Notre Dame Journal of Formal Logic
Volume 50, Number 3, 2009

Proof-finding Algorithms for Classical and Subclassical Propositional Logics

M. W. Bunder and R. M. Rizkalla

Abstract The formulas-as-types isomorphism tells us that every proof and theorem, in the intuitionistic implicational logic $H_{\rightarrow}$, corresponds to a lambda term or combinator and its type. The algorithms of Bunder very efficiently find a lambda term inhabitant, if any, of any given type of $H_{\rightarrow}$ and of many of its subsystems. In most cases the search procedure has a simple bound based roughly on the length of the formula involved. Computer implementations of some of these procedures were done in Dekker. In this paper we extend these methods to full classical propositional logic as well as to its various subsystems. This extension has partly been implemented by Oostdijk.

1 Introduction

The Curry-Howard or formulas-as-types isomorphism allows proofs and theorems of intuitionistic implicational logic to be represented as combinators or lambda-terms and their types. The work of Trigg, Hindley, and Bunder [12] shows that, for a large class of weaker logics, proofs can be represented by restricted classes of lambda-terms or combinators.

In Bunder [3], this work was used to develop proof-finding algorithms for these implicational logics. Most of these algorithms have a bound on the number of algorithm steps required to produce a λ-term proof, or a guarantee that there is no proof, based on the structure of the formula being examined.

In this paper, we extend these algorithms to many logics with additional connectives. This is done, first, by simply adding a constant type f and some definitions. Intuitionistic versions of the relevance logics $R_{\rightarrow}$ and $T_{\rightarrow}$ are among the logics to which these algorithms apply. Algorithms for stronger logics are obtained when we add a "combinator" **F** with $f \rightarrow \alpha$ as its type, for every α. Independently, we can

Received June 25, 2007; accepted April 26, 2009; printed September 30, 2009
2000 Mathematics Subject Classification: Primary, 03B20, 03B35, 03B40, 03B47
Keywords: proof-finding algorithms, propositional logics
© 2009 by University of Notre Dame 10.1215/00294527-2009-011

add a “combinator” **P** with Peirce’s Law as its type, or, equivalently, a new operator ν. Again we develop proof-finding algorithms for these logics. The strongest one covered is classical propositional logic, which can be based on the types of **F** and **P** and of the standard combinators **K** and **S**.

2 Q-Combinators and Restricted Lambda Terms

Combinators are operators which manipulate arbitrary expressions by cancellation, duplication, bracketing, and permutation. Given any finite (basis-) set Q of combinators, Q-combinators are defined as follows.

Definition 2.1

(i) If $X \in Q$, X is a Q-combinator.
(ii) If X and Y are Q-combinators, so is (XY).

The operation in (ii) is called *application.* In terms formed by application we assume association to the left. In particularly complex terms we will sometimes use { } or [ ] instead of ( ).

The most common basis of combinators is $\{\mathbf{S}, \mathbf{K}\}$, where **S** and **K** have the properties

$$\mathbf{K}XY = X$$
$$\mathbf{S}XYZ = XZ(YZ).$$

Combinatory reduction involves replacing $\mathbf{K}XY$ and $\mathbf{S}XYZ$, in terms, by X and $XZ(YZ)$, respectively.

$\{\mathbf{S}, \mathbf{K}\}$-combinators, or more simply **SK**-combinators, have the “combinatorial completeness” property,

If $A(X_1, \ldots, X_n)$ is formed by application, with zero or more copies of each of $X_1, \ldots, X_n$, then there is an **SK**-combinator **A** such that

$$\mathbf{A}X_1 \ldots X_n = A(X_1, \ldots, X_n).$$

Lambda(or λ)-terms are defined, using a set of variables V, as follows.

Definition 2.2

(i) If $x \in V$, x is a λ-term.
(ii) If X and Y are λ-terms, so is (their application) (XY).
(iii) If X is a λ-term and x a variable, then $(\lambda x.X)$ (called a λ-abstract) is a λ-term.

We assume association to the left in λ-terms formed by application.

Definition 2.3

(i) If x occurs in Y and $\lambda x.Y$ is (a subterm of) X, then that occurrence of x is *bound* in X.
(ii) If an occurrence of x in X is not bound in X, then that occurrence is *free* in X.

The main rule of λ-calculus is

(β) $(\lambda x.X)Y = [Y/x]X$

where in $[Y/x]X$, Y replaces all free occurrences of x in X in a capture-avoiding manner. Replacing $(\lambda x.X)Y$ by $[Y/x]X$ in a term is called *β-reduction.*

Note that (β) gives combinatorial completeness in the λ-calculus. The **A** above becomes $(\lambda x_1.(\lambda x_2 \ldots (\lambda x_n.A(x_1,\ldots,x_n))\ldots))$, which we will write more briefly as

$$\lambda x_1 \ldots x_n.A(x_1,\ldots,x_n).$$

More details on combinatory logic and the λ-calculus, including the formal definition of $[Y/x]X$, can be found in Hindley and Seldin [9].

We will restrict λ-terms by restricting Definition 2.2(iii). For example, if x must be free in X, the λ-terms defined are the λI-terms of Church. Other restrictions used in [11] depend on where x is in X or how many occurrences there are of x in X.

SK-combinatory logic is, in a sense, equivalent to the full λ-calculus (see [2]). In a similar way, many Q-combinatory logics have been shown to be equivalent to certain restricted λ-calculi. Details appear in Trigg, Hindley, and Bunder [12].

3 Types, Implicational Logics, Combinators, and Lambda Terms

Many combinators and λ-terms represent functions. For example, if $x \in \alpha$ and $y \in \beta$, where α and β can be thought of as sets, given the property of **K**, $\mathbf{K}x : \beta \to \alpha$; that is, $\mathbf{K}x$ is a function from β to α, or, equivalently, an element of the set $\beta \to \alpha$. Then also $\mathbf{K} : \alpha \to (\beta \to \alpha)$. The set $\alpha \to (\beta \to \alpha)$ is then called a type of **K**.

Types, for combinators and λ-terms, are defined as follows using a countable set of *atoms* or *atomic types* A.

Definition 3.1

(i) If $a \in A$, a is a type.
(ii) If α and β are types, so is $(\alpha \to \beta)$.

For types we assume association to the right.

In this paper we will not be much concerned with combinatory or λ-reduction (which via Curry-Howard corresponds to proof reduction) or with how types are assigned to combinators using their properties. We take each combinator from a set Q, with its type as an axiom scheme.

Definition 3.2 Q-type theory has, for each of the combinators **I**, **B**, **B′**, **K**, **C**, **S**, **S′**, and **W** that appears in Q, the corresponding axiom scheme from

$$\begin{array}{lcl}
\vdash \mathbf{I} & : & \alpha \to \alpha \\
\vdash \mathbf{B} & : & (\alpha \to \beta) \to (\gamma \to \alpha) \to \gamma \to \beta \\
\vdash \mathbf{B}' & : & (\alpha \to \beta) \to (\beta \to \gamma) \to \alpha \to \gamma \\
\vdash \mathbf{K} & : & \alpha \to \beta \to \alpha \\
\vdash \mathbf{C} & : & (\alpha \to \beta \to \gamma) \to \beta \to \alpha \to \gamma \\
\vdash \mathbf{S} & : & (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma \\
\vdash \mathbf{S}' & : & (\alpha \to \beta) \to (\alpha \to \beta \to \gamma) \to \alpha \to \gamma \\
\vdash \mathbf{W} & : & (\alpha \to \alpha \to \beta) \to \alpha \to \beta.
\end{array}$$

The sole rule of inference is

$$\to_e \qquad \frac{\vdash X : \alpha \to \beta \qquad \vdash Y : \alpha}{\vdash (XY) : \beta}\ .$$

The combinators in Q can be considered as labels of axioms of a Q-logic and the Q-combinators as labels of theorems. In fact, a Q-combinator is a very concise representation of a Hilbert-style proof of a theorem which is its type. Each combinator in Q that appears in it represents a use of an axiom and each application (forming an (XY)) a use of $\to_e$.

Example 3.3

$$\begin{array}{lcl} \vdash \mathbf{SK} & : & (a \to a \to a) \to a \to a \\ \vdash \mathbf{SKK} & : & a \to a\ . \end{array}$$

Types can be assigned to λ-terms using the type assignment system for λ-calculus, TA_λ.

Definition 3.4 The rules of TA_λ are given by

$$\to_e \qquad \dfrac{\Delta \vdash X : \alpha \to \beta \qquad \Delta' \vdash Y : \alpha}{\Delta, \Delta' \vdash (XY) : \beta}$$

where each of Δ and Δ' is a set of judgments of the form $x : \gamma$ and Δ, Δ' is $\Delta \cup \Delta'$.

$$\Delta, x : \alpha \vdash x : \alpha \qquad \text{where } x \text{ is not in } \Delta$$

and

$$\to_i \qquad \dfrac{\Delta, x : \alpha \vdash X : \beta}{\Delta \vdash \lambda x.X : \alpha \to \beta} \qquad \text{where } x \text{ is not in } \Delta\,.$$

If in a combinatory logic or λ-calculus-based type theory $\vdash X : \tau$, then X is called an *inhabitant* of τ and τ a *type* of X in the given type theory.

Each λ-term inhabitant of a type represents a natural deduction proof of the type, each application represents a use of $\to_e$, and each λ-abstraction a use of $\to_i$.

Example 3.5

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x_1 : \alpha \to \beta \vdash x_1 : \alpha \to \beta \quad x_2 : \alpha \vdash x_2 : \alpha}{x_1 : \alpha \to \beta, x_2 : \alpha \vdash x_1 x_2 : \beta} \qquad x_3 : \beta \to \gamma}{x_1 : \alpha \to \beta, x_2 : \alpha, x_3 : \beta \to \gamma \vdash x_3(x_1 x_2) : \gamma}}{x_1 : \alpha \to \beta, x_2 : \alpha \vdash \lambda x_3.x_3(x_1 x_2) : (\beta \to \gamma) \to \gamma}}{x_1 : \alpha \to \beta \vdash \lambda x_2 x_3.x_3(x_1 x_2) : \alpha \to (\beta \to \gamma) \to \gamma}}{\vdash \lambda x_1 x_2 x_3.x_3(x_1 x_2) : (\alpha \to \beta) \to \alpha \to (\beta \to \gamma) \to \gamma}$$

Details on assigning types to λ-terms can be found in Hindley [8]. The book also includes an algorithm of Ben-Yelles, which finds λ-term inhabitants for types, that is, proofs of potential theorems of $H_\to$.

In [3], we gave proof-finding algorithms for each of the following implicational logics (named by their axioms): **BCK**-logic, **BCI**-logic, **BCIW**-logic (the relevance logic $R_\to$), **BB′IW**-logic ($T_\to$), **SK**-logic ($H_\to$), **BB′IK**-logic, and **BB′I**-logic; we indicated how algorithms could be found for a number of others. For any Q-logic this involved finding a natural deduction or λ-term proof of a form that is translatable into a Q-combinator via the work in [2]. In most cases a fixed bound on the number of λ-terms the algorithm would need to create to find the proof term, or to prove there was none, was given. This bound is based on the form of the potential theorem being tested. In the cases of **BCIW** and **BB′IW** there was no simple bound, but the finiteness of the procedure could be guaranteed. In each case the Q-combinator, that is, Hilbert-style proof, can be found.

Our aim will be to extend the (sub)intuitionistic proof algorithms of [3], for the above implicational logics, to provide us with proofs of theorems involving connectives other than $\to$. We also consider various strengthenings of the above logics and the corresponding type theories.

Note that several combinators are interdefinable. All combinators are definable in terms of **S** and **K** or (**B, C, K**, and **W**) and as $\mathbf{B}' = \mathbf{CB}, \mathbf{I} = \mathbf{SKK} = \mathbf{CKK}$, $\mathbf{S} = \mathbf{B}(\mathbf{BW})(\mathbf{B}(\mathbf{B}'\mathbf{B}')\mathbf{B})$, and equal combinators have the same types, it follows that **SK**-logic is the same as $\mathbf{BB}'\mathbf{WK}$-logic, and so on.

4 Logics with Minimal Negation

We pick an atomic type f from A to define minimal negation.

Definition 4.1 $\sim \alpha \equiv \alpha \to f$.

Definition 4.2 A Q-logic (Q-type theory), which includes connectives defined using f (e.g., negation, as defined in Definition 4.1), will be called a Q-logic (Q-type theory) with minimal negation, or an MQ-logic (MQ-type theory).

As f has only the properties possessed by other atomic types, such as being uninhabited, derivations in MQ-type theories are exactly the same as those in Q-type theories; however, we will use $\vdash_{\mathrm{MQ}}$ for Q-provability where the types may include a connective defined using f. Hence we have the following.

Proof-finding algorithm for MQ-logics

Aim Given a type τ, to find an X such that $\vdash_{\mathrm{MQ}} X : \tau$.

Method

1. Rewrite τ in terms of $\to$ and f using the definitions of any other connectives.
2. Find X using the Q-algorithm of [3].

Theorem 4.3 *If τ is a type, which may include connectives definable using f, then τ is inhabited in* MQ *if and only if an X such that $\vdash_{\mathrm{MQ}} X : \tau$ is found by the algorithm for* MQ.

Proof For each Q this follows from the appropriate theorem for Q in [3]. □

Here are some results, easily derivable using this algorithm, for MQ-logics.

Proposition 4.4

(i) *In* MQ-*logics with* $\mathbf{B}'$ *as a theorem,* $\vdash_{\mathrm{MQ}} (a \to b) \to \sim b \to \sim a$.

(ii) *In* MQ-*logics with* **C** *as a theorem,* $\vdash_{\mathrm{MQ}} (a \to \sim b) \to b \to \sim a$.

(iii) *In* MQ-*logics with* **W** *as a theorem,* $\vdash_{\mathrm{MQ}} (a \to \sim a) \to \sim a$.

(iv) *In* MQ-*logics with* **C** *and* **I** *as theorems,* $\vdash_{\mathrm{MQ}} a \to \sim\sim a$.

We will now consider several other possible connectives definable using f.

Definition 4.5

$$\alpha \& \beta \equiv (\alpha \to \beta \to f) \to f$$
$$\alpha \wedge \beta \equiv ((\alpha \to \beta) \to \alpha \to f) \to f$$
$$\alpha \vee \beta \equiv (\alpha \to f) \to \beta$$
$$\alpha \vee_1 \beta \equiv (\alpha \to \beta) \to \beta$$
$$\Box \alpha \equiv \alpha \to \alpha \to \alpha.$$

Proofs involving these connectives are rather less trivial. We give the λ-terms generated by Dekker's Brouwer 7.9.0 [6]. The algorithms used by Dekker, that is, those of [3], only generate λ-terms which, via the work in [2], are translatable into Hilbert-style logics.

Proposition 4.6

(i) *In* MQ-*logics with* **K** $\vdash_{\mathrm{MQ}} \alpha \to \beta \vee \alpha$.

(ii) *In* MQ-*logics with* **B**,**C**,**I** *and* **W** $\vdash_{\mathrm{MQ}} (\alpha \to \beta)\&(\alpha \to \gamma) \to \alpha \to \beta\&\gamma$.

(iii) *In* MQ-*logics with* **S** *and* **K** $\vdash_{\mathrm{MQ}} (\alpha \to \beta) \wedge (\alpha \to \gamma) \to \alpha \to \beta \wedge \gamma$.

(iv) *In* MQ-*logics with* **B** *and* **B**′ $\vdash_{\mathrm{MQ}} \Box\alpha \,\&\, \Box\beta \to \Box(\alpha\&\beta)$.

(v) *In* MQ-*logics with* **B**,**C** *and* **I** $\vdash_{\mathrm{MQ}} \Box\alpha \wedge \Box\beta \to \Box(\alpha \wedge \beta)$.

(vi) *In* MQ-*logics with* **B**,**C** *and* **K** $\vdash_{\mathrm{MQ}} \alpha \to \beta \to \alpha \wedge \beta$.

(vii) *In* MQ-*logics with* **B**,**C** *and* **I** $\vdash_{\mathrm{MQ}} \alpha \to \beta \to \alpha\&\beta$.

(viii) *In* MQ-*logics with* **C** *and* **B** *or* **B**′ $\vdash_{\mathrm{MQ}} \alpha \wedge \beta \to \beta \wedge \alpha$.

(ix) *In* MQ-*logics with* **C** *and* **W** $\vdash_{\mathrm{MQ}} \alpha \to \alpha\&\alpha$.

(x) *In* MQ-*logics with* **C** *and* **I** $\vdash_{\mathrm{MQ}} \alpha \to \alpha \wedge \alpha$.

(xi) *In* MQ-*logics with* **B** *and* **B**′ $\vdash_{\mathrm{MQ}} (\alpha \to \beta) \to (\alpha \vee_1 \gamma) \to (\beta \vee_1 \gamma)$.

(xii) *In* MQ-*logics with* **B**,**C**,**I** *and* **W** $\vdash_{\mathrm{MQ}} \alpha \wedge (\beta \vee \gamma) \to \alpha \wedge \beta \vee \alpha \wedge \gamma$.

(xiii) *In* MQ-*logics with* **S** *and* **K** $\vdash_{\mathrm{MQ}} \alpha\&(\beta \vee \gamma) \to \alpha\&\beta \vee \alpha\&\gamma$.

(xiv) *In* MQ-*logics with* **B**,**C**,**I** *and* **W** $\vdash_{\mathrm{MQ}} (\alpha \to \beta)\&(\alpha \to \gamma) \to \alpha \to \beta\&\gamma$.

Proof

(i) $\lambda x_1x_2.x_1$.

(ii) $\lambda x_1x_2x_3.x_1(\lambda x_4x_5.x_3(x_4x_2)(x_5x_2))$.

(iii) $\lambda x_1x_2x_3.x_1(\lambda x_4x_5.x_3(\lambda x_6.x_4x_5x_2)(x_5x_2))$.

(iv) $\lambda x_1x_2x_3x_4.x_1(\lambda x_5x_6.x_3(\lambda x_7x_8.x_2(\lambda x_9x_{10}.x_4(x_5x_7x_9)(x_6x_8x_{10}))))$.

(v) $\lambda x_1x_2x_3x_4.x_2(\lambda x_5x_6.x_3(\lambda x_7x_8.x_1(\lambda x_9x_{10}.x_4(\lambda x_{11}.x_9x_{10}(x_5x_6)(x_7x_{11}))x_8)$.

(vi) $\lambda x_1x_2x_3.x_3(\lambda x_1.x_2)x_1$.

(vii) $\lambda x_1x_2x_3.x_3x_1x_2$.

(viii) $\lambda x_1x_2.x_1.(\lambda x_3x_4.x_2x_4x_3)$.

(ix) $\lambda x_1x_2.x_2x_1x_1$.

(x) $\lambda x_1x_2.x_2(\lambda x_1.x_1)x_1$.

(xi) $\lambda x_1x_2x_3.x_2(\lambda x_4.x_3(x_1x_4))$.

(xii) $\lambda x_1x_2x_3.x_1(\lambda x_5x_4.x_2(\lambda x_6.x_3x_5(\lambda x_6.x_4(x_6x_5))))$.

(xiii) $\lambda x_1x_2x_3.x_1(\lambda x_4x_5.x_2(\lambda x_6.x_3(\lambda x_5.x_4x_5(\lambda x_7.x_6(\lambda x_5.x_7)x_5))x_5))$.

(xiv) $\lambda x_1x_2x_3.x_1(\lambda x_4x_5.x_3(x_4x_2)(x_5x_2))$. □

The combinatory versions of the proofs, which may be complex, are also given in Oostdijk's (partial) implementation of the algorithms. The proof in (ii), for example, in **BCIW**-logic, translates to **C(BBB)**$\{$**B(C(B[BW(B(BC(BB)))]B)(CI)**$\}$. The proofs of (iii) to (v) are more complex still.

Notation 4.7 Whereas in the above theorem and others in this paper we talk of logics "with the combinators $\mathbf{C_1}, \ldots, \mathbf{C_n}$," all that is really needed is to have combinators with the *types* of $\mathbf{C_1}, \ldots, \mathbf{C_n}$. The lambda terms given in the proofs can be translated in terms of the combinators and the theorems are proved, by $\rightarrow_e$, from their types. Hence, other combinators with the same types will do.

The algorithms can also show that many theorems are unprovable. For example, in **SK**,

$$\begin{array}{rl}
\sim a \rightarrow a \rightarrow b & -(a) \\
(\sim a \rightarrow \sim b) \rightarrow b \rightarrow a & -(b) \\
\sim\sim a \rightarrow a & -(c) \\
a \vee a \rightarrow a & -(d) \\
a \rightarrow a \vee b & -(e) \\
a \& b \rightarrow a & -(f) \\
a \wedge b \rightarrow a. & -(g).
\end{array}$$

5 Logics with Implication and Intuitionistic Negation

In this section we extend the type theory to include one extra axiom.

Axiom F $\vdash \mathbf{F} : f \rightarrow \alpha.$

We then have the following definition.

Definition 5.1 **QF**-type theory is MQ-type theory with Axiom **F**. **QF**-logic is MQ-logic with the extra axiom $\vdash f \rightarrow \alpha$.

We will treat **F** like a combinator, or label representing this axiom, in our Hilbert-style proofs as well as in the natural deduction proofs. We therefore have **BCIF**-logic, **SKF**-logic, and so on.

The following theorem will justify the proof-finding algorithm for such logics.

Theorem 5.2 *If τ is a type whose subtypes are $\alpha_1, \ldots, \alpha_n$, then $\vdash_{\mathrm{QF}} X : \tau$ if and only if there is a Q-term Y such that*

(i) $FV(Y) = \{y_1, \ldots, y_n\}$,

(ii) $y_1 : f \rightarrow \alpha_1, y_2 : f \rightarrow \alpha_2, \ldots, y_n : f \rightarrow \alpha_n \vdash_{\mathrm{Q}} Y : \tau,$

where, for the purpose of checking whether Y is a Q-term, the y_is are treated as constants.

(iii) $[\mathbf{F}/y_1, \ldots, \mathbf{F}/y_n]Y \equiv X.$

Proof ($\Rightarrow$) In a cut-free proof of $\vdash_{\mathrm{QF}} X : \tau$, by the subformula property, we can replace each use of Axiom **F** by a hypothesis $y_i : f \rightarrow \alpha_i$, where α_i is a subtype of τ. This gives rise to a term Y which clearly satisfies (i) and (ii). $[\mathbf{F}/y_1, \ldots, \mathbf{F}/y_n]Y \equiv X$ as this operation restores the original **F**s, so (iii) holds.

($\Leftarrow$) If, in the derivation of (ii), we replace each hypothesis $y_i : f \rightarrow \alpha_i$ by an axiom $\mathbf{F} : f \rightarrow \alpha_i$, we obtain a proof of $\vdash_{\mathrm{QF}} X : \tau$. □

Proof-finding algorithm for logics with implication and intuitionistic negation

Aim Given a type τ, to find, for a basis **QF**, an X such that $\vdash_{\mathrm{QF}} X : \tau$.

Method

1. Rewrite τ in terms of $\rightarrow$ and f.

2. Use the Q-algorithm of [3], while treating $y_1, \ldots, y_n$ as constants, to find a Y such that

$$y_1 : f \to \alpha_1, \ldots, y_n : f \to \alpha_n \vdash_Q Y : \tau$$

where $\alpha_1, \ldots, \alpha_n$ are the subtypes of τ. If there is no such Y, there is no X.
3. If there is such a Y,

$$X = [\mathbf{F}/y_1, \ldots, \mathbf{F}/y_n]Y.$$

Using the algorithm and translation into combinators, we have the following proposition.

Proposition 5.3
(i) $\vdash \mathbf{BF} : \sim a \to a \to b.$
(ii) $\vdash \mathbf{C(BF)} : a \to \sim a \to b.$
(iii) $\vdash \mathbf{C(BF)} : a \to a \vee b.$

We also have $\vdash \mathbf{BF} : \sim a \to a \to b \to c, \vdash \mathbf{C(BF)} : (a \to c) \to (a \to c) \vee (b \to c)$ and so on.

For logics that have **B** and **K** we can replace Axiom **F** by a simpler axiom as a result of the next lemma.

Lemma 5.4 *In any logic with* **B**, **K**, *and* **F**, *the only uses of the* **F** *axiom that are required are ones where* α *is an atom.*

Proof An arbitrary type α must have the form $\alpha_1 \to \cdots \to \alpha_n \to a$ for some $n \geq 0$. In a logic with **B** and **K** we have, for $1 \leq i \leq n$,

$$\vdash_{\mathbf{QF}} (f \to \alpha_{i-1} \to \cdots \to \alpha_1 \to a) \to f \to \alpha_i \to \cdots \to \alpha_1 \to a,$$

so from $\vdash_{\mathbf{QF}} f \to a$ we can prove $\vdash_{\mathbf{QF}} f \to \alpha$.

In the corresponding **QF**-type theory, from $\vdash_{\mathbf{QF}} \mathbf{F} : f \to a$ we can prove

$$\vdash_{\mathbf{QF}} \mathbf{BK}(\ldots(\mathbf{BKF})\ldots) : f \to \alpha,$$

where there are n **BK**s. So in the logic with the restricted Axiom **F**, we have the same theorems but more complex proofs. The algorithm, however, can be simplified as $\alpha_1, \ldots, \alpha_n$ can be the atoms in τ. □

Most standard tautologies, other than those in Propositions 4.6 and 5.3, in particular, (b), (c), (d), (f), and (g), remain unprovable in **SKF**-logic as we do not have double negation, or equivalently, Peirce's Law.

6 Implicational Logics with Peirce's Law (and Minimal Negation)

We can extend the Hilbert-style logics, or typed combinatory logics, to classical logics by adding one extra axiom.

Axiom P $\vdash \mathbf{P} : ((\alpha \to \beta) \to \alpha) \to \alpha.$

P is treated as a new combinator. Typed lambda calculus is extended by means of a new operator ν with the introduction rule:

$$\nu_I \quad \frac{\Delta, x : \alpha \to \beta \vdash Y : \alpha}{\Delta \vdash \nu x.Y : \alpha}$$

The corresponding logical rule was first proposed in Popper [11] and is also used in Curry [5]. Versions similar to the above typed rule (one with α and one with β at the end of the top line) appear in Gabbay and de Queroz [7] but have λx instead of νx, which causes problems when translating from λ-terms to combinators. Gabbay and de Queroz also have the equivalent of Axiom **F** but do not use the axiom or the "ν_I-rule" to build proof-finding algorithms. Work on the properties of the ν-operator can be found in Bunder and Hirokawa [4]. Translations from **QP**-combinators (with or without **F**) to $\lambda\nu$-terms and vice versa are given by

$$\mathbf{P} = \lambda y.\nu x.yx$$
$$\nu x.Y = \mathbf{P}(\lambda x.Y).$$

The number of logics that we will be interested in is greatly reduced because, with Axiom **P**, the type of **W** can be obtained using Axioms **B′** and **K** and that of **K** can be obtained using Axioms **B**, **C**, and **I**:

$$\vdash \mathbf{BPB'} : (a \to a \to b) \to a \to b \qquad (\mathbf{B'} = \mathbf{CB})$$
$$\vdash \mathbf{B(BP)(BC(CI))} : a \to b \to a.$$

Thus **BCIP, BCKP, BCIWP**, and **SKP** logics are all the same.

Also **BB′IKP** and **BB′KP** logics are the same as **SKP** logic as

$$\vdash \mathbf{B(BW)(B(B'(BB'K(B')} : (a \to b \to c) \to b \to a \to c$$

and

$$\vdash \mathbf{WK} : a \to a$$

have the same types as **C** and **I**, respectively.

Based on the lemma below, we have a proof-finding algorithm for **SKP**-logic, that is, classical implicational logic, but not at this stage for the remaining weaker systems, **BB′P** and **BB′IP**. We can, of course, add minimal negation and the definitions of the other connectives.

Lemma 6.1 *A proof of τ in* **SKP**-*logic can be written as a proof in* **SK**-*logic of τ using the hypotheses*

$$\tau \to a_1, \ldots, \tau \to a_n$$

where $a_1, \ldots, a_n$ are some or all of the atoms of τ, followed by ν_I steps.

Proof Consider the last proof segments ending in a ν_I step in a proof tree of τ. These will take the form

$$\begin{array}{cc} & \Delta_i, \alpha_i \to \beta_{i1} \to \cdots \to \beta_{ik_i} \to b_i \vdash \alpha_i \to \beta_{i1} \to \cdots \to \beta_{ik_i} \to b_i \\ & D_{i1} \\ \nu_I & \dfrac{\Delta'_i, \alpha_i \to \beta_{i1} \to \cdots \to \beta_{ik_i} \to b \vdash \alpha_i}{\Delta'_i \vdash \alpha_i} \end{array}$$

where $1 \le i \le m$.

These are then used in a part D_2 of the proof tree of τ. Each D_{i1} may contain further uses of ν_I but the remainder of D_2 will contain none.

Now consider a proof tree D'_2 in which, for each i, $\Delta'_i \vdash \alpha_i$ and the steps above it are replaced by

$$\Delta'_i, \beta_{i1}, \ldots, \beta_{ik_i}, \alpha_i \vdash \alpha_i. \tag{1}$$

D'_2 will be the same as D_2 except that extra hypotheses $\beta_{11}, \ldots, \beta_{1k_1}$, $\alpha_1, \ldots$, $\beta_{m1}, \ldots, \beta_{mk_m}, \alpha_m$ are included. The final step will give

$$\beta_{11}, \ldots, \beta_{1k_1}, \alpha_1, \ldots, \beta_{m1}, \ldots, \beta_{mk_m}, \alpha_m \vdash \tau.$$

We then obtain

$$\tau \to b_1, \beta_{11}, \ldots, \beta_{1k_1}, \alpha_1, \ldots, \beta_{m1}, \ldots, \beta_{mk_m}, \alpha_m \vdash b_1$$

and

$$\tau \to b_1, \beta_{21}, \ldots, \beta_{2k_2}, \alpha_2, \ldots, \beta_{m1}, \ldots, \beta_{mk_m}, \alpha_m \vdash \alpha_1 \to \beta_{11} \to \ldots \beta_{1k_1} \to b_1.$$

We now add Δ_1 to the hypotheses and apply D'_{11}, which is D_{11} with extra hypotheses, to give

$$\Delta'_1, \tau \to b_1, \beta_{21}, \ldots, \beta_{2k_2}, \alpha_2, \ldots, \beta_{m1}, \ldots, \beta_{mk_m}, \alpha_m \vdash \alpha_1. \tag{2}$$

We now apply D_2^2, which is D'_2 with (1) for $2 \leq i$ and (2) instead of (1) for $i = 1$. This will give

$$\tau \to b_1, \beta_{21}, \ldots, \beta_{2k_2}, \alpha_2, \ldots, \beta_{m1}, \ldots, \beta_{mk_m}, \alpha_m \vdash \tau,$$
$$\tau \to b_1, \tau \to b_2, \beta_{21}, \ldots, \beta_{2k_2}, \alpha_2, \ldots, \beta_{m1}, \ldots, \beta_{mk_m}, \alpha_m \vdash b_2$$
$$\tau \to b_1, \tau \to b_2, \beta_{21}, \ldots, \beta_{2k_2}, \alpha_2, \ldots, \beta_{m1}, \ldots, \beta_{mk_m}, \alpha_m \vdash \alpha_2 \to \ldots \beta_{2k_2} \to b_2$$

and so on, and eventually we have D_2^{m+1} ending in

$$\tau \to b_1, \tau \to b_2, \ldots, \tau \to b_m \vdash \tau.$$

In D_2^{m+1} each D_{i1} is used once each, so the total number of ν_I steps in the derivation is n fewer than the number in the derivation of $\vdash \tau$. The last uses of ν_I in D_2^{m+1} can be eliminated from this derivation in the same way, eventually resulting in

$$\tau \to b_1, \tau \to b_2, \ldots, \tau \to b_n \vdash \tau,$$

where $m \leq n$. n ν_I steps then give $\vdash \tau$. □

Note that Lemma 6.1 standardizes a proof in classical logic so that it consists of an **SK** (i.e., $H_\to$) proof followed by a sequence of ν_I steps (at most one for each atom b).

Proof-finding algorithm for classical implicational logic (SKP) (with minimal negation)

Aim Given a type τ, to find an X such that $\vdash_{\mathbf{SKP}} X : \tau$.

Method

1. Rewrite τ in terms of $\to$ and f.
2. Find a Y such that

$$y_1 : \tau \to a_1, \ldots, y_n : \tau \to a_n \vdash_{\mathbf{(M)SK}} Y : \tau$$

   by the **SK**-algorithm of [3] or the **MSK**-algorithm above, where $a_1, \ldots, a_n$ are the atomic types of τ. If there is no such Y, there is no X.
3. If there is, then

$$X = \nu y_1 \ldots y_n.Y.$$

Theorem 6.2 *Given a type τ, $\vdash_{\mathbf{SKP}} X : \tau$, for some X if and only if the* **SKP**-*algorithm finds an* **SKP**-*proof of τ.*

Proof If τ has an **SKP**-proof, it can be rewritten as in Lemma 6.1 and the **SK**-algorithm can find a Y such that

$$y_1 : \tau \to a_1, \ldots, y_n : \tau \to a_n \vdash_{\mathbf{SK}} Y : \tau.$$

n applications of ν_I, and the translation from ν to **P**, then give

$$\vdash_{\mathbf{SKP}} X : \tau.$$

If τ has no **SKP**-proof, then no such Y or X can be found and this, again, is determined by the **SK**-algorithm in [3]. □

The following is derived using the algorithm.

Proposition 6.3 *The following are theorems of* **SKP***-logic with minimal negation.*

(i) $((a \to b) \to a) \to a$.
(ii) $a \vee a \to a$.
(iii) $a \vee_1 b \to b \vee_1 a$.
(iv) $(a \to c) \to (b \to c) \to (a \vee_1 b) \to c$.
(v) $a\&(b \vee_1 c) \to (a\&b) \vee_1 c$.
(vi) $a \wedge (b \vee_1 c) \to (a \wedge b) \vee_1 c$.

Proof

(i) $\nu y_1.\lambda x_1.x_1(\lambda x_2.y_1(\lambda x_1.x_2))$.
(ii) $\nu y_1.\lambda x_1.x_1(\lambda x_2.y_1(\lambda x_3.x_2))$.
(iii) $\nu y_1.\lambda x_1 x_2.x_2(x_1(\lambda x_3.y_1(\lambda x_4 x_5.x_3)))$.
(iv) $\nu y_1 y_2.\lambda x_1 x_2 x_3.x_1(y_1(\lambda x_4 x_5 x_6.x_2(x_3(\lambda x_7.x_6(\lambda x_8.y_2(\lambda x_9 x_{10} x_{11}.x_4 x_7))))))$.
(v) $\nu y_1.\lambda x_1 x_2 x_3.x_3(\lambda x_4.x_2(\lambda x_5 x_6.x_1(\lambda x_7 x_8.x_6(\lambda x_9.x_8(\lambda x_{10}.x_7(\lambda x_{11} x_{12}.x_{10}.$
$(y_1(\lambda x_{13} x_{14}.x_{12}(\lambda x_{15}.x_{14}(\lambda x_{16}.x_{13}(\lambda x_{17} x_{18}.x_4 x_{11} x_{15})))))x_9)))))))$.
(vi) $\nu y_1 y_2.\lambda x_1 x_2.x_2(\lambda x_3.x_1(\lambda x_4 x_5.y_2(\lambda x_6 x_7.x_4 x_5(\lambda x_8.x_7(\lambda x_9.x_6$
$(\lambda x_{10} x_{11}.x_3.(\lambda x_{12}.x_8)(y_1(\lambda x_{13} x_{14}.x_{10} x_{11}(\lambda x_{15}.x_{14}(\lambda x_{16}.x_{13}$
$(\lambda x_{17} x_{18}.x_{16}(\lambda x_{19}.x_{15})x_{18})))))))))))$. □

Note that parts (i), (ii), (iii), and (iv) are also **BB′P** and **BB′IP** theorems, but the proofs given above do not translate into **BB′P** or **BB′IP** terms. Here are proofs, obtained by hand, that do:

(i) and (ii) $\lambda x_1.\nu y_1.x_1 y_1$.
(iii) $\lambda x_1 x_2.\nu y_1.x_2(x_1 y_1)$.
(iv) $\lambda x_1 x_2 x_3.\nu y_1.x_2(x_3(\lambda x_4.y_1(x_1 x_4)))$.

It is clear from this that postponing the ν_I steps can make a longer proof; however, the postponement allows a simpler algorithm. For **BB′P** and **BB′IP** logic the postponement of the ν_I step cannot, in general, be done and we have no simple algorithm.

Common tautologies such as (a), (b), (c), (e), (f), and (g) are not theorems of **SKP**-logic as they require Axiom **F**. Given the corresponding remark about **SKF**, it is clear that none of the algorithms introduced here cover common logical systems such as R, T, or E (see Anderson and Belnap [1]).

7 Full Classical Logic

It is clear that the system **SKFP** represents full classical logic. Here is an algorithm.

Proof-finding algorithm for classical propositional (or SKFP-) logic

Aim Given a type τ, to find an X such that $\vdash_{\mathbf{SKFP}} X : \tau$.

Method

1. Rewrite τ in terms of $\to$ and f.
2. Find, using the **SKF** algorithm, a Y such that

$$\vdash_{\mathbf{SKF}} Y : (\tau \to f) \to f;$$

thus, $X = \nu y.\mathbf{F}(Yy)$.

Theorem 7.1 *τ is a theorem of classical propositional logic if and only if a proof of it can be found using the* **SKFP***-algorithm.*

Proof It is well known that if τ is a theorem of classical logic then $(\tau \to f) \to f$ is a theorem of intuitionistic logic. By Theorem 5.2, for any such theorem, a proof can be found by the **SKF**-algorithm. □

Here is an example.

$$\nu y.\mathbf{F}([\lambda x_1.x_1(\lambda x_2 x_3.\mathbf{F}(\lambda x_1.x_3(x_2(\lambda x_4.x_1(\lambda x_2 x_3.x_4)))))]y) : a \vee b \to b \vee a.$$

All the algorithms in this paper were to be implemented in Oostdijk [10] and generally they work well; however, a "shortcut" taken in the implementation has meant that a few proofs are unobtainable, even though the algorithm can be followed easily by hand.

References

[1] Anderson, A. R., and N. D. Belnap, Jr., *Entailment. Volume I. The Logic of Relevance and Necessity*, Princeton University Press, Princeton, 1975. Zbl 0323.02030. MR 0406756. 271

[2] Bunder, M. W., "Lambda terms definable as combinators," *Theoretical Computer Science*, vol. 169 (1996), pp. 3–21. Zbl 0868.03008. MR 1424925. 263, 264, 266

[3] Bunder, M. W., "Proof finding algorithms for implicational logics," *Theoretical Computer Science*, vol. 232 (2000), pp. 165–86. Zbl 0972.03022. MR 1734553. 261, 264, 265, 266, 268, 270, 271

[4] Bunder, M. W., and S. Hirokawa, *Classical Formulas As Types of $\lambda\nu$-terms*, Department of Mathematics, University of Wollongong, Preprint series 14/96, 1996. 269

[5] Curry, H. B., *A Theory of Formal Deducibility*, vol. 6 of *Notre Dame Mathematical Lectures*, University of Notre Dame, Notre Dame, 1950. Zbl 0041.34807. MR 0033779. 268

[6] Dekker, A. H., "Brouwer 7.9.0 - A Proof Finding Program for Intuitionistic, BCI, BCK and Classical Logic," 1996. 266

[7] Gabbay, D. M., and R. J. G. B. de Queiroz, "Extending the Curry-Howard interpretation to linear, relevant and other resource logics," *The Journal of Symbolic Logic*, vol. 57 (1992), pp. 1319–65. Zbl 0765.03005. MR 1195274. 269

[8] Hindley, J. R., *Basic Simple Type Theory*, vol. 42 of *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press, Cambridge, 1997. Zbl 0906.03012. MR 1466699. 264

[9] Hindley, J. R., and J. P. Seldin, *Introduction to Combinators and λ-Calculus*, vol. 1 of *London Mathematical Society Student Texts*, Cambridge University Press, Cambridge, 1986. Zbl 0614.03014. MR 879272. 263

[10] Oostdijk, M., "Lambdacal 2." 272

[11] Popper, K. R., "New foundations for logic," *Mind*, vol. 56 (1947), pp. 193–235; errata 57, 69–70 (1948). MR 0021924. 263, 268

[12] Trigg, P., J. R. Hindley, and M. W. Bunder, "Combinatory abstraction using **B**, **B**$'$ and friends," *Theoretical Computer Science*, vol. 135 (1994), pp. 405–22. Zbl 0838.03012. MR 1311210. 261, 263

School of Mathematics and Applied Statistics
University of Wollongong
Wollongong NSW 2522
AUSTRALIA
mbunder@uow.edu.au

22 Low St
Hurstville NSW 2220
AUSTRALIA