THE LOGIC OF B

by

P.H.B. Gardiner
T.N. Vickers

Technical Monograph PRG-92
ISBN 0-902928-70-8

January 1991

Oxford University Computing Laboratory
Programming Research Group
11 Keble Road
Oxford OX1 3QD
England

Oxford University Computing Laboratory
Programming Research Group
11 Keble Road
Oxford OX1 3QD
England

Electronic mail: Paul.Gardiner@prg.oxford.ac.uk (JANET)

Contents

Introduction

The two papers collected in this monograph describe the logic that forms the theoretical foundation of the proof assistant known as the B-tool. The B-tool was designed as a computerized support for the formal development of imperative programs from Z-like specifications. Its prime functions are to provide a simply-accessed database for recording program developments, and to provide a secure environment for interactively constructing the proofs necessary for those developments.

The style of proof encouraged by the tool is that of building up theories of related facts which are relevant to particular applications. Complete freedom is allowed in the order in which facts are proved. New axioms can be added at any time; one can attempt to prove an axiom of questionable correctness from more obviously-correct axioms at a later date. Although new axioms and rules of inference can be added freely, their use in proofs is strongly controlled, protecting the user from the common errors such as capturing free variables. The tool provides little automation, but concentrates on making interactive proof construction a smooth-running and error-free process.

Proofs are conducted in a goal-oriented way, the tool suggesting rules that match the goal, and the user controlling the proof by accepting or refusing the suggested rules. There is also a simple tactic language which allows some automation of proof construction. With the tactic language the user controls the order in which the theories (i.e., sets of related rules and axioms) are searched. Often a tactic can be chosen which causes the tool always to find the correct rule on its first attempt. In that case the user can waive his right to vet rules, and instead allow the proof to continue automatically. Typically a combination of interaction and automation is used.

The B-tool is quite flexible. Most of the syntax and many of the axioms and inference rules are user-defined, and so the tool can be applied to many problem domains, using various logics. The flexibility is due mainly to the expressiveness of the logical language employed by the tool. The language is actually a meta-language for first order predicate calculus and has a form similar to the meta-languages used informally in logic text books. In it one can state general inference rules such as mathematical induction and existential introduction, thus rules like these can be added as easily as can single predicate calculus formulae. A few indispensable rules have been

built-in and these support a mixture of the natural deduction and equational styles of reasoning.

The use made within the B-tool of its logical language is not restricted to proof. The same language often doubles as a programming language, and as such performs many of the administrative functions, such as reading-in files and pretty-printing. We make no attempt to explain these secondary features of the language; we concentrate on the logical aspects alone. In fact, we doubt the existence of a simple explanation of all aspects of the language, and would recommend a strict separation of the activities of proving and programming within the B-tool.

A Logic for a Theorem Proving Assistant is intended for users of the B-tool and designers of proof assistants that function along the same lines. No deep issues concerning logic are discussed and to those wishing to take the ideas presented here further, we recommend the work of Milsted [8].

Variable Lifting presents the technique by which the B-tool delegates the checking for variable capture to the logic. Because of its slightly unusual nature, we felt obliged to give a full account of its justification.

A Logic for a Theorem Proving Assistant

Paul Gardiner* Trevor Vickers†

Abstract

A detailed description of a simple logic for a theorem proving assistant is presented. The approach taken is closely related to that used for Abrial's B-tool, and so an understanding of the behaviour of the B-tool may be gained. The description of the logic is supported by motivation of the logic's design, and by an example of its use. Some issues for a practical implementation are discussed, including a technique called variable-lifting, which separates the concerns of pattern-matching and variable-capture. Simplicity is the foremost goal. No deep facts concerning logic are assumed or presented.

1 Introduction

The use of a variety of logics in computer science has lead to an increasing interest in tools that allow some freedom in the choice of logic. Our goal has been to design a tool that not only allows this choice but also allows one to capture proofs much as one would on paper and to express and prove new rules of inference. Such flexibility can be achieved using a logical framework in which various (object-) logics can be captured. Here we present a logical framework (or meta-logic), known as *BL*, which provides much of the freedom of naive mathematics, without loss of formality.

Many of the ideas presented here come from work on the B-tool [1, 11]. The developers of the B-tool felt that, for a prototype tool, the choice of

*Programming Research Group, Oxford University, 11 Keble Rd, Oxford OX1 3QG, U.K.

†Department of Computer Science, Australian National University, Canberra, Australia.

a particular logic would prematurely fix the modes of reasoning they could consider. So initially the B-tool was developed by intuition: sometimes adding to its logical language and inference mechanism for the sake of practicality, sometimes restricting its behaviour because unsoundness had been discovered.

In time, the tool became a useful support for the kind of proof methods required, but with no logical basis it could not he trusted. Effort then turned towards the search for a logic to act as a foundation for the tool. The resulting logic is not compromised by its development in this way. As the understanding of the logic increased, and discrepancies between the logic and the tool's behaviour were found, it was the program which was modified and not the logic. The logic therefore is quite general and not restricted by its use as the basis of B.

The following sections motivate the logic's design, supporting the formal definition of *BL*. The definition is followed by an example of *BL*'s use in reasoning about logical systems. We also discuss how a tool might be based on this logic, introducing techniques that simplify the avoidance of variable-capture.

2 Overview of BL

Working with logics necessarily involves the creation of notations for string manipulation (i.e., a meta-language). Books on logic make much use of these. For example, many predicate calculus systems have axiom schemata that cannot be captured by a finite set of formulae. Another example is the proving of schematic theorems. In both cases a meta-language is required.

In designing *BL*, we have taken advantage of the similarity one finds in the meta-languages of a wide range of logics. Rather than work with a particular logic, we have abstracted the parts common to most meta-languages and formalised them. The result is a reasonably-general logical framework, in which one reasons much as one would on paper.

The advantage of this freedom is most apparent in tools based on *BL*. When one wishes to reason about a quantifier, say, not known to the tool, one can usually write introduction and elimination rules directly in *BL* — far better

than having to reprogram the tool. Of course if those rules are invalid, unsound results may follow, and we deliberately place the onus of validity of those rules on the user.

The way we intend exploring a mathematical theory is first to choose some logical system that works well for that theory, and then reason about the strings of that logical system in *BL*. Most importantly, we would want to prove that certain strings are theorems of that system. The logical systems which we reason about in *BL* will be known as object-logics or object-languages.

Since reasoning about strings presents no special problems, the Predicate Calculus seems a natural candidate for a meta-language — with its terms denoting strings of the object-logic, and its formulae expressing properties such as 'is a theorem' and 'does not occur free in'. In fact *BL* is much like the Predicate Calculus: it differs only in being simpler.

Simplifications are possible because we wish to assert and prove only a very restricted set of properties of object-level strings — properties that correspond to object-level inference rules and axiom schemata. These are always of a certain form, just stating that a basic property (such as theoremhood) of one string follows from basic properties of others. For recording such properties, logical connectives like negation and implication are overly expressive. So the first simplification of predicate calculus is to replace the logical connectives by a single construct — the sequent. Rules of inference with side conditions need no special treatment: since side conditions are also properties of strings, they appear simply as extra antecedents. Our notation for sequents copies one often used for displaying rules of inference. The antecedents are written above the consequent with a horizontal line between.

Just as logical connectives are unnecessary for expressing inference rules, so are quantifiers. Thus their removal is the second simplification to the Predicate Calculus. We retain, however, the distinction between terms and formulae. Note that these simplifications are not imposed on the object-languages that *BL* reasons about. The object-languages may have logical connectives and quantifiers.

By simplifying the meta-language we also simplify the task of building a tool. Sequents are ideal for a goal-oriented proof style (without committing us to

that style), and the lack of quantifiers removes the possibility of variable capture (i.e. capture of meta variables), so that simple pattern matching can be employed during proof construction.

3 Overview of BL's use

The meta-languages found in logic texts are quite schematic in nature. Axiom schemata are presented by writing one instance with the parts that can vary indicated by letters of some reserved font (calligraphic say). For example, Hilbert-style formalisations of Predicate Calculus often have the schema

$$\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{A})$$

This schematic style is easily formalised in *BL* by associating a meta-language function symbol with each connective of the object-language. In the case of implication, the function symbol is 2-placed and its meaning is the function that takes two strings and returns the result of concatenating them with an implication symbol between. That is

$$[\![\phi \Rightarrow \psi]\!] \mathrel{\hat{=}} [\![\phi]\!] \Rightarrow [\![\psi]\!]$$

where $[\![\,]\!]$ is the semantic function from meta-language to object-language.

Note that the $\Rightarrow$ occurring on the right is the logical connective of Predicate Calculus, whereas the one occurring on the left is the associated meta-language function symbol.

The formal counterpart of the calligraphic letters are the variables of *BL*. By convention, we use capital italic letters for variables. So the instances of the axiom schema are exactly the denotations of the *BL* term

$$A \Rightarrow (B \Rightarrow A)$$

as the variables range over all object-level formulae.

The fact that each instance is an axiom implies that each is also a theorem. It is theoremhood that we will record and reason about in *BL*. Theoremhood is represented by the one-place predicate symbol $\vdash$. So the *BL* formula

$$\vdash A \Rightarrow (B \Rightarrow A)$$

formalises the axiom schema above.

There are still some flaws in our formalism, which we now address. *BL*'s domain of discourse is the set of all strings of the object-language. Only some of these are well-formed, and those that are may be members of one of several syntactic classes. In the literature, axiom schemata are accompanied by text that says what sort of string the calligraphic letters stand for. In our example schema the calligraphic letters stand for formulae. We must make the same restriction in our formalism. To achieve this, we introduce in *BL* a predicate symbol for each syntactic class of the object-language. So, if the object-language is the Predicate Calculus then we introduce one-placed predicate symbols var, trm and frm. The meta-formula

$$\mathsf{frm}\ \phi$$

is true if and only if the meta-term ϕ denotes a well-formed object-level formula. In a similar way, var means object-level variable and trm means object-level term.

Now we can correctly formalise the example axiom schema with the following *BL* sequent.

$$\frac{\mathsf{frm}\ A\ \ \mathsf{frm}\ B}{\vdash\ A \Rightarrow (B \Rightarrow A)}$$

So far we have a language sufficient to express, for example, most of the Hilbert-style axiom schemata, but there are still a few we cannot. One such is

$$(\forall\, x \bullet \mathcal{A}(x)) \Rightarrow \mathcal{A}(\tau)$$

Here $\mathcal{A}(x)$ stands for a formula in which a variable, x, has been singled out, and $\mathcal{A}(\tau)$ stands for the result of replacing each occurrence of that variable by the term τ. We could make this formal by considering $\mathcal{A}$ to be a function from strings to strings, but this would complicate *BL* unnecessarily. Instead, we formalise substitution in *BL* with a 3-place function symbol, which we will call formal substitution, written $[\phi := \psi]\theta$. Its meaning is defined only when ϕ denotes a variable, ψ denotes a term and θ denotes a term or formula. In that case $[\![[\phi := \psi]\theta]\!]$ is the result of replacing all free occurrences of $[\![\phi]\!]$ by $[\![\psi]\!]$ in $[\![\theta]\!]$. In all other cases its meaning is arbitrary. Using formal substitution, the above axiom schema can be written as follows.

$$\frac{\mathsf{var}\ X\ \ \mathsf{trm}\ T\ \ \mathsf{frm}\ P}{\vdash\ (\forall X \bullet P) \Rightarrow [X := T]P}$$

There is a danger with substitution that variables may become captured. In the literature, capture is often avoided by enforcing side conditions like 'τ is free for x in $\mathcal{A}$'. We could represent such side conditions by predicate symbols of *BL*, but there is a simpler path: substitutions can be performed in a way that avoids capture.

If one accepts the validity of changing the names of bound variables, then one can change all the bound variables of $\mathcal{A}$, making them different from the free variables of τ. After this, the substitution can be performed without risk of capture. We choose this combination of alpha-conversion and substitution (known as safe substitution) for the meaning of $[\_ := \_]\_$.

Not all side conditions can be avoided. One that appears often and that we do have to formalise is 'does not occur free in'. This is represented in *BL* by the predicate symbol $\phi \setminus \psi$. Its meaning is defined only when ϕ denotes a variable and ψ denotes a term or formula. In that case, $\phi \setminus \psi$ is true iff $[\![\phi]\!]$ does not occur free in $[\![\psi]\!]$.

With what we have presented so far, we can provide a meta-language for many logics. Common practice, however, is to alternate between reasoning schematically within the meta-language and reasoning directly in the object-language. *BL* can be used for the former, but how do we achieve the latter? The answer is quite simple. If we extend *BL*, so that it can refer to particular object-level strings, then we can mirror individual object-level proofs. Little needs to be added to *BL*. We already have meta-level function symbols associated with each object-level connective. All we need do is associate a meta-constant with each of the object-level constants and variables. That is, for each object-variable (x say) we have a meta-constant (also written x) such that

$$[\![x]\!] \cong x$$

Similarly, object-constants are denoted by meta-constants.

Thus, for example, theoremhood of the object-formula

$$m + n = n + m$$

is expressed by the *BL* sequent

$$\frac{}{\vdash m + n = n + m}$$

Having introduced all the main concepts, we are ready to define *BL*. We show how *BL* is applied as a meta-language in section 5, where we use it to describe Hilbert-style Predicate Calculus.

4 Definition of BL

In this section we define the syntax and inference mechanism of *BL*, forgetting for now that *BL* will be used to reason about other logics.

We said earlier that *BL* is like a simple form of Predicate Calculus, with the logical operators replaced by sequents. So, strictly speaking, *BL* is a sequent calculus. Sequents are constructed in the following way.

$$
\begin{aligned}
SEQUENT &::= \frac{FORMULA_1 \cdots FORMULA_k}{FORMULA} \\
FORMULA &::= \mathsf{P}_n(TERM_1, \ldots, TERM_n) \\
TERM &::= \mathsf{F}_m(TERM_1, \ldots, TERM_m) \\
&\quad \mid VARIABLE \\
&\quad \mid \mathsf{C}
\end{aligned}
$$

where P_n stands for any n-ary predicate symbol, F_m for any m-ary function symbol and C for any constant symbol.

For variables we will use upper-case italic letters, sometimes subscripted by a number. The predicate, function and constant symbols will vary according to application.

As in the predicate calculus, theorems are derived from axioms using rules of inference — although in *BL* the axioms and theorems are sequents, not formulae. The *BL* rules of inference are as follows. They are not unusual (see Scott [9]).

$$
\begin{aligned}
&ASSUME && \frac{\phi}{\phi} \\
&THIN && \textit{if } \frac{\Gamma}{\phi} \textit{ then } \frac{\Gamma\,\Delta}{\phi} \\
&CUT && \textit{if } \frac{\Gamma}{\phi} \textit{ and } \frac{\Gamma\,\phi}{\psi} \textit{ then } \frac{\Gamma}{\psi}
\end{aligned}
$$

$$\textit{INSTANTIATE} \quad \textit{if } \frac{\Gamma}{\phi} \textit{ then } \frac{\Gamma'}{\phi'}$$

where ϕ and ψ are formulae, Γ and Δ are sets of formulae, and Γ' and ϕ' are respectively Γ and ϕ after application of a substitution of terms for variables. Standard techniques can be used to show the soundness and completeness of *BL*. These results are not presented here.

The following example illustrates the use of these rules.

Example
Let $\rhd$ be a predicate symbol. Then from

$$\frac{X \rhd Y \quad Y \rhd Z}{X \rhd Z} \qquad S1$$

we can derive

$$\frac{A \rhd B \quad B \rhd C \quad C \rhd D}{A \rhd D}$$

Proof
Apply *INSTANTIATE* to $S1$ to get

$$\frac{A \rhd B \quad B \rhd C}{A \rhd C} \qquad S2$$

and

$$\frac{A \rhd C \quad C \rhd D}{A \rhd D} \qquad S3$$

then apply *THIN* to $S2$ and $S3$ to get respectively

$$\frac{A \rhd B \quad B \rhd C \quad C \rhd D}{A \rhd C} \qquad S4$$

and

$$\frac{A \rhd B \quad B \rhd C \quad C \rhd D \quad A \rhd C}{A \rhd D} \qquad S5$$

and finally apply *CUT* to $S4$ and $S5$.

As you will have noticed, the inference rules are cumbersome. So when presenting proofs we will use a different method of inference, which is justified by the inference rules. This method works at the level of formulae rather

than sequents. The method consists in the construction of a set of formulae, starting with the antecedents of the sequent to be proved, and then adding new formulae until the consequent is present. A new formula can be added to the set only if it is the consequent of a sequent (either an axiom or the result of *INSTANTIATE* applied to an axiom) whose antecedents are already present.

Using this method of inference the example can be presented more succinctly, as follows.

Proof

(1)	$A \rhd B$	assumption
(2)	$B \rhd C$	assumption
(3)	$C \rhd D$	assumption
(4)	$A \rhd C$	1, 2 and $S1$
(5)	$A \rhd D$	3, 4 and $S1$

Equality

The equality relation is sufficiently domain-independent to warrant inclusion as part of *BL*. We represent it by the predicate symbol ==, saving = for equality in the object-languages. It is axiomatised by the following standard sequents, for each function symbol F and predicate symbol P.

$$\frac{}{X == X}$$

$$\frac{X_1 == Y_1 \cdots X_n == Y_n}{\mathsf{F}(X_1, \ldots, X_n) == \mathsf{F}(Y_1, \ldots, Y_n)}$$

$$\frac{X_1 == Y_1 \cdots X_n == Y_n \quad \mathsf{P}(X_1, \ldots, X_n)}{\mathsf{P}(Y_1, \ldots, Y_n)}$$

One can think of these sequents as being like the logical axioms of Predicate Calculus: to reason about a particular domain of discourse they are augmented by other domain specific axioms.

The == axioms justify term rewriting, and that is how we use them. During a proof in the style of the last example, if we have derived a formula $\mathcal{A}(\phi)$ in which the term ϕ occurs, and we are able to derive $\phi == \psi$ or $\psi == \phi$, then we may also derive $\mathcal{A}(\psi)$.

This concludes the definition of *BL*.

5 Example: Hilbert-style predicate calculus

In this section we present a formalisation of Hilbert-style Predicate Calculus as an example of *BL*'s use. We follow the description of Predicate Calculus found in Mendelson [7] except we formalise, as *BL* sequents, what is stated there in English. We will refer to that calculus as *PC*, and our formalisation as *PC*-theory.

First we deal with the rules of inference and logical axioms of *PC*. To introduce these we need to mention the following *PC* symbols.

variables	$a, b, c, \ldots$
predicate symbols	$\_ = \_$
quantifiers	$\forall \_ \bullet \_$
logical operators	$\neg \_\,,\ \_ \Rightarrow \_$

To record facts about these *PC* symbols, we will need the following *BL* symbols.

constants	$a, b, c, \ldots$
variables	$A, B, C, \ldots$
function symbols	$\_ = \_\,,\ \forall \_ \bullet \_\,,\ \neg \_\,,\ \_ \Rightarrow \_\,,\ [\_ := \_]\_$
predicate symbols	$\mathsf{var}\ \_\,,\ \mathsf{trm}\ \_\,,\ \mathsf{frm}\ \_\,,\ \_ \setminus \_\,,\ \vdash \_$

When describing a logic one must present its syntax. The following sequents can be thought of as being the *BL* equivalent of a syntax for the part of *PC* introduced so far. They express the concept of well-formedness.

WLF

$$\frac{}{\mathsf{var}\ \pi} \qquad \text{for each } \pi \text{ drawn from } a, b, c, \ldots .$$

$$\frac{\mathsf{var}\ X}{\mathsf{trm}\ X} \qquad \frac{\mathsf{trm}\ S\ \ \mathsf{trm}\ T}{\mathsf{frm}\ S = T} \qquad \frac{\mathsf{var}\ X\ \ \mathsf{frm}\ P}{\mathsf{frm}\ \forall X \bullet P}$$

$$\frac{\mathsf{frm}\ P}{\mathsf{frm}\ \neg P} \qquad \frac{\mathsf{frm}\ P\ \ \mathsf{frm}\ Q}{\mathsf{frm}\ P \Rightarrow Q}$$

In section 3 we said that $\_ \setminus \_$ represents non-freeness. To allow reasoning about non-freeness in *BL* we introduce the following axioms.

NFR

$$\frac{}{\pi \setminus \rho}$$ for each pair π, ρ of distinct elements drawn from $a, b, c, \ldots$

$$\frac{\textsf{var } X \;\; \textsf{trm } S \;\; \textsf{trm } T \qquad X \setminus S \;\; X \setminus T}{X \setminus S = T}$$

$$\frac{\textsf{var } X \;\; \textsf{frm } P}{X \setminus \forall X \bullet P}$$

$$\frac{\textsf{var } X \;\; \textsf{var } Y \;\; \textsf{frm } P \qquad X \setminus P}{X \setminus \forall Y \bullet P}$$

$$\frac{\textsf{var } X \;\; \textsf{frm } P \qquad X \setminus P}{X \setminus \neg P}$$

$$\frac{\textsf{var } X \;\; \textsf{frm } P \;\; \textsf{frm } Q \qquad X \setminus P \;\; X \setminus Q}{X \setminus P \Rightarrow Q}$$

The following sequents axiomatise formal substitution.

SUB

$$\frac{\mathsf{var}\ X\ \ \mathsf{trm}\ E\ \ \mathsf{trm}\ T \qquad X \setminus T}{[X := E]T == T} \qquad \frac{\mathsf{var}\ X\ \ \mathsf{trm}\ E\ \ \mathsf{frm}\ P \qquad X \setminus P}{[X := E]P == P}$$

$$\frac{\mathsf{var}\ X\ \ \mathsf{trm}\ E}{[X := E]X == E}$$

$$\frac{\mathsf{var}\ X\ \ \mathsf{trm}\ E\ \ \mathsf{trm}\ S\ \ \mathsf{trm}\ T}{[X := E](S = T) == [X := E]S = [X := E]T}$$

$$\frac{\mathsf{var}\ X\ \ \mathsf{trm}\ E\ \ \mathsf{var}\ Y\ \ \mathsf{frm}\ P \qquad Y \setminus X \qquad Y \setminus E}{[X := E](\forall\, Y \bullet P) == \forall\, Y \bullet [X := E]P}$$

$$\frac{\mathsf{var}\ X\ \ \mathsf{trm}\ E\ \ \mathsf{frm}\ P}{[X := E]\neg\, P == \neg\, [X := E]P}$$

$$\frac{\mathsf{var}\ X\ \ \mathsf{trm}\ E\ \ \mathsf{frm}\ P\ \ \mathsf{frm}\ Q}{[X := E](P \Rightarrow Q) == [X := E]P \Rightarrow [X := E]Q}$$

Lastly we have the rules of inference and axioms of PC. PC has two rules of inference: Modus Ponens and Generalisation. These can be written in BL as follows.

$$\frac{\begin{array}{c}\mathsf{frm}\ P\ \ \mathsf{frm}\ Q \\ \vdash\ P \\ \vdash\ P \Rightarrow Q\end{array}}{\vdash\ Q}\ MP \qquad\qquad \frac{\begin{array}{c}\mathsf{var}\ X\ \ \mathsf{frm}\ P \\ \vdash\ P\end{array}}{\vdash\ \forall X \bullet P}\ GEN$$

And PC has five axiom schemata.

$$\frac{\mathsf{frm}\ P\ \ \mathsf{frm}\ Q}{\vdash\ P \Rightarrow (Q \Rightarrow P)}\ L1$$

$$\frac{\mathsf{frm}\ P\ \ \mathsf{frm}\ Q\ \ \mathsf{frm}\ R}{\vdash\ (P \Rightarrow Q) \Rightarrow ((P \Rightarrow (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R))}\ L2$$

$$\frac{\mathsf{frm}\ P\ \ \mathsf{frm}\ Q}{\vdash\ (\neg\, P \Rightarrow Q) \Rightarrow ((\neg\, P \Rightarrow \neg\, Q) \Rightarrow P)}\ L3$$

$$\frac{\mathsf{var}\ X\ \ \mathsf{trm}\ T\ \ \mathsf{frm}\ P}{\vdash\ (\forall X \bullet P) \Rightarrow [X := T]P}\ L4$$

$$\frac{\begin{array}{c}\mathsf{var}\ X\ \ \mathsf{frm}\ P\ \ \mathsf{frm}\ Q \\ X \setminus P\end{array}}{\vdash\ (\forall X \bullet P \Rightarrow Q) \Rightarrow (P \Rightarrow \forall X \bullet Q)}\ L5$$

The following example illustrates how one uses PC-theory.

Example

$$\frac{\mathsf{frm}\ P}{\vdash\ P \Rightarrow P}$$

Proof

(1)	$\mathsf{frm}\ P$	assumption
(2)	$\mathsf{frm}\ P \Rightarrow P$	1 and *WLF*
(3)	$\mathsf{frm}\ P \Rightarrow (P \Rightarrow P)$	1, 2 and *WLF*
(4)	$\mathsf{frm}\ (P \Rightarrow P) \Rightarrow P$	1, 2 and *WLF*
(5)	$\mathsf{frm}\ P \Rightarrow ((P \Rightarrow P) \Rightarrow P)$	1, 4 and *WLF*
(6)	$\mathsf{frm}\ (P \Rightarrow ((P \Rightarrow P) \Rightarrow P)) \Rightarrow (P \Rightarrow P)$	2, 5 and *WLF*
(7)	$\vdash\ P \Rightarrow (P \Rightarrow P)$	1 and *L1*
(8)	$\vdash\ P \Rightarrow ((P \Rightarrow P) \Rightarrow P)$	1, 2 and *L1*
(9)	$\vdash\ (P \Rightarrow (P \Rightarrow P)) \Rightarrow ((P \Rightarrow ((P \Rightarrow P) \Rightarrow P)) \Rightarrow (P \Rightarrow P))$	1, 2 and *L2*
(10)	$\vdash\ (P \Rightarrow ((P \Rightarrow P) \Rightarrow P)) \Rightarrow (P \Rightarrow P)$	3, 6, 7, 9 and *MP*
(11)	$\vdash\ P \Rightarrow P$	2, 5, 8, 10 and *MP*

6 A proof assistant based on BL

In this section we discuss a very simple way in which *BL* proofs might be constructed with machine support. We imagine a proof assistant to which the user has added the axioms of *PC*-theory. Proofs are constructed in reverse, using a goal directed approach. For example, the sequent

$$\frac{}{\vdash\ \forall x \bullet x = x}$$

might be proved as follows. A set of goals would be formed, initially containing just the one formula $\vdash\ \forall x \bullet x = x$. The data base of axioms would be searched to find one whose consequent matches the goal. There are two such.

$$\frac{\begin{array}{c}\mathsf{frm}\ P\ \ \mathsf{frm}\ Q \\ \vdash\ P \\ \vdash\ P \Rightarrow Q\end{array}}{\vdash\ Q}\quad MP$$

and

$$\frac{\begin{array}{c}\textsf{var } X \ \ \textsf{frm } P \\ \vdash P\end{array}}{\vdash \forall X \bullet P} \qquad GEN$$

The user would choose one of these (*GEN* say). The matching process would yield tbe instantiation, x for X and $x = x$ for P, thus transforming *GEN* to,

$$\frac{\begin{array}{c}\textsf{var } x \ \ \textsf{frm } x = x \\ \vdash x = x\end{array}}{\vdash \forall x \bullet x = x} \qquad GEN'$$

and the goal would be replaced by the antecedents of *GEN'*. If the user chose *MP* then much the same would happen, except that only Q's instantiation would be obtained hy matching. The user would be prompted for P's instantiation.

The application of axioms would continue, acting on new goals as for the first, until no goals remain. The proof is then complete.

In theory, a tool with the facilities we have just described could be used as it stands; one could type in the axioms of *PC*-theory, add to these the axioms of number tbeory, and prove number theoretic results. In reality, proof construction would be far too slow with such a tool. In the next section we look at ways of speeding up the process.

7 A practical implementation

The inefficiency of our imagined tool has two sources. One source is the *PC – theory* axioms for well-formedness, non-freeness and substitution; these have to be applied repeatedly just to perform a single substitution or derive a single non-freeness condition. The other source is the object-logic; *PC* was designed for the study of proof, not for practical use.

The problem with well-formedness, non-freeness and substitution is easily dealt with. The axioms for these can be built in to the tool. Then the many steps necessary to perform a substitution, say, can be presented to the user as a single step.

There is another advantage in building in this part of *PC*-theory. These

axioms occur in groups containing one axiom per symbol of the object-language. So when a new symbol is introduced these axioms must be supplemented. With the axioms built in, the tool can be responsible for supplementing them — the user having merely to specify whether a new symbol is of the object-language, and if so, what type of symbol (e.g., constant symbol, function symbol etc.). Building in parts of PC-theory restricts the user's choice of object-logic, so the decision to do so might not be appropriate for all applications.

To avoid the awkwardness of PC we can use a different inference system. The B-tool uses the language of PC, with a combination of Natural Deduction and Term Rewriting. Next, we outline how this is achieved in BL.

7.1 Natural deduction

To support natural deduction we need to have BL reason about theorems under hypotheses. So we supplement the predicate symbol $\vdash \_$ with an infinite family of new predicate symbols, one for each length of hypothesis list. These are written

$$\_,\ldots,\_ \vdash \_$$

We give meaning to these new symbols by saying that

$$\phi_1,\ldots,\phi_n \vdash \psi$$

is true iff the following is true.

$$\vdash \phi_1 \Rightarrow (\phi_2 \Rightarrow \cdots(\phi_n \Rightarrow \psi))$$

This relationship can be used to justify replacing the logical axioms of PC by the following sequents.

$$\frac{\mathsf{frm}\ H_1 \cdots \mathsf{frm}\ H_n}{H_1, \ldots, H_n \vdash H_i} \quad HYP \qquad 1 \leq i \leq n$$

$$\frac{\begin{array}{c}\mathsf{frm}\ G_1 \cdots \mathsf{frm}\ G_n \\ \mathsf{frm}\ H_1 \cdots \mathsf{frm}\ H_m\ \ \mathsf{frm}\ P \\ H_1, \ldots, H_m \vdash P \\ G_1, \ldots, G_n \vdash H_1 \\ \vdots \\ G_1, \ldots, G_n \vdash H_m\end{array}}{G_1, \ldots, G_n \vdash P} \quad TRANS$$

$$\frac{\begin{array}{c}\mathsf{var}\ X\ \ \mathsf{frm}\ P \\ \mathsf{frm}\ H_1 \cdots \mathsf{frm}\ H_n \\ X \setminus H_1 \cdots X \setminus H_n \\ H_1, \ldots, H_n \vdash P\end{array}}{H_1, \ldots, H_n \vdash \forall X \bullet P} \quad GEN$$

$$\frac{\mathsf{var}\ X\ \ \mathsf{trm}\ T\ \ \mathsf{frm}\ P}{\forall X \bullet P \vdash [X := T]P} \quad SPEC$$

$$\frac{\begin{array}{c}\mathsf{frm}\ P\ \ \mathsf{frm}\ Q \\ \mathsf{frm}\ H_1 \cdots \mathsf{frm}\ H_n \\ H_1, \ldots, H_n, P \vdash Q\end{array}}{H_1, \ldots, H_n \vdash P \Rightarrow Q} \quad DED \qquad \frac{\mathsf{frm}\ P\ \ \mathsf{frm}\ Q}{P, P \Rightarrow Q \vdash Q} \quad MP$$

$$\frac{\mathsf{frm}\ P\ \ \mathsf{frm}\ Q}{\neg P \Rightarrow Q, \neg P \Rightarrow \neg Q \vdash P} \quad CONTRA$$

These axioms give shorter and more easily-constructed proofs, but they do have one drawback: there are infinitely many of them. So some of them will have to be built in to the proof assistant.

SPEC, *MP* and *CONTRA* need not be built in. They are single sequents and therefore can be typed in when needed. *HYP*, *GEN* and *DED* can be built in directly as they stand. This leaves *TRANS*.

TRANS is built in indirectly. It modifies the way in which certain sequents are used in proofs. The sequents that are affected are those of the form

$$\frac{\theta_1 \cdots \theta_n}{\phi_1, \ldots, \phi_m \vdash \psi} \qquad (S1)$$

where $\theta_1, \ldots, \theta_n$ are well-formedness or non-freeness conditions. If we consider how proofs are generated, we notice that a sequent of this form can be used only when its consequent exactly matches a goal — an uncommon occurrence. However, if we take such a sequent together with *TRANS* we can derive

$$\frac{\begin{array}{c}\theta_1 \cdots \theta_n \\ \mathsf{frm}\ \phi_1 \cdots \mathsf{frm}\ \phi_m \quad \mathsf{frm}\ \psi \\ \mathsf{frm}\ \tau_1 \cdots \mathsf{frm}\ \tau_k \\ \tau_1, \ldots, \tau_k \vdash \phi_1 \\ \vdots \\ \tau_1, \ldots, \tau_k \vdash \phi_m\end{array}}{\tau_1, \ldots, \tau_k \vdash \psi} \qquad (S2)$$

where $\tau_1, \ldots, \tau_k$ are arbitrary meta-terms.

With these sequents, a greater range of goals can be matched. A match is possible provided the goal has $\vdash$ as its predicate symbol with the rightmost argument matching ψ. The transformation from $(S1)$ to $(S2)$ can be thought of as a derived rule of inference.

7.2 Term rewriting

BL already supports meta-level term rewriting, but this is of use only in exchanging meta-terms that denote the same object-level string. Wbat we wish to do is reason in *BL* about object-level term rewriting.

We can give a simple axiomatisation in *BL* as follows, for each object-level function symbol $\mathcal{F}$, predicate symbol $\mathcal{P}$ and logical operator $\mathcal{L}$.

$$\frac{\mathsf{trm}\ T}{\vdash\ T = T}$$

$$\frac{\begin{array}{c}\mathsf{trm}\ S_1 \cdots \mathsf{trm}\ S_n \\ \mathsf{trm}\ T_1 \cdots \mathsf{trm}\ T_n\end{array}}{S_1 = T_1, \ldots, S_n = T_n \vdash\ \mathcal{F}(S_1, \ldots, S_n) = \mathcal{F}(T_1, \ldots, T_n)}$$

$$\frac{\begin{array}{c}\mathsf{trm}\ S_1 \cdots \mathsf{trm}\ S_n \\ \mathsf{trm}\ T_1 \cdots \mathsf{trm}\ T_n\end{array}}{S_1 = T_1, \ldots, S_n = T_n \vdash\ \mathcal{P}(S_1, \ldots, S_n) \Leftrightarrow \mathcal{P}(T_1, \ldots, T_n)}$$

$$\frac{\begin{array}{c}\mathsf{frm}\ S_1 \cdots \mathsf{frm}\ S_n \\ \mathsf{frm}\ T_1 \cdots \mathsf{frm}\ T_n\end{array}}{S_1 \Leftrightarrow T_1, \ldots, S_n \Leftrightarrow T_n \vdash\ \mathcal{L}(S_1, \ldots, S_n) \Leftrightarrow \mathcal{L}(T_1, \ldots, T_n)}$$

These axioms are not used directly. Instead, they appear implicitly as alterations to the procedures for pattern matching and goal generation — much in the way that *TRANS* does. The details are a little messy, and we omit them here.

7.3 Variable lifting

The above proof system works well provided the sequents called upon in a proof are schematic (i.e., contain meta-variables). There are more problems to solve, however, if we wish to use non-schematic sequents effectively.

Consider the following sequent.

$$\frac{}{\vdash\ a \leq b \Leftrightarrow (\exists x \bullet a + x = b)} \qquad (1)$$

From this, we would expect to derive directly:

$$\frac{}{\vdash\ p \leq q + r \Leftrightarrow (\exists\, y \bullet p + y = q + r)} \qquad (2)$$

But in fact, the derivation is anything but direct.

To see why this is so, we must take a closer look at (1), remembering that it is a meta-language expression. The symbols a, b and x, which look as though they are PC variables, are actually BL constants and cannot be altered by application of tbe BL inference rule *INSTANTIATE*. Instead, we rely on the inference rules of PC, formalised in BL. In particular, we need generalisation and specialisation to alter a and b, and alpha-conversion to alter x.

It seems, at first sight, to be more convenient to avoid sequents like (1), replacing them by more easily-applied ones like

$$\frac{\mathsf{var}\ X\ \ \mathsf{trm}\ A\ \ \mathsf{trm}\ B}{\vdash\ A \leq B \Leftrightarrow (\exists\, X \bullet A + X = B)} \qquad (3)$$

but these are more difficult to interpret. To decide on the truth of (3) one would have to consider the results of replacing X, A and B by all possible object-language strings, whereas one can safely imagine that (1) is written directly in the object-language. So sequents like (3) involve an extra level of complexity.

Complexity can lead to error. For example, one might think (1) and (3) interchangeable. But, whereas (1) is a legitimate definition of $\leq$, (3) introduces a contradiction. To exhibit the contradiction, apply *INSTANTIATE* to (3) to obtain both

$$\frac{}{\vdash\ a \leq b \Leftrightarrow (\exists\, b \bullet a + b = b)}$$

and

$$\frac{}{\vdash\ 1 \leq 2 \Leftrightarrow (\exists\, x \bullet 1 + x = 2)}$$

Simplify respectively to

$$\frac{}{\vdash\ a \leq b \Leftrightarrow a = 0}$$

and

$$\frac{}{\vdash 1 \leq 2}$$

and, from these last two seqnents, derive

$$\frac{}{\vdash 1 = 0}$$

The cause of the inconsistency can be found at the very first step, where variable capture occurred. Capture can be avoided by adding further antecedents to (3), thus obtaining

$$\frac{\mathsf{var}\ X \quad \mathsf{trm}\ A \quad \mathsf{trm}\ B \quad X \setminus A \quad X \setminus B}{\vdash A \leq B \Leftrightarrow (\exists X \bullet A + X = B)} \qquad (4)$$

But how can we tell whether inconsistency has crept in through some other door?

Rather than leave such questions to the discretion of each user of *BL*, we have systematised the generation of schematic sequents from non-schematic ones. We include the transformation as an inference rule called *VAR-lift*. In fact, (4) is the result of applying *VAR-lift* to (1), and thus (4) is admissible.

Looking at (4), one can see a pattern to the antecedents: var X is there because x occurs bound in (1), trm A and trm B because a and b occur free, $X \setminus A$ and $X \setminus B$ because a and b occur in the scope of x. The pattern generalises to any sequent without antecedents, provided it is made up from only $\vdash$ and symbols of the object-language.

VAR-lift also applies to sequents which have antecedents, and those in which meta-variables, substitution and non-freeness occur. But these additions complicate the transformation and we will not cover them here. For the general statement of *VAR-lift* and a proof of its validity, we refer you to [12].

A proof assistant would apply *VAR-lift* to every sequent entered by its user. Both the lifted and the unlifted version would be stored. That way, the user may enter easily-interpreted sequents like (1), and still have schematic versions of them available during proofs of other sequents. It is probably best if all this is hidden from the user. All that should be discernible is an increased applicability of non-schematic sequents.

8 Conclusions

Although *BL* is very simple and certainly doesn't break any new ground in the study of logic, there are a few advantages we can claim for it over other logics for proof assistants. In fact, it is the simplicity that provides most of the advantages.

Firstly, since the use of *BL* does not involve any complicated coding of the object logic, there is no need to hide *BL* formulae from the users of a tool; formulae can be viewed directly without pretty-printing. This may seem a trivial point but it is very important, when using interactive systems, that man and machine share a common formalism. A disparity between formalisms can lead to confusing machine behaviour and diminish a user's ability to direct a proof.

Our simple approach also works well when applied to object logics in which variable names are significant to meaning (e.g. Hoare Logic [6], Weakest Precondition [4], *Z* [10]). Object logics such as these don't have to be treated specially in *BL*, because object logic variables are treated explicitly. Some other systems (e.g. the LF [2]) avoid free variables by considering terms as functions and quantifiers as higher-order functions. That approach deals cleanly with many object logics, but it runs into trouble with some programming calculi where variable names are significant.

Another advantage of *BL* is that it is easy for non-logicians to understand. Anyone who has read an introductory text on formal logic should find *BL* familiar. *BL* is, after all, just a formal version of the meta languages typically used by such books. The popularity of the B-tool is evidence of the ease with which *BL* is picked up.

There are a few simple extensions to *BL* that we have not covered here. One extension is the use of types. Most object logics have a variety of syntactic classes. In a typed version of *BL*, each syntactic class could be assigned a type, so that object-level, syntactic correctness could be assured by meta-level type checking. The use of types in this way is far more natural and practical than our predicates var, trm, and frm. We avoided types, however, to stay consistent with the B-tool. Milstead [8] takes a similar approach to ours, but uses types.

The use made within the B-tool of its logical language is not restricted to

proof. The same language often doubles as a programming language, and as such performs many of the administrative functions, such as reading-in files and pretty-printing. We have made no attempt to explain these secondary features of the language; we concentrate on the logical aspects alone. In fact, we doubt the existence of a simple explanation of all aspects of the language, and would recommend a strict separation of the activities of proving and programming within the B-tool.

Acknowledgements

Many of our initial ideas were derived from Bourbaki [3]; the presentation of First Order Logic, contained there, is worded very precisely. The further step to a completely formal meta-language was small.

The B-tool was developed by Jean-Raymond Abrial, with assistance from Carroll Morgan, Paul Gardiner, Mike Spivey and Trevor Vickers. Each has also made contributions to the logic of the B-tool, from which *BL* is derived.

Variable Lifting:

deriving schematic object-level inference rules

Trevor Vickers and Paul Gardiner

Abstract

In the context of a formal meta-logic, a process (called *variable lifting*) is described which produces a completely schematic object-level inference rule from a non-schematic (or incompletely schematic) object-level inference rule, axiom or theorem. A full description of variable lifting and a rigorous proof of its derivability is presented.

Variable lifting is not only important for the description and implementation of theorem proving assistants, but also justifies an informal practice found in many logic texts.

1 Introduction

The logic *BL* [5] is a formal meta-language for theorem proving assistants, and was originally designed to describe the behaviour of the B-tool [1].

A proof assistant based on *BL* encourages the user to enter non-schematic object-level inference rules, axioms and theorems. The correctness of these is easily verified by user interpretation, but its applicability is often restricted. A process of transformation, called *variable lifting*, produces a schematic object-level inference rule from a given non-schematic (or partially schematic) object-level inference rule *etc.* The process consists of the replacement of variables by meta-variables while adding antecedents which represent non-freeness properties and prevent variable capture. The resultant inference rule is of general applicability.

We present here a full description of variable lifting, and prove that the lifted sequent can be derived from the given sequent.

That proof is quite complex. We reduce the complexity by introducing some simplifying notation and several theorems and lemmas before presenting the proof.

2 Overview of BL

This section presents the elements of *BL* essential to the understanding of the variable lifting discussion. The remaining details can be found in [5].

BL consists of a simple meta-logic which may be extended by embedding a chosen object-logic. All facts about the object-logic are formalized as *sequents*, written $\frac{\Gamma}{\phi}$. The meta-logic is essentially the rules *assume*, *cut*, *thin*, and *instantiate* of the sequent calculus. There is one given meta-predicate symbol: $P == Q$ means P and Q are the same (object-level) expression.

An object-logic extends *BL* by the association of *BL* meta-function symbols with object-logic predicate and function symbols, *BL* meta-constants with object-logic variables, and so on. For example, the Predicate Calculus includes the following symbols.

variables	$a, b, c, \ldots$
quantifiers	$\forall\_ \bullet \_$
logical operators	$\neg \_\,,\ \_ \Rightarrow \_$

Facts about these symbols are recorded by the following *BL* symbols.

constants	$a, b, c, \ldots$
variables	$A, B, C, \ldots$
function symbols	$\forall\_ \bullet \_\,,\ \neg \_\,,\ \_ \Rightarrow \_\,,\ [\_ := \_]\_$
predicate symbols	$\mathsf{var}\ \_\,,\ \mathsf{trm}\ \_\,,\ \mathsf{frm}\ \_\,,\ \_ \setminus \_\,,\ \_\,,\ldots,\_ \vdash \_$

Note that $\mathsf{var}\ X$, $\mathsf{trm}\ X$, and $\mathsf{frm}\ X$ mean that X denotes an object-variable, object-term, and object-formula respectively. $X \setminus P$ means there are no free occurrences of X in P. $[X := T]P$ represents the object-level substitution of T for all free occurrences of X in P. Facts about the object-logic are axiomatized as a set of *BL* sequents.

In the remainder of the paper we use the Predicate Calculus extension of

BL to demonstrate variable lifting. It will be clear what conditions must be met by other object-logics for variable lifting to be applied to them.

To avoid ambiguity we often use 'meta-variable' to refer to variables of the logic, reserving 'object-variable', and sometimes 'variable' for meta-constants representing object-level (e.g., Predicate Calculus) variables. On other occasions the context will indicate which is intended. We adopt similar terms for other constructs of the object- and meta-levels.

3 Motivation

Suppose we have the following sequent:

$$\frac{}{\vdash n + 0 = n}$$

If we now wish to show the theoremhood of $m+0 = m$, we could perform the following steps. Each step here represents several steps in the meta-logic.

$\vdash n + 0 = n$	
$\vdash \forall n \bullet n + 0 = n$	$\forall$-introduction
$\vdash [n := m](n + 0 = n)$	specialization
$\vdash m + 0 = m$	substitution

Similar steps would be performed in cases where our interest lay in object-terms other than m. Consider the ease with which the result could be established in the presence of the variable-lifted version of the initial sequent:

$$\frac{\mathsf{trm}\ N}{\vdash N + 0 = N}$$

Consider also tbe ease with which a mechanization of the logic could apply the lifted rule by simple pattern matching. A single application of instantiation (of N as m) yields the result. For more complex sequents the derivation of a new sequent requires many steps, while the lifted form will still yield

the result in one step, subject to the satisfaction of its antecedents. For example, suppose we have established,

$$\frac{}{\vdash\ s \subseteq t \equiv \forall x \bullet x \in s \Rightarrow x \in t}$$

To establish the following, similar result is quite laborious.

$$\frac{}{\vdash\ \{a, b\} \subseteq k \equiv \forall y \bullet y \in \{a, b\} \Rightarrow y \in k}$$

$\vdash\ s \subseteq t \equiv \forall x \bullet x \in s \Rightarrow x \in t$	
$\vdash\ \forall s \bullet s \subseteq t \equiv \forall x \bullet x \in s \Rightarrow x \in t$	$\forall$-introduction
$\vdash\ [s := \{a, b\}](s \subseteq t \equiv \forall x \bullet x \in s \Rightarrow x \in t)$	specialization
$\vdash\ \{a, b\} \subseteq t \equiv \forall x \bullet x \in \{a, b\} \Rightarrow x \in t$	substitution
$\vdash\ \forall t \bullet \{a, b\} \subseteq t \equiv \forall x \bullet x \in \{a, b\} \Rightarrow x \in t$	$\forall$-introduction
$\vdash\ [t := k](\{a, b\} \subseteq t \equiv \forall x \bullet x \in \{a, b\} \Rightarrow x \in t)$	specialization
$\vdash\ \{a, b\} \subseteq k \equiv \forall x \bullet x \in \{a, b\} \Rightarrow x \in k$	substitution
$\vdash\ \{a, b\} \subseteq k \equiv \forall y \bullet y \in \{a, b\} \Rightarrow y \in k$	alpha-conversion

The lifted form of the given sequent is:

$$\frac{\mathsf{var}\ X \quad \mathsf{trm}\ S \quad \mathsf{trm}\ T \qquad X \setminus S \quad X \setminus T}{\vdash\ S \subseteq T \equiv \forall X \bullet X \in S \Rightarrow X \in T}$$

Instantiations of S as $\{a, b\}$, T as k and X as y yield the result in one step (ignoring the satisfaction of antecedents).

The added antecedents are very important. Without the non-freeness conditions, the rule is unsound, as they prevent variable capture. Particular instantiations would lead to the following invalid sequent.

$$\frac{}{\vdash\ s \subseteq \{x\} \equiv \forall x \bullet x \in s \Rightarrow x \in \{x\}}$$

While the right hand side of the equivalence is always true, that isn't the case for the left, for arbitrary values of s.

Of course, we could dispense with writing object-expressions altogether, and write the already-lifted sequent down to begin with. The complexity of the last example is convincing evidence of the difficulties to be encountered if we hope to correctly write down such general sequents. We can be much more confident of the correctness of the simpler, non-schematic sequent.

We informally characterize the result of variable lifting by the following. Transliterate object-variables to their corresponding meta-variables. For the new meta-variables, T, which appear as bound variables, add the side-condition var T; for the rest add trm T. For a new bound meta-variable, T, in whose scope appears a new meta-variable, S (different from T), add the side-condition $T \setminus S$. For an existing meta-variable found in the scope of two distinct object-variables translated to X and Y, add the side-condition $X \setminus Y$.

4 Issues in Lifting

In this section we present an example in which the lifted form of a sequent is derived. Our purpose is to give a glimpse of the complexity variable lifting entails, and to provide intuition and understanding for the steps in the general proof of variable lifting.

Our example is the strong induction rule,

$$\frac{k \setminus P \quad \mathsf{frm}\, P}{(\forall k \bullet k < n \Rightarrow [n := k]P) \Rightarrow P \quad \vdash \quad P}$$

whose lifted form we show to be,

$$\frac{K \setminus P \quad \mathsf{frm}\, P \quad \mathsf{var}\, K \quad \mathsf{var}\, N \quad K \setminus N}{(\forall K \bullet K < N \Rightarrow [N := K]P) \Rightarrow P \quad \vdash \quad P}$$

In order to 'guard' the existing meta-variable P from the substitutions to which it will be subject, we first instantiate it to $[k' := k][n' := n][k := k'][n := n']P$ (abbreviated $\dot{P}$). The 'intermediate' variables k' and n' are

distinct from k, n and each other, and do not occur free in P. The 'temporary' variables k'' and n'' are distinct from k, n, k', n' and each other, and do not occur free in P, K or N. (The construction of intermediate and temporary variables within the logic is described in Section 5.4). Instantiation yields,

$$\frac{k \setminus \dot{P} \quad \mathit{frm}\ \dot{P}}{(\forall k \bullet k < n \Rightarrow [n := k]\dot{P}) \Rightarrow \dot{P} \quad \vdash \quad \dot{P}}$$

By a combination of standard rules we introduce the substitution $[k := k'][n := n']$ onto the operands of $\vdash$. At the same time, by the axioms of frm and $\setminus$, we introduce that substitution onto the operand of *frm*, and the substitution $[n := n']$ onto the second operand of $\setminus$. After distributing these substitutions where possible, the sequent is,

$$\frac{k \setminus [n := n']\dot{P} \quad \mathit{frm}\,[k := k'][n := n']\dot{P}}{\begin{array}{r}(\forall k \bullet k < n' \Rightarrow [n := k]\dot{P}) \Rightarrow [k := k'][n := n']\dot{P} \\ \vdash \quad [k := k'][n := n']\dot{P}\end{array}}$$

In general, the above steps will change a variable x to x' unless x occurs where only a variable may: for example, in $[x := T]Q$, or in $x \setminus Q$, or as a bound variable in $\forall x \bullet Q$. Alpha conversion deals with the last case, and axioms of $[:=]$ and $\setminus$ deal with the others. Applied to our example, and after some simplification, they yield,

$$\frac{k' \setminus [k := k'][n := n']\dot{P} \quad \mathit{frm}\,[k := k'][n := n']\dot{P}}{\begin{array}{r}(\forall k' \bullet k' < n' \Rightarrow [n' := k']([k := k'][n := n']\dot{P})) \Rightarrow [k := k'][n := n']\dot{P} \\ \vdash \quad [k := k'][n := n']\dot{P}\end{array}}$$

We note,

$$\begin{array}{l}[k := k'][n := n']\dot{P} \\ \quad == [k := k'][n := n'][k' := k][n' := n][k := k''][n := n'']P \\ \quad == [k := k''][n := n'']P\end{array}$$

giving,

$$\frac{k' \setminus [k := k''][n := n'']P \quad \mathsf{frm}\,[k := k''][n := n'']P}{(\forall k' \bullet [n' := k'][k := k''][n := n'']P) \Rightarrow [k := k''][n := n'']P \vdash [k := k''][n := n'']P}$$

To remove these substitutions on P, we introduce the substitution $[k'' := k][n'' := n]$ onto the operands of $\vdash$ and frm and the second operand of $\setminus$, as before. This substitution distributes completely, affecting only $[k := k''][n := n'']P$, which simplifies to P. Thus we have,

$$\frac{k' \setminus P \quad \mathsf{frm}\,P}{(\forall k' \bullet k' < n' \Rightarrow [n' := k']P) \Rightarrow P \vdash P}$$

Significantly, this is our initial sequent with all occurrences of k, n replaced by k', n', and concludes the first phase of the derivation. The second phase is almost identical, except we replace k' by K and n' by N. The 'freshness' of k', n', k'', n'' has enabled the dismissal of many conditions which would have remained had we attempted simply to replace k, n by K, N.

We commence the second phase by 'guarding' P by instantiation to $[K := k'][N := n'][k' := k''][n' := n'']P$ (abbreviated $\ddot{P}$).

$$\frac{k' \setminus \ddot{P} \quad \mathsf{frm}\,\ddot{P}}{(\forall k' \bullet k' < n' \Rightarrow [n' := k']\ddot{P}) \Rightarrow \ddot{P} \vdash \ddot{P}}$$

The substitution $[k' := K][n' := N]$ is introduced onto the operands of $\vdash$ and frm, and $[n' := N]$ onto the second operand of $\setminus$. At this point certain conditions are required to maintain object-level well-formedness.

$$\frac{\mathsf{trm}\,K \quad \mathsf{trm}\,N \qquad k' \setminus [n' := N]\ddot{P} \quad \mathsf{frm}\,[k' := K][n' := N]\ddot{P}}{(\forall k' \bullet k' < N \Rightarrow [n' := k']\ddot{P}) \Rightarrow [k' := K][n' := N]\ddot{P} \vdash [k' := K][n' := N]\ddot{P}}$$

Again we use alpha conversion and the other rules above to change k' to K and n' to N. This requires the extra conditions $\mathsf{var}\,K$, $\mathsf{var}\,N$, and $K \setminus N$

below.

$$\frac{\mathsf{var}\ K \quad \mathsf{var}\ N \quad K \setminus N \qquad K \setminus [k' := K][n' := N]\ddot{P} \quad \mathsf{frm}\,[k' := K][n' := N]\ddot{P}}{(\forall K \bullet K < N \Rightarrow [N := K]([k' := K][n' := N]\ddot{P})) \Rightarrow [k' := K][n' := N]\ddot{P} \vdash [k' := K][n' := N]\ddot{P}}$$

We note, given $K \setminus N$,

$$\begin{aligned}&[k' := K][n' := N]\ddot{P}\\ &\quad == [k' := K][n' := N][K := k'][N := n'][k' := k''][n' := n'']P\\ &\quad == [k' := k''][n' := n'']P\end{aligned}$$

This simplification gives,

$$\frac{\mathsf{var}\ K \quad \mathsf{var}\ N \quad K \setminus N \qquad K \setminus [k' := k''][n' := n'']P \quad \mathsf{frm}\,[k' := k''][n' := n'']P}{(\forall K \bullet K < N \Rightarrow [N := K]([k' := k''][n' := n'']P)) \Rightarrow [k' := k''][n' := n'']P \vdash [k' := k''][n' := n'']P}$$

Again these remaining substitutions are removed during simplification of the introduced substitution $[k'' := k'][n'' := n']$, as before. This leaves the lifted form of the initial sequent:

$$\frac{K \setminus P \quad \mathsf{frm}\ P \quad \mathsf{var}\ K \quad \mathsf{var}\ N \quad K \setminus N}{(\forall K \bullet K < N \Rightarrow [N := K]P) \Rightarrow P \quad \vdash \quad P}$$

5 Notation

5.1 Well-formedness

Many of the sequents reasoned with in *BL* have antecedent formulae present merely to ensure well-formedness at the object-level. That is the purpose of $\mathsf{frm}\ P$ below.

$$\frac{\mathsf{frm}\ P}{\vdash P \Rightarrow P}$$

As these well-formedness antecedents can always be ascertained from the rest of the sequent, we adopt the convention of not displaying them. Thus the above sequent will be written,

$$\frac{}{\vdash P \Rightarrow P}$$

We also omit these antecedents from our proofs. It can be shown, by analysis of the inference rules and axioms of the meta-logic, that the omission does not affect the validity of the proof.

5.2 Consecutive Substitution

In the proof of the derivation of the lifted sequent, we consider the consecutive substitution of all object-variables in the sequent. Such consecutive substitutions can be cumbersome:

$$[\rho_1 := \eta_1] \ldots [\rho_n := \eta_n]\phi$$

For convenience we choose to write this as

$$[\rho := \eta]_{1..n}\phi$$

Occasionally we shall restrict attention to a subset of the substitutions, and use the notation,

$$[\sigma \mid \rho := \eta]_{1..n}\phi$$

which includes the substitution $[\rho_i := \eta_i]$ only if i is in the set σ and $1 \leq i \leq n$. Similarly, we omit particular substitutions by,

$$[\bar{\sigma} \mid \rho := \eta]_{1..n}\phi$$

where $\bar{\sigma}$ is the complement (with respect to the set of indices $1..n$) of σ. The above includes a substitution $[\rho_i := \eta_i]$ only if i is *not* in σ and $1 \leq i \leq n$.

In many cases we shall re-order the list of substitutions to bring one in particular to the front or rear. We can do this in a proof whenever SC_P (defined below) is among the assumptions, since the following sequent is

derivable.

$$\frac{SC_P(\rho,\eta)}{[\![\rho := \eta]\!]_t\theta == [\![\rho := \eta]\!]_u\theta}$$

where u is a permutation of t

and where the antecedents $SC_P(\rho,\eta)$ are as follows.

$\rho \setminus \eta$
distinct ρ

Note: For the sake of brevity we write $\rho \setminus \eta$ for $\rho_i \setminus \eta_j$ (all i,j), and ***distinct*** ρ for $\rho_i \setminus \rho_j$ $(i \neq j)$. Elsewhere we write var ρ for var ρ_i (all i) and so on.

The proof of this result is not presented, but relies on the following derivable sequent.

$$\frac{X \setminus Y \quad X \setminus T \quad Y \setminus S}{[X := S][Y := T]P == [Y := T][X := S]P}$$

When the permutation result is applicable (i.e., $SC_P(\rho,\eta)$ is derivable), and the order of substitution unimportant, we omit the subscript, writing simply $[\![\rho := \eta]\!]P$.

The following rules are derivable and are presented without proof.

$$\frac{SC_P(\rho,\eta)}{[\![\delta \mid \rho := \eta]\!][\![\bar{\delta} \mid \rho := \eta]\!]\theta == [\![\rho := \eta]\!]\theta} \quad A.1 \qquad \frac{SC_P(\rho,\eta)}{[\![\rho := \eta]\!]\rho_i == \eta_i} \quad A.2$$

$$\frac{SC_P(\rho,\eta) \quad \rho \setminus \phi}{[\![\rho := \eta]\!]\phi == \phi} \quad A.3 \qquad \frac{SC_P(\rho,\eta) \quad SC_P(\eta,\rho) \quad \rho \setminus \phi}{[\![\rho := \eta]\!][\![\eta := \rho]\!]\phi == \phi} \quad A.4$$

$$\frac{SC_P(\rho,\eta)}{\rho \setminus [\![\rho := \eta]\!]\phi} \quad A.5 \qquad \frac{SC_P(\rho,\eta) \quad \omega \setminus \eta \quad \omega \setminus \phi}{\omega \setminus [\![\rho := \eta]\!]\phi} \quad A.6$$

5.3 A Unifying Concept

Performing variable lifting relies on the ability to change the variables of a formula. For simple expressions, such as $t = t$ the change can be effected through a substitution, perhaps introduced in the larger context of the expression. For quantified expressions, such as $\forall x \bullet x = y$, the x is changed through alpha conversion and the y by an introduced substitution. Another example is $[x := t](x = y)$, in which the t and y can be changed by introduced substitutions, but the x must be changed by applying the axioms of substitution.

In searching for a common representation for all these expressions, we note three important categories of function symbol operands: those (like x above) which occur in a position which can only be occupied by a variable, those (like $x=y$) which refer to that x, and those (like t) which are not influenced by the choice of x.

We use a scheme $\Theta(\beta, \varsigma, \nu)$, in which the bound variable β (e.g., x above), the operand in the scope ς (e.g., $x = y$ above), and those operands outside the scope ν (e.g., t above) are a simple re-arrangement of the usual function symbol operands. An empty category is represented by ϵ.

Every term has a $\Theta(\beta, \varsigma, \nu)$ expression. We call this the *bsn* representation (for *b*ound variable, *s*cope, *n*on-scope). For example, the above expressions become $\Theta_1(\epsilon, \epsilon, (t, t))$, $\Theta_2(x, x = y, \epsilon)$, and $\Theta_3(x, x = y, t)$.

Using this scheme for $\forall$, the standard alpha conversion rule is written as a rule scheme below.

$$\frac{Y \setminus \varsigma}{\Theta(X, \varsigma, \nu) == \Theta(Y, [X := Y]\varsigma, \nu)}$$

The importance of the generalization can be seen if we interpret Θ in that rule as substitution $[X := T]P$. We then have the following.

$$\frac{Y \setminus P}{[X := T]P == [Y := T][X := Y]P}$$

This is exactly the rule needed to change the X in $[X := T]P$, and it is derivable. Similarly, we can express whole families of related and useful

rules using the *bsn* notation. Each can be checked by case analysis to be derivable.

$$\frac{Y \setminus \varsigma}{\Theta(X,\varsigma,\nu) == \Theta(Y,[X := Y]\varsigma,\nu)} \quad \alpha_\Theta$$

$$\frac{\begin{array}{c} SC_P(\rho,\eta) \\ \beta \setminus \rho \quad \beta \setminus \eta \end{array}}{[\![\rho := \eta]\!]\Theta(\beta,\varsigma,\nu) == \Theta(\beta,[\![\rho := \eta]\!]\varsigma,[\![\rho := \eta]\!]\nu)} \quad [\![\Theta]\!]_1$$

$$\frac{}{[X := T]\Theta(X,\varsigma,\nu) == \Theta(X,\varsigma,[X := T]\nu)} \quad [\![\Theta]\!]_2$$

The $\Theta(\beta,\varsigma,\nu)$ representation leads us to adopt the term *bound variable* for any β, saying it has *scope* ς. Similarly, we refer to the rule α_Θ above, as *alpha conversion.*

We need not restrict the *bsn* notation to terms. We can make similar groupings of the operands of predicate symbols, and extend the above descriptions to these. Example representations are $\Theta_4(\epsilon,\epsilon,(P,Q))$ for $P == Q$, $\Theta_5(X,P,\epsilon)$ for $X \setminus P$, and $\Theta_6(X,\epsilon,\epsilon)$ for var X. The following are the rule schemes for formulae.

$$\frac{\begin{array}{c} Y \setminus \varsigma \\ \Theta(X,\varsigma,\nu) \end{array}}{\Theta(Y,[X := Y]\varsigma,\nu)} \quad \alpha_{\Theta_I} \qquad \frac{\begin{array}{c} Y \setminus \varsigma \\ \Theta(Y,[X := Y]\varsigma,\nu) \end{array}}{\Theta(X,\varsigma,\nu)} \quad \alpha_{\Theta_E}$$

$$\frac{\begin{array}{c} SC_P(\rho,\eta) \\ \beta \setminus \rho \quad \beta \setminus \eta \\ \Theta(\beta,\varsigma,\nu) \end{array}}{\Theta(\beta,[\![\rho := \eta]\!]\varsigma,[\![\rho := \eta]\!]\nu)} \quad [\![\Theta]\!]_{I.1} \qquad \frac{\Theta(X,\varsigma,\nu)}{\Theta(X,\varsigma,[X := T]\nu)} \quad [\![\Theta]\!]_{I.2}$$

$$\frac{SC_P(\rho,\eta)\quad \beta \setminus \rho\quad \beta \setminus \eta\quad \Theta(\beta,[\rho:=\eta]\varsigma,[\rho:=\eta]\nu)}{\Theta(\beta,\varsigma,\nu)}\ [\Theta]_{E.1} \qquad \frac{\Theta(X,\varsigma,[X:=T]\nu)}{\Theta(X,\varsigma,\nu)}\ [\Theta]_{E.2}$$

Note: $[\Theta]_E$ *is not derivable when* Θ *is representing* $==$ *or* $\vdash$.

Again, we call the first two rules alpha conversions (introduction and elimination). Because of the link with the standard specialization rule, we refer to the other four as *specialization* rules.

It should be noted that the above rules are assumed derivable by the proof of variable lifting. When new symbols are added to a logic, if the above rules are derivable for that symbol, variable lifting will continue to apply to the logic.

It is possible to further generalize the *bsn* representation to allow a list of scope expressions, and perhaps a list of bound variables. For the purposes of this paper, such generalization is unnecessary.

5.4 Variables

The process of variable lifting replaces object-variables by meta-variables not already existing in the original sequent. We will replace a variable τ by a fresh meta-variable τ^*. We say τ^* is the *corresponding* meta-variable of τ, and write σ^* for the list of meta-variables $\sigma_1^*,\ldots,\sigma_n^*$.

To attempt the transition directly from object-variables to meta-variables would produce conditions of the form $\tau \setminus \tau^*$, as explained in Section 4. To avoid these unwanted conditions, we use intermediate variables, denoted by τ', whose definition ensures the derivability of both $\tau \setminus \tau'$ and $\tau' \setminus \tau^*$. We shall also make use of temporary variables, denoted τ'', whose purpose was demonstrated in Section 4.

The choice of fresh variables with respect to given expressions is formalized by a family of function symbols written $\mathrm{nvar}(\_,\ldots,\_)$, which we assume are part of the meta-logic. They are defined by the following axioms.

$$\frac{}{\mathsf{var}\ \mathsf{nvar}(\phi_1, \ldots, \phi_n)} \qquad \frac{}{\mathsf{nvar}(\phi_1, \ldots, \phi_n) \setminus \phi_i} \quad \text{for } i \text{ such that } 1 \leq i \leq n$$

We introduce abbreviations for intermediate and temporary variables that will be used in the proof of variable lifting. We write σ' for the list of nvar expressions (which we call *temporary variables*) $\sigma'_1, \ldots, \sigma'_n$, and σ'' for the list of nvar expressions (which we call *intermediate variables*) $\sigma''_1, \ldots, \sigma''_n$. The abbreviations σ' and σ'' are defined as follows:

$$\begin{array}{ll} \sigma'_i & \textit{for} \quad \mathsf{nvar}(\sigma'_1, \ldots, \sigma'_{i-1}, \sigma, \sigma^*) \\ \sigma''_i & \textit{for} \quad \mathsf{nvar}(\sigma''_1, \ldots, \sigma''_{i-1}, \sigma', \sigma, \sigma^*, \Pi) \end{array}$$

where σ and Π are the object-variables and meta-variables of a given sequent.

The following properties result from the above definitions, and are collectively referred to as *var.def*.

var σ'	var σ''
distinct σ'	*distinct* σ''
$\sigma' \setminus \sigma$	$\sigma'' \setminus \sigma$
$\sigma' \setminus \sigma^*$	$\sigma'' \setminus \sigma^*$
$\sigma' \setminus \sigma''$	$\sigma'' \setminus \Pi$

5.5 Scope

For a given sequent we shall be interested in the scope of its bound-like variables. We define scope in terms of the set of free object-variables and the set of meta-variables in the expressions involved. They are defined as follows.

Let P be any meta-variable, and x be any object-variable. Then,

$$\begin{array}{ll} \textit{freevars}\ P & = \{\,\} \\ \textit{freevars}\ x & = \{x\} \\ \textit{freevars}\ \Theta(\beta, \varsigma, \nu) & = (\textit{freevars}\ \varsigma - \{\beta\}) \cup \textit{freevars}\ \nu \end{array}$$

$$\begin{array}{ll} \textit{metavars}\ P & = \{P\} \\ \textit{metavars}\ x & = \{\,\} \\ \textit{metavars}\ \Theta(\beta, \varsigma, \nu) & = \textit{metavars}\ \varsigma \cup \textit{metavars}\ \nu \end{array}$$

Let Δ be the set of all formulae for which *scope* is to be defined, and Γ be the union of Δ and all the sub-terms of each element of Δ, and let x and y be any object-variable or meta-variable. Then,

$$y \in \mathit{scope}\ x \quad \Leftrightarrow \quad (\exists\, \Theta(x, \varsigma, \nu) \in \Gamma \bullet y \in \mathit{freevars}\ \varsigma \vee y \in \mathit{metavars}\ \varsigma)$$

6 Statement of Variable Lifting

Section 4 gave an indication of the process of variables lifting. The following is a precise description of variable lifting for an arbitrary sequent. The proof of the derivability of the lifted form is presented in Section 11.

Variable lifting applies only to sequents satisfying the following applicability conditions. Each antecedent formula is constructed from a predicate symbol satisfying the elimination rules of Section 5.3. The consequent formula is constructed from a predicate symbol satisfying the introduction rules of Section 5.3. Function symbols used in the sequent must satisfy the equivalence rules for terms in the same section. No meta-variable occupies the position of a bound variable.

Given a well-formed sequent,

$$\frac{\phi_1(\sigma) \quad \ldots \quad \phi_n(\sigma)}{\psi(\sigma)}$$

where σ is a list containing all variables in the sequent, and that satisfies the applicability conditions, the lifted form is the sequent,

$$\frac{SC \qquad \phi_1(\sigma^*) \quad \ldots \quad \phi_n(\sigma^*)}{\psi(\sigma^*)}$$

where SC are the added side-conditions, defined as follows.

var x^*	for every bound variable x
trm x^*	for every other variable x
$x^* \setminus y^*$	if $y \in scope(x)$ and x and y are distinct
$x^* \setminus y^*$	if there exists a meta-variable P such that $P \in scope(x) \cap scope(y)$ and x and y are distinct

The first two conditions ensure object-level well-formedness; the last two ensure object-level variable capture is avoided.

7 Substitution Introduction and Elimination

We have emphasized the importance of changing variables by substitutions. The following two lemmas describe the introduction and elimination of an arbitrary substitution on the operands of predicate symbols.

Lemma 1 *Introduction onto operands.*
For each formula $\mathcal{P}(\theta_1, \ldots, \theta_m)$, where $\mathcal{P}$ is any predicate symbol satisfying the introduction rules of Section 5.3 and where θ_i are its operands, the following rule is derivable.

$$\frac{\begin{array}{c} SC_P(\rho, \eta) \\ SC_S(\mathcal{P}(\theta_1, \ldots, \theta_m)) \\ \mathcal{P}(\theta_1, \ldots, \theta_m) \end{array}}{\mathcal{P}([\![\rho := \eta]\!]\theta_1, \ldots, [\![\rho := \eta]\!]\theta_m)}$$

where the antecedents $SC_S(\Theta(\beta, \varsigma, \nu))$ are as follows.

If β is an element of ρ (jth, say) then,

var η_j

$\eta_j \setminus [\![\overline{\{j\}} \mid \rho := \eta]\!]\varsigma$

Otherwise,

$\beta \setminus \rho$

$\beta \setminus \eta$

Note: SC_P is defined in Section 5.2.
◇

Proof
Let $\Theta(\beta, \varsigma, \nu)$ be the *bsn* representation of $\mathcal{P}(\theta_1, \ldots, \theta_m)$.

Case: β is an element of ρ (the jth, say).

1.	$SC_P(\rho, \eta)$	*assumption*
2.	$SC_S(\Theta(\rho_j, \varsigma, \nu))$	*assumption*
3.	$\Theta(\rho_j, \varsigma, \nu)$	*assumption*
4.	$\Theta(\rho_j, [\![\overline{\{j\}} \mid \rho := \eta]\!]\varsigma, [\![\overline{\{j\}} \mid \rho := \eta]\!]\nu)$	$[\![\Theta]\!]_{1.1}, 1, 3$
5.	$\Theta(\rho_j,$ $[\![\overline{\{j\}} \mid \rho := \eta]\!]\varsigma,$ $[\![\{j\} \mid \rho := \eta][\![\overline{\{j\}} \mid \rho := \eta]\!]\nu)$	$[\![\Theta]\!]_{1.2}, 4$
6.	$\Theta(\eta_j,$ $[\![\{j\} \mid \rho := \eta][\![\overline{\{j\}} \mid \rho := \eta]\!]\varsigma,$ $[\![\{j\} \mid \rho := \eta][\![\overline{\{j\}} \mid \rho := \eta]\!]\nu)$	$\alpha_{\Theta_1}, 2, 5$
7.	$\Theta(\eta_j, [\![\rho := \eta]\!]\varsigma, [\![\rho := \eta]\!]\nu)$	$A.1, 1, 6$
8.	$\Theta([\![\rho := \eta]\!]\rho_j, [\![\rho := \eta]\!]\varsigma, [\![\rho := \eta]\!]\nu)$	$A.2, 1, 7$

Case: β is not an element of ρ.

1.	$SC_P(\rho, \eta)$	*assumption*
2.	$SC_S(\Theta(\beta, \varsigma, \nu))$	*assumption*
3.	$\Theta(\beta, \varsigma, \nu)$	*assumption*
4.	$\Theta(\beta, [\![\rho := \eta]\!]\varsigma, [\![\rho := \eta]\!]\nu)$	$[\![\Theta]\!]_{1.1}, 1, 2, 3$
5.	$\Theta([\![\rho := \eta]\!]\beta, [\![\rho := \eta]\!]\varsigma, [\![\rho := \eta]\!]\nu)$	$A.3, 1, 2, 4$

□

Lemma 2 *Elimination from operands.*
For each formula $\mathcal{P}(\theta_1, \ldots, \theta_m)$, where $\mathcal{P}$ is any predicate symbol satisfying the elimination rules of Section 5.3, and where θ_i are its operands, the following rule is derivable.

$$\frac{\begin{array}{c} SC_P(\rho, \eta) \\ SC_S(\mathcal{P}(\theta_1, \ldots, \theta_m)) \\ \mathcal{P}([\![\rho := \eta]\!]\theta_1, \ldots, [\![\rho := \eta]\!]\theta_m) \end{array}}{\mathcal{P}(\theta_1, \ldots, \theta_m)}$$

◇

Proof
The proof is the reverse of the preceding proof, with elimination rules used in place of introduction rules.
□

8 Distribution

Lemma 3 *Distribution.*
For all function symbols, $\mathcal{F}$, the following rule is derivable.

$$\frac{SC_P(\rho,\eta)\ \ SC_S(\mathcal{F}(\zeta_1,\ldots,\zeta_m))}{[\![\rho := \eta]\!]\mathcal{F}(\zeta_1,\ldots,\zeta_m) == \mathcal{F}([\![\rho := \eta]\!]\zeta_1,\ldots,[\![\rho := \eta]\!]\zeta_m)}$$

Note: SC_P is defined in Section 5.2, and SC_S is defined for Lemma 1 (Section 7).
◇

Proof
The proof follows the same pattern as those in the preceding section, using, for example, α_Θ in place of α_{Θ_I} and α_{Θ_E}.
□

9 Substitution

Substitutions applied to meta-constants representing object-constants (e.g., 0) have no effect. Thus we can safely omit these from the list of components when writing $\phi(\xi)$, and do so in the remainder of the paper.

9.1 Two lemmas

Our 'specialization' rules allow, for certain formulae, the introduction and elimination of substitutions. These substitutions can then be distributed

to the components of the formula. Thus, from a particular formula we can derive a second in which the structure of the first is preserved, and some substitution applied to the components from which it is constructed. We capture this with the following lemma.

Lemma 4 *Introduction onto components.*
For a formula $\phi(\xi)$ constructed only from terms ξ_i by application of function symbols and those predicate symbols which satisfy the introduction rules of Section 5.3, the following is derivable.

$$\frac{\begin{array}{c} SC_P(\rho, \eta) \\ SC_{SS}(\phi(\xi)) \\ \phi(\xi) \end{array}}{\phi([\rho := \eta]\xi)}$$

where $SC_{SS}(\psi)$ is as follows.

For each formula and its sub-terms θ in the list ψ,

$$SC_S(\theta)$$

Note: SC_P is defined in Section 5.2, and SC_S is defined for Lemma 1 (Section 7).
◇

Proof
By structural induction: the base case is Lemma 1; the step case is Lemma 3.
□

Similarly, from a formula in which a particular substitution is found to apply to all constituent components, we expect to be able to factor out those substitutions to the predicate symbol operands, and remove them by the appropriate specialization elimination rule. We capture this with the following lemma.

Lemma 5 *Elimination from components.*
For a formula $\phi(\xi)$ constructed only from terms ξ_i by application of function

symbols and those predicate symbols which satisfy the elimination rules of Section 5.3, the following is derivable.

$$\frac{SC_P(\rho,\eta) \quad SC_{SS}(\phi(\xi)) \quad \phi([\![\rho := \eta]\!]\xi)}{\phi(\xi)}$$

◇

Proof
By structural induction: the base case is Lemma 2; the step case is Lemma 3.
□

9.2 Substitution Theorem

Theorem 1 *Substitution.*
If each $\phi_i(\xi)$ is constructed only from terms ξ_j by application of function symbols and those predicate symbols which satisfy the elimination rules of Section 5.3, and if $\psi(\xi)$ is constructed only from terms ξ_j by application of function symbols and those predicate symbols which satisfy the introduction rules of Section 5.3, then from the sequent,

$$\frac{\Delta \quad \phi_1(\xi) \quad \ldots \quad \phi_n(\xi)}{\psi(\xi)}$$

where Δ is any set of formulae, it is possible to derive the sequent,

$$\frac{\Delta \quad SC_P(\rho,\eta) \quad SC_{SS}((\phi_0(\xi),\ldots,\phi_n(\xi))) \quad \phi_1([\![\rho := \eta]\!]\xi) \quad \ldots \quad \phi_n([\![\rho := \eta]\!]\xi)}{\psi([\![\rho := \eta]\!]\xi)}$$

◇

Proof
Let the initial sequent be called (I).

0.	Δ	*assumption*
$\dot{0}$.	$SC_P(\rho, \eta)$	*assumption*
$\tilde{0}$.	$SC_{SS}((\phi_0(\xi), \ldots, \phi_n(\xi)))$	*assumption*
1.	$\phi_1([\![\rho := \eta]\!]\xi)$	*assumption*
	...	
n.	$\phi_n([\![\rho := \eta]\!]\xi)$	*assumption*
n+1.	$\phi_1(\xi)$	$\dot{0}, \tilde{0}, 1$, *Lemma* 5
	...	
$2n$.	$\phi_n(\xi)$	$\dot{0}, \tilde{0}, n$, *Lemma* 5
$2n$+1.	$\psi(\xi)$	$(I), 0, n$+1..$2n$
$2n$+2.	$\psi([\![\rho := \eta]\!]\xi)$	$\dot{0}, \ddot{0}, 2n$+1, *Lemma* 4

□

10 Substitution Application

During the proof of the theorem of variable lifting, we shall apply the Substitution Theorem (Theorem 1, Section 9.2) a number of times. In three of these cases the side conditions SC_{SS} and SC_P appearing in the resulting sequent are themselves derivable, justifying their omission. In one case, a small component of SC_{SS} and SC_P is not derivable. It is beneficial to examine these claims now, in the form of theorems, to simplify the forthcoming argument.

This section is long and detailed. The reader may prefer to return to it after the proof of variable lifting in Section 11.

Application of the Substitution Theorem is dealt with by Theorems 2, 3, 4 and 5 below. Lemmas 8 and 9 introduce some useful simplifications.

10.1 Non-freeness

In this section we factor out the complex parts of Theorems 2 and 3 below. Those parts deal with the non-freeness of a variable in a complicated expression.

If a variable does not occur free in the components of a term, then it does not occur free in the term itself:

$$\frac{\omega \setminus \Gamma}{\omega \setminus \phi(\Gamma)} \quad NF.1$$

Given σ_i not in the set *freevars* $\phi(\sigma, \Pi)$, where $\phi(\sigma, \Pi)$ is constructed from variables σ and meta-variables Π, it follows that if σ'_i does not occur free in terms Δ, then σ'_i does not occur free in the term $\phi(\sigma', \Delta)$:

$$\frac{\sigma'_i \setminus \Delta}{\sigma'_i \setminus \phi(\sigma', \Delta)} \quad (\sigma_i \notin \mathit{freevars}\ \phi(\sigma, \Pi)) \quad NF.2$$

Both *NF*.1 and *NF*.2 are derivable. Note that the side-condition of *NF*.2 implies that $i \neq j$ for all j such that $\sigma_j \in \mathit{freevars}\ \phi(\sigma, \Pi)$, and therefore $\sigma'_i \setminus \sigma'_j$ (by *var.def*) for those j, which is required for its proof.

During the proof of variable lifting, meta-variables are 'guarded' by substitutions in such a way that certain non-freeness properties hold. Two of these are expressed by the following lemmas.

Lemma 6
If $\varsigma(\sigma, \Pi)$ is the scope of σ_j then the following sequent is derivable.

$$\overline{\sigma'_j \setminus [\![\overline{\{j\}} \mid \sigma := \sigma']\!]\, \varsigma(\sigma, s\Pi)}$$

where $s\Pi_i$ is $[\![\{j \mid \Pi_i \in \mathit{scope}(\sigma_j)\} \mid \sigma' := \sigma]\!]\, [\![\sigma := \sigma'']\!]\, \Pi_i$

◇

Proof

1. $\sigma'_j \setminus \sigma$ — *var.def*
2. $\sigma'_j \setminus s\Pi_i \quad (\mathit{all}\ i : \Pi_i \in \mathit{metavars}\ \varsigma(\sigma, \Pi))$ — *A*.5, *var.def*
3. $\sigma'_j \setminus \varsigma(\sigma, s\Pi)$ — *NF*.1, 1, 2
4. $\mathit{distinct}\ \sigma'$ — *var.def*
5. $\sigma'_j \setminus [\![\overline{\{j\}} \mid \sigma := \sigma']\!]\varsigma(\sigma, s\Pi)$ — *A*.6, 3, 4, *var.def*

□

Lemma 7
If $\varsigma(\sigma, \Pi)$ is the scope of σ_j then the following sequent is derivable.

$$\frac{SC}{\sigma_j^* \setminus [\![\overline{\{j\}} \mid \sigma' := \sigma^*]\!] \varsigma(\sigma', t\Pi)}$$

where $t\Pi_i$ is $[\![\{j \mid \Pi_i \in scope(\sigma_j)\} \mid \sigma^* := \sigma']\!]_{1..k}[\![\sigma' := \sigma'']\!]\Pi_i$

Note: *SC* is defined in Section 6.
◇

Proof
Let $q = \{i \mid (\sigma_i \in freevars\ \varsigma(\sigma, \Pi) \vee (metavars\ \varsigma(\sigma, \Pi) \cap scope(\sigma_i) \neq \{\})) \wedge i \neq j\}$

1.	$\sigma_j^* \setminus \sigma'$	*var.def*
2.	$\sigma_j^* \setminus t\Pi_i \quad (all\ i : \Pi_i \in metavars\ \varsigma(\sigma, \Pi))$	*A*.5, *var.def*
3.	$\sigma_j^* \setminus \varsigma(\sigma', t\Pi)$	*NF*.1, 1, 2
4.	$\sigma_j^* \setminus \sigma_k^* \quad (all\ k \in q)$	*SC*
5.	$\sigma_j^* \setminus [\![q \mid \sigma' := \sigma^*]\!]\varsigma(\sigma', t\Pi)$	*A*.6, 3, 4, *var.def*
6.	$\sigma_m' \setminus t\Pi \quad (all\ m \in \overline{q} - \{j\})$	*A*.6, *var.def*
7.	$\sigma_m' \setminus \varsigma(\sigma', t\Pi) \quad (all\ m \in \overline{q} - \{j\})$	*NF*.2, 6
8.	$\sigma_j^* \setminus [\![q \mid \sigma' := \sigma^*]\!][\![\overline{q} - \{j\} \mid \sigma' := \sigma^*]\!]\varsigma(\sigma', t\Pi)$	*A*.3, 5, 7, *var.def*
9.	$\sigma_j^* \setminus [\![\overline{\{j\}} \mid \sigma' := \sigma^*]\!]\varsigma(\sigma', t\Pi)$	*A*.1, *var.def*

□

10.2 Deriving SC_{SS}

Theorem 2
Given the well-formed sequent below, in which formulae are constructed from the variables σ and expressions $s\Pi$, and in which no meta-variable occupies a bound variable position,

$$\frac{\phi_1(\sigma, s\Pi) \quad \ldots \quad \phi_m(\sigma, s\Pi)}{\phi_0(\sigma, s\Pi)}$$

where $s\Pi_i$ is $[\![\{j \mid \Pi_i \in scope(\sigma_j)\} \mid \sigma' := \sigma]\!]\,[\![\sigma := \sigma'']\!]\,\Pi_i$

the formulae $SC_{SS}((\phi_0, \ldots, \phi_m))$ and $SC_P(\sigma, \sigma')$ which result from the application of the Substitution Theorem with substitution $[\![\sigma := \sigma']\!]$ are derivable.
◇

Proof
We note that as no meta-variable occupies a bound variable position, all bound variables are drawn from σ. In the following, justification for deriving the formulae on the left is given on the right. The permutability conditions SC_P are easily dismissed, as follows.

$SC_P(\sigma, \sigma')$ *var.def*

For each formula in $\phi_0(\sigma, s\Pi), \ldots, \phi_m(\sigma, s\Pi)$, and for each term in each of those formulae, SC_S must hold. For those formulae or terms $\Theta(\beta, \varsigma, \nu)$ the formulae to be derived are as follows.

β is an element of σ (the jth, say),

var σ'_j *var.def*

$\sigma'_j \setminus [\![\overline{\{j\}} \mid \sigma := \sigma']\!]\varsigma$ *Lemma* 6

□

Theorem 3
Given the well-formed sequent below, in which formulae are constructed from the variables σ' and expressions $t\Pi$, and in which no meta-variable occupies a bound variable position,

$$\frac{\phi_1(\sigma', t\Pi) \quad \ldots \quad \phi_m(\sigma', t\Pi)}{\phi_0(\sigma', t\Pi)}$$

where $t\Pi_i$ is $[\![\{j \mid \Pi_i \in scope(\sigma'_j)\} \mid \sigma^* := \sigma']\!]_{1..A}[\![\sigma' := \sigma'']\!]\Pi_i$

the formulae $SC_{SS}((\phi_0(\sigma', t\Pi), \ldots, \phi_m(\sigma', t\Pi)))$ and $SC_P(\sigma', \sigma^*)$ which arise from the application of the Substitution Theorem with the substitution $[\![\sigma' := \sigma^*]\!]$ are derivable from SC.
◇

Proof
Assume SC. We note that as no meta-variable occupies a bound variable

position, all bound variables are drawn from σ'. Permutability is established as follows.

$SC_P(\sigma', \sigma^*)$ *var.def*

The formulae $SC_{SS}((\phi_0(\sigma', t\Pi), \ldots, \phi_m(\sigma', t\Pi))$ hold if SC_S holds for every formulae in $\phi_0(\sigma', t\Pi), \ldots, \phi_m(\sigma', t\Pi)$, and for every term of those formulae. For each formula or term $\Theta(\beta, \varsigma, \nu)$ tbe following are the formulae.

β is an element of σ' (the jth, say),

var σ_j^* *SC*

$\sigma_j^* \setminus [\![\overline{\{j\}} \mid \sigma' := \sigma^*]\!]\varsigma$ *Lemma 7, SC*

□

Theorem 4

Given the well-formed sequent, in which formulae are constructed from the variables σ' and expressions $[\![\sigma := \sigma'']\!]\Pi$,

$$\frac{\phi_1(\sigma', [\![\sigma := \sigma'']\!]\Pi) \quad \ldots \quad \phi_m(\sigma', [\![\sigma := \sigma'']\!]\Pi)}{\phi_0(\sigma', [\![\sigma := \sigma'']\!]\Pi)}$$

each of the formulae $SC_{SS}((\phi_0(\sigma', [\![\sigma := \sigma'']\!]\Pi), \ldots, \phi_m(\sigma', [\![\sigma := \sigma'']\!]\Pi)))$ and $SC_P(\sigma'', \sigma)$ which arise from the application of the Substitution Theorem with substitution $[\![\sigma'' := \sigma]\!]$ are derivable.

◇

Proof

All bound variables in the sequent are drawn from σ'. Therefore, by *var.def*, each bound variable is distinct from each σ''. That is, there is no index j of σ'' such that σ_j'' is a bound variable, and so the 'otherwise' formulae from SC_S are relevant.

Permutability is established as follows.

$SC_P(\sigma'', \sigma)$ *var.def*

The formulae,

$$SC_{SS}((\phi_0(\sigma', [\![\sigma := \sigma'']\!]\Pi), \ldots, \phi_m(\sigma', [\![\sigma := \sigma'']\!]\Pi)))$$

hold if SC_S holds for every formulae of

$$\phi_0(\sigma', [\![\sigma := \sigma'']\!]\Pi), \ldots, \phi_m(\sigma', [\![\sigma := \sigma'']\!]\Pi)$$

and for each sub-term of each of those formulae. For such formulae or terms $\Theta(\beta, \varsigma, \nu)$ the following are the formulae.

β is not an element of σ'',

$\beta \setminus \sigma''$ *var.def*

$\beta \setminus \sigma$ *var.def*

□

Theorem 5
Given the well-formed sequent, in which formulae are constructed from the variables $\sigma^\star$ and expressions $[\![\sigma' := \sigma'']\!]\Pi$,

$$\frac{\phi_1(\sigma^\star, [\![\sigma' := \sigma'']\!]\Pi) \quad \ldots \quad \phi_m(\sigma^\star, [\![\sigma' := \sigma'']\!]\Pi)}{\phi_0(\sigma^\star, [\![\sigma' := \sigma'']\!]\Pi)}$$

each of the formulae

$$SC_{SS}((\phi_0(\sigma^\star, [\![\sigma' := \sigma'']\!]\Pi), \ldots, \phi_m(\sigma^\star, [\![\sigma' := \sigma'']\!]\Pi)), \sigma'', \sigma')$$

which arise from the application of the Substitution Theorem with substitution $[\![\sigma'' := \sigma']\!]$ are derivable.
◇

Proof
All bound variables in the sequent are drawn from $\sigma^\star$. Therefore, by *var.def*, each bound variable is distinct from each σ''. That is, there is no index j of σ'' such that σ''_j is a bound variable, and so the 'otherwise' formulae from SC_S are relevant.

Permutability is established as follows.

$SC_P(\sigma'', \sigma)$ *var.def*

The formulae

$$SC_{SS}((\phi_0(\sigma^\star, [\![\sigma' := \sigma'']\!]\Pi), \ldots, \phi_m(\sigma^\star, [\![\sigma' := \sigma'']\!]\Pi)), \sigma'', \sigma')$$

hold if SC_S holds for every formulae of

$$\phi_0(\sigma^\star, [\![\sigma' := \sigma'']\!]\Pi), \ldots, \phi_m(\sigma^\star, [\![\sigma' := \sigma'']\!]\Pi)$$

and for each term of each of those formulae. For such formulae or terms $\Theta(\beta, \varsigma, \nu)$ the following are the formulae.

β is not an element of σ'',

$\beta \setminus \sigma''$ *var.def*

$\beta \setminus \sigma'$ *var.def*

□

10.3 Two simplifications

Lemma 8
For any set s, the following is derivable.

$$\frac{\sigma \setminus \phi}{[\![\sigma := \sigma']\!]\ [\![\overline{s} \mid \sigma' := \sigma]\!]\ \phi == \phi}$$

◇

Proof
Assume $\sigma \setminus \phi$.

$[\![\sigma := \sigma']\!]\ [\![\overline{s} \mid \sigma' := \sigma]\!]\ \phi$

$== [\![s \mid \sigma := \sigma']\!]\ [\![\overline{s} \mid \sigma := \sigma']\!]\ [\![\overline{s} \mid \sigma' := \sigma]\!]\ \phi$ *A.1, var.def*

$== [\![s \mid \sigma := \sigma']\!]\ \phi$ *A.4, var.def*

$== \phi$ *A.3, var.def*

□

Lemma 9
For any set s, the following is derivable.

$$\frac{\sigma' \setminus \phi \quad \sigma_i^* \setminus \sigma_j^*}{[\![\sigma' := \sigma^*]\!]\ [\![s \mid \sigma^* := \sigma']\!]\ \phi == \phi} \quad (i, j \in s \wedge i \neq j)$$

◇

Proof
Similar to Lemma 8. □

11 Variable Lifting

Variable lifting is carried out in two almost identical stages. The first replaces all object-variables τ by intermediate object-variables τ' (with correct non-freeness properties); the second replaces those variables by the final meta-variables τ^*. Both stages are performed by identical steps. Only the replaced variable and the replacing expression changes.

In this section we present a rigorous proof that from a given sequent we can derive its lifted sequent. This is of course subject to the initial sequent satisfying certain applicability conditions.

Applicability Conditions
Each antecedent formula $\phi_i(\sigma)$ of the sequent is constructed from a predicate symbol satisfying the elimination rules of Section 5.3. The consequent formula is constructed from a predicate symbol satisfying the introduction rules of Section 5.3. No meta-variable occupies the position of a bound variable.

Theorem 6 *Variable Lifting.*
From an arbitrary sequent which satisfies the applicability conditions,

$$\frac{\phi_1(\sigma) \quad \ldots \quad \phi_m(\sigma)}{\phi_0(\sigma)}$$

where all object-variables are drawn from σ, we can derive the lifted form,

$$\frac{\begin{array}{c} SC \\ \phi_1(\sigma^*) \quad \ldots \quad \phi_m(\sigma^*) \end{array}}{\phi_0(\sigma^*)}$$

Note: *SC* is defined in Section 6.
◇

Proof
We make explicit the meta-variables Π of the sequent. Our initial sequent is then,

$$\frac{\phi_1(\sigma,\Pi) \quad \ldots \quad \phi_m(\sigma,\Pi)}{\phi_0(\sigma,\Pi)}$$

It is from this sequent that we generate σ' and σ'', and which we take for the definition of *scope*.

By instantiation of the meta-variables, we obtain the next sequent.

$$\frac{\phi_1(\sigma,s\Pi) \quad \ldots \quad \phi_m(\sigma,s\Pi)}{\phi_0(\sigma,s\Pi)}$$

where $s\Pi_i$ is $[\![\{j \mid \Pi_i \in scope(\sigma_j)\} \mid \sigma' := \sigma]\!]\ [\![\sigma := \sigma'']\!]\ \Pi_i$

We note that the properties of the variables concerned allow re-ordering of the above substitutions, justifying the lack of subscript.

Using the Substitution Theorem (Theorem 1) we impose the substitution $[\![\sigma := \sigma']\!]$. From Theorem 2 we know the conditions SC_{SS} are derivable in this case, and so we omit SC_{SS} from the resulting sequent. Similarly, $SC_P(\sigma,\sigma')$ are derivable (by *var.def*).

$$\frac{\phi_1([\![\sigma := \sigma']\!]\sigma, [\![\sigma := \sigma']\!]s\Pi) \quad \ldots \quad \phi_m([\![\sigma := \sigma']\!]\sigma, [\![\sigma := \sigma']\!]s\Pi)}{\phi_0([\![\sigma := \sigma']\!]\sigma, [\![\sigma := \sigma']\!]s\Pi)}$$

We next apply the simplification $[\![\sigma := \sigma']\!]\sigma == \sigma'$ and note from Lemma 8 that $[\![\sigma := \sigma']\!]s\Pi_i == [\![\sigma := \sigma'']\!]\Pi_i$.

$$\frac{\phi_1(\sigma', [\![\sigma := \sigma'']\!]\Pi) \quad \ldots \quad \phi_m(\sigma', [\![\sigma := \sigma'']\!]\Pi)}{\phi_0(\sigma', [\![\sigma := \sigma'']\!]\Pi)}$$

We use the Substitution Theorem to introduce $[\![\sigma'' := \sigma]\!]$, with the aim of removing the current substitution on meta-variables. Theorem 4 has shown that we can derive the side conditions SC_{SS} in this case, justifying their

omission from the resulting sequent. Similarly, $SC_P(\sigma'', \sigma)$ are derivable (by *var.def*).

$$\frac{\phi_1([\![\sigma'' := \sigma]\!]\sigma', [\![\sigma'' := \sigma]\!][\sigma := \sigma'']\!]\Pi) \quad \ldots \quad \phi_m([\![\sigma'' := \sigma]\!]\sigma', [\![\sigma'' := \sigma]\!][\sigma := \sigma'']\!]\Pi)}{\phi_0([\![\sigma'' := \sigma]\!]\sigma', [\![\sigma'' := \sigma]\!][\sigma := \sigma'']\!]\Pi)}$$

We note the simplification $[\![\sigma'' := \sigma]\!]\sigma' == \sigma'$ and that $[\![\sigma'' := \sigma]\!][\sigma := \sigma'']\!]\Pi_i == \Pi_i$ follows from *A*.4 allowing derivation of the following sequent.

$$\frac{\phi_1(\sigma', \Pi) \quad \ldots \quad \phi_m(\sigma', \Pi)}{\phi_0(\sigma', \Pi)}$$

This is the half-way point. We have derived a sequent in which each variable σ_i of the initial sequent has been replaced by the intermediate variable σ'_i. The second stage performs the same steps as the first, but our goal is to replace σ' by σ^*. As before, we begin by instantiation of each Π_i. We note carefully the absence of σ'' in the sequent, since they abbreviate a string in which Π_i is present, which would complicate the instantiation.

$$\frac{\phi_1(\sigma', t\Pi) \quad \ldots \quad \phi_m(\sigma', t\Pi)}{\phi_0(\sigma', t\Pi)}$$

where $t\Pi_i$ is $[\![\{j \mid \Pi_i \in scope(\sigma_j)\} \mid \sigma^* := \sigma']\!]_{1..A}[\sigma' := \sigma'']\!]\Pi_i$

The instantiation is similar to our earlier instantiation, though we retain the subscript on the first substitution for the moment. We introduce $[\![\sigma' := \sigma^*]\!]$ using the Substitution Tbeorem. Theorem 3 demonstrates that in this case, SC_{SS} can be derived from SC. Thus we include SC in the antecedents, resulting in the following sequent. We note $SC_P(\sigma', \sigma^*)$ are derivable (by *var.def*).

$$\frac{SC \qquad \phi_1([\![\sigma' := \sigma^*]\!]\sigma', [\![\sigma' := \sigma^*]\!]t\Pi) \quad \ldots \quad \phi_m([\![\sigma' := \sigma^*]\!]\sigma', [\![\sigma' := \sigma^*]\!]t\Pi)}{\phi_0([\![\sigma' := \sigma^*]\!]\sigma', [\![\sigma' := \sigma^*]\!]t\Pi)}$$

From SC we note that the substitution $[\![\{j \mid \Pi_i \in scope(\sigma_j)\} \mid \sigma^* := \sigma']\!]_{1..k}$ is permutable. We note also that $[\![\sigma' := \sigma^*]\!]\sigma' == \sigma^*$ and that the result $[\![\sigma' := \sigma^*]\!]t\Pi_i == [\![\sigma' := \sigma'']\!]\Pi_i$ follows from Lemma 9.

$$\frac{SC \qquad \phi_1(\sigma^*, [\![\sigma' := \sigma'']\!]\Pi) \quad \ldots \quad \phi_m(\sigma^*, [\![\sigma' := \sigma'']\!]\Pi)}{\phi_0(\sigma^*, [\![\sigma' := \sigma'']\!]\Pi)}$$

As before, we introduce a substitution to eliminate the remaining substitutions on meta-variables. We introduce $[\![\sigma'' := \sigma']\!]$ using the Substitution Theorem, noting (from Theorem 3) the conditions SC_{SS} for this instance are derivable and may therefore be omitted from the resulting sequent. $SC_P(\sigma'', \sigma')$ are derivable (by $var.def$) also.

$$\frac{\begin{array}{c} SC \\ \phi_1([\![\sigma'' := \sigma']\!]\sigma^*, [\![\sigma'' := \sigma']\!][\![\sigma' := \sigma'']\!]\Pi) \\ \ldots \\ \phi_m([\![\sigma'' := \sigma']\!]\sigma^*, [\![\sigma'' := \sigma']\!][\![\sigma' := \sigma'']\!]\Pi) \end{array}}{\phi_0([\![\sigma'' := \sigma']\!]\sigma^*, [\![\sigma'' := \sigma']\!][\![\sigma' := \sigma'']\!]\Pi)}$$

Applying the simplification $[\![\sigma'' := \sigma']\!]\sigma^* == \sigma^*$ and noting the result $[\![\sigma'' := \sigma']\!][\![\sigma' := \sigma'']\!]\Pi_i == \Pi_i$ (from $A.4$) allows us to derive the following sequent.

$$\frac{SC \qquad \phi_1(\sigma^*, \Pi) \quad \ldots \quad \phi_m(\sigma^*, \Pi)}{\phi_0(\sigma^*, \Pi)}$$

□

12 Non-Examples

Our purpose in this section is to discuss sequents to which the lifting procedure as described is not applicable. We intend to justify the applicability conditions previously presented. We shall present a number of valid sequents, which fall outside these conditions, and demonstrate that what we may have expected to be their lifted form is invalid.

We note the lack of a specialization elimination rule for the symbol $==$. Thus we expect problems for a sequent with an $==$ expression in the antecedent. Consider the following sequent:

$$\frac{\mathsf{frm}\ P \qquad [x := y]P == P}{P \vdash \forall x \bullet P}$$

We note first that the sequent is valid: only when x is not free in P will the antecedent be true, in which case the consequent follows. If we applied the informal lifting procedure to this (i.e., apply the informal description of lifting), we would have:

$$\frac{\mathsf{frm}\ P \qquad \mathsf{var}\ X \quad \mathsf{trm}\ Y \qquad [X := Y]P == P}{P \vdash \forall X \bullet P}$$

By instantiating X as x, Y as x and P as $x = 1$, however, we arrive at the invalid consequence:

$$\frac{[x := x](x = 1) == (x = 1)}{x = 1 \vdash \forall x \bullet x = 1}$$

Similarly, there is no specialization elimination rule for the symbol $\vdash$, and the following example demonstrates the danger in lifting (informally)

sequents in which $\vdash$ appears in an antecedent.

$$\frac{\mathsf{frm}\ P \qquad \vdash\ [x := y]P \equiv P}{\vdash\ (\forall x \bullet P) \vee (\forall x \bullet \neg P)}$$

The validity of this rule is established as follows. Under the assumption $[x := y]P \equiv P$, since $x \setminus [x := y]P$, we can derive each of,

$$[x := y]P \vdash\ (\forall x \bullet P)$$
$$\neg[x := y]P \vdash\ (\forall x \bullet \neg P)$$

and so derive,

$$\vdash\ (\forall x \bullet P) \vee (\forall x \bullet \neg P)$$

The example informally lifts to:

$$\frac{\mathsf{var}\ X \quad \mathsf{trm}\ Y \qquad \vdash\ [X := Y]P \equiv P}{\vdash\ (\forall X \bullet P) \vee (\forall X \bullet \neg P)}$$

Instantiations of X as x, Y as x and P as $x = 1$ yield an invalid sequent.

$$\frac{\vdash\ [x := x](x = 1) \equiv (x = 1)}{\vdash\ (\forall x \bullet x = 1) \vee (\forall x \bullet \neg\ (x = 1))}$$

Next, consider the lifting of sequents in which meta-variables occur in bound-like positions.

$$\frac{\mathsf{var}\ X}{\vdash\ \exists X \bullet X = y}$$

We note this is valid (by cases: X is y; X is not y). Informally, this would be lifted to the sequent:

$$\frac{\mathsf{trm}\ Y \qquad \mathsf{var}\ X}{\vdash\ \exists X \bullet X = Y}$$

Instantiating X as x and Y as $x + 1$ gives the invalid sequent:

$$\frac{}{\vdash \exists x \bullet x = x + 1}$$

All of the above problems would be removed by the addition in the 'lifted' forms of the extra condition $X \setminus Y$. However, unlike the conditions usually added by the lifing procedure, there is a lack of underlying reason as to why this condition should be added, and good intuitive reasons why it shouldn't.

For example, $X \setminus Y$ in the previous 'lifted' sequent, would forbid instantiation to the valid sequent,

$$\frac{}{\vdash \exists z \bullet z = z}$$

13 Summary

The process of variable lifting is one which is informally employed in every logic text book. The mechanization of logic reasoning requires that this process be fully understood and shown to be sound.

Applicability of variable lifting is not limited to the use of the predicate and function symbols described here. New symbols may be added freely. As long as the appropriate 'specialization' rules hold, and in the case of quantifier-like symbols, the appropriate 'alpha conversion' rule holds, variable lifting will apply to sequents using the new symbol. As with particular predicate symbols described here, if no elimination rule exists, the constraint of only appearing in the consequent of a sequent to be lifted will also apply.

References

[1] J.-R. Abrial. An informal introduction to the b tool. B.P. Project Report, Programming Research Group, Oxford University, 1986.

[2] A. Avron, F.A. Honsell, and I.A. Mason. Using typed lambda calculus to implement formal systems on a machine. Technical Report ECS-LFCS-87-31, Laboratory for the Foundations of Computer Science, Edinburgh University, 1987.

[3] N. Bourbaki. *Theory of Sets.* Hermann and Addison-Wesley, 1968.

[4] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, 1976.

[5] P.H.B. Gardiner and T.N. Vickers. A logic for a theorem proving assistant. Submitted to Science of Computer Programming, 1991.

[6] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 583, October 1969.

[7] E. Mendelson. *Introduction to Mathematical Logic.* Van Nostrand, 1952.

[8] K. Milstead. *A Framework for Describing Formal Systems.* D.Phil. thesis, University of Oxford, 1990.

[9] D. Scott. Notes on the formalization of logic. Study Aids Monograph 2 and 3, Sub-faculty of Philosophy, Oxford University, 1981.

[10] J.M. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall, 1989.

[11] T.N. Vickers. An overview of a theorem proving assistant. *Australian Computer Science Communications*, 12(1):402–411, 1990.

[12] T.N. Vickers and P.H.B. Gardiner. Variable lifting. B.P. Project Report, Programming Research Group, Oxford University, 1989.