delta$ (Cut)
$\Gamma, A, B \vdash \Delta$
$\Gamma, A\wedge B \vdash \Delta$ $(\wedge_L)$
$\Gamma \vdash \Delta$ (Cut)


M
Q
P
N
(Ax)
(Ax)
Γ, A ⊢ A, Δ
Γ, B ⊢ A, Δ
Γ, A ⊢ B, Δ
Γ, B ⊢ B, Δ
(∨_L)
(∨_L)
Γ ⊢ A, B, Δ
Γ, A∨B ⊢ A, Δ
Γ, A∨B ⊢ B, Δ
(∨_R)
(∧_R)
Γ ⊢ A∨B, Δ
Γ, (A∨B) ⊢ (A∧B), Δ
Γ, A, B ⊢ Δ
(Cut)
(∧_L)
Γ ⊢ (A∧B), Δ
Γ, A∧B ⊢ Δ
(Cut)
Γ ⊢ Δ


M
Q
M
P
N
(Ax)
(Ax)
Γ ⊢ A, B, Δ
Γ, A ⊢ A, Δ
Γ, B ⊢ A, Δ
Γ ⊢ A, B, Δ
Γ, A ⊢ B, Δ
Γ, B ⊢ B, Δ
(∨_R)
(∨_L)
(∨_R)
(∨_L)
Γ ⊢ A∨B, Δ
Γ, A∨B ⊢ A, Δ
Γ ⊢ A∨B, Δ
Γ, A∨B ⊢ B, Δ
(Cut)
(Cut)
Γ,⊢ A, Δ
Γ ⊢ B, Δ
Γ, A, B ⊢ Δ
(∧_R)
(∧_L)
Γ ⊢ (A∧B), Δ
Γ, A∧B ⊢ Δ
(Cut)
Γ ⊢ Δ


M
Q
M
P
N
$\Gamma \vdash A, B, \Delta$
$\Gamma, B \vdash A, \Delta$
(Cut)
(Ax)
$\Gamma \vdash A, B, \Delta$
$\Gamma, A \vdash B, \Delta$
(Cut)
(Ax)
$\Gamma \vdash A, \Delta$
$\Gamma, A \vdash A, \Delta$
(Cut)
$\Gamma \vdash B, \Delta$
$\Gamma, B \vdash B, \Delta$
(Cut)
$\Gamma \vdash A, \Delta$
$\Gamma \vdash B, \Delta$
$(\wedge_R)$
$\Gamma, A, B \vdash \Delta$
$(\wedge_L)$
$\Gamma \vdash (A \wedge B), \Delta$
$\Gamma, A \wedge B \vdash \Delta$
(Cut)
$\Gamma \vdash \Delta$


M
Q
M
P
N
$\Gamma \vdash A, B, \Delta$
$\Gamma, B \vdash A, \Delta$
(Cut)
(Ax)
$\Gamma \vdash A, B, \Delta$
$\Gamma, A \vdash B, \Delta$
(Cut)
(Ax)
$\Gamma \vdash A, \Delta$
$\Gamma, A \vdash A, \Delta$
(Cut)
$\Gamma \vdash B, \Delta$
$\Gamma, B \vdash B, \Delta$
(Cut)
$\Gamma \vdash A, \Delta$
$\Gamma, A, B \vdash \Delta$
(Cut)
$\Gamma \vdash B, \Delta$
$\Gamma, B \vdash \Delta$
(Cut)
$\Gamma \vdash \Delta$