Higher-Order Pattern Complement and the Strict λ-Calculus

ALBERTO MOMIGLIANO
University of Leicester
and
FRANK PFENNING
Carnegie Mellon University

We address the problem of complementing higher-order patterns without repetitions of existential variables. Differently from the first-order case, the complement of a pattern cannot, in general, be described by a pattern, or even by a finite set of patterns. We therefore generalize the simply-typed λ-calculus to include an internal notion of *strict function* so that we can directly express that a term must depend on a given variable. We show that, in this more expressive calculus, finite sets of patterns without repeated variables are closed under complement and intersection. Our principal application is the transformational approach to negation in higher-order logic programs.

Categories and Subject Descriptors: D.3.3 [**Programming Languages**]: Language Constructs and Features; D.1.6 [**Programming Techniques**]: Logic Programming; F.4.1 [**Mathematical Logic and Formal Language**]: Mathematical Logic—*Lambda calculus and related systems*

General Terms: Languages, Theory

Additional Key Words and Phrases: Complement, higher-order patterns, strict λ-calculus

1. INTRODUCTION

In most functional and logic programming languages the notion of a pattern, together with the requisite algorithms for matching or unification, play an important role in the operational semantics. Besides unification, other problems such as generalization or complement also arise frequently. In this paper we are concerned with the problem of pattern complement in a setting where patterns may contain binding operators, so-called *higher-order patterns* [Miller 1991; Nipkow 1991]. Higher-order patterns have found applications in logic programming [Miller 1991; Pfenning 1991a], logical frameworks [Despeyroux et al. 1997], term rewriting [Nipkow 1993], and functional logic programming [Hanus and Prehofer 1996]. Higher-order pat-

Author's addresses:
A. Momigliano, Department of Mathematics and Computer Science, University of Leicester, Leicester, LE1 HR2, U.K., `am133@mcs.le.ac.uk`
F. Pfenning, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A., `fp@cs.cmu.edu`
This work has been support by the National Science Foundation under grant CCR-9988281.

terns inherit many pleasant properties from the first-order case. In particular, most general unifiers [Miller 1991] and least general generalizations [Pfenning 1991b] exist, even for complex type theories.

Unfortunately, the complement operation does not generalize as smoothly. Lugiez [1995] has studied the more general problem of higher-order disunification and had to go outside the language of patterns and terms to describe complex constraints on sets of solutions. We can isolate one basic difficulty: a pattern such as $\lambda x.\ E\ x$ for an existential variable E matches any term of appropriate type, while $\lambda x.\ E$ matches precisely those terms $\lambda x.\ M$ where M does not depend on x. The complement then consists of all terms $\lambda x.\ M$ such that M *does* depend on x. However, this set cannot be described by a pattern, or even a finite set of patterns.

This formulation of the problem suggests that we should consider a λ-calculus with an internal notion of *strictness* so that we can directly express that a term must depend on a given variable. For reasons of symmetry and elegance we also add the dual concept of *invariance* expressing that a given term does *not* depend on a given variable. As in the first-order case, it is useful to single out the case of *linear patterns*, namely those where no existential variable occurs more than once.[1] We further limit attention to *simple* patterns, that is, those where constructors must be strict in their arguments—a condition naturally satisfied in our intended application domains of functional and logic programming. Simple linear patterns in our λ-calculus of strict and invariant function spaces then have the following properties:

(1) The complement of a pattern is a finite set of patterns.

(2) Unification of two patterns is decidable and finitary.

Consequently, finite sets of simple linear patterns in the strict λ-calculus are closed under complement and unification. If we think of finite sets of linear patterns as representing the set of all their ground instances, then they form a boolean algebra under set-theoretic union union, intersection (implemented via unification) and the complement operation.

The paper is organized as follows: Section 2 briefly reviews related work and introduces some preliminary definitions. In Section 3 we introduce a strict λ-calculus and prove some basic properties culminating in the proof of the existence of canonical forms in Section 4. Section 5 introduces simple terms, followed by the algorithm for complementation in Section 6. In Section 7 we give a corresponding unification algorithm. Section 8 observes how the set of those patterns can be arranged in a boolean algebra. We conclude in Section 9 with some applications and speculations on future research.

2. PRELIMINARIES AND RELATED WORK

A pattern t with free variables can be seen as a representation of the set of its ground instances, denoted by $||t||$. According to this interpretation, the *complement* of t is the set of ground terms that are *not* instances of t, i.e., the terms are in the set-theoretic complement of $||t||$. It is natural to generalize this to finite sets of terms,

[1] This notion of linearity should not be confused with the eponymous concept in linear logic and λ-calculus.

where $\|t_1, \ldots, t_n\| = \|t_1\| \cup \cdots \cup \|t_n\|$. If we take this one step further we obtain the important problem of *relative complement*; this corresponds to computing a suitable representation of all the ground instances of a given (finite) set of terms which are not instances of another given one, written as $\|t_1, \ldots, t_n\| - \|u_1, \ldots, u_m\|$.

Complement problems have a number of applications in theoretical computer science (see Comon [1991] for a list of references). For example, they are used in functional programming to produce unambiguous function definitions by patterns and to improve their compilation. In rewriting systems they are used to check whether an algebraic specification is sufficiently complete. They can also be employed to analyze communicating processes expressed by infinite transition systems. Other applications lie in the areas of machine learning and inductive theorem proving. In logic programming, Kunen [1987] used term complement to represent infinite sets of answers to negative queries. Our main motivation has been the explicit synthesis of the negation of higher-order logic programs [Momigliano 2000a; 2000b], as discussed briefly in Section 9.

Lassez and Marriot [1987] proposed the seminal *uncover* algorithm for computing *first-order* relative complements and introduced the now familiar restriction to linear terms. We quote the definition of the "Not" algorithm for the (singleton) complement problem given in [Barbuti et al. 1990] which we generalize in Definition 6.1. Given a finite signature Σ and a linear term t they define:

$$\begin{array}{lcl}
\mathrm{Not}_\Sigma(x) & = & \emptyset \\
\mathrm{Not}_\Sigma(f(t_1, \ldots, t_n)) & = & \{g(x_1, \ldots, x_m) \mid g \in \Sigma \text{ and } g \neq f\} \\
 & \cup & \{f(z_1, \ldots, z_{i-1}, s, z_{i+1}, \ldots, z_n) \mid s \in \mathrm{Not}_\Sigma(t_i), 1 \leq i \leq n\}
\end{array}$$

The relative complement problem is then solved by composing the above complement operation with term intersection implemented via first-order unification.

An alternative solution to the relative complement problem is *disunification* (see [Comon 1991] for a survey and [Lugiez 1995] for an extension to the simply-typed λ-calculus). Here, operations on sets of terms are translated into conjunctions or disjunctions of equations and dis-equations under explicit quantification. Nondeterministic application of a few dozen rules eventually turns a given problem into a solved form. Though a reduction to a significant subset of the disunification rules is likely to be attainable for complement problems, control is a major problem. We argue that using disunification for this purpose is unnecessarily general. Moreover, the higher-order case results in additional complications, such as restrictions on the occurrences of bound variables, which fall outside an otherwise clean framework. As we show in this paper, this must not necessarily be the case. We believe that our techniques can also be applied to analyze disunification, although we have not investigated this possibility at present.

We now introduce some preliminary definitions and examples which guide our development. We begin with the simply-typed λ-calculus. We write a for atomic types, c for term-level constants, and x for term-level variables. Note that variables x should be seen as parameters and not subject to instantiation.

$$
\begin{array}{rrcl}
\textit{Simple Types} & A & ::= & a \mid A_1 \to A_2 \\
\textit{Terms} & M & ::= & c \mid x \mid \lambda x{:}A.\ M \mid M_1\ M_2 \\
\textit{Signatures} & \Sigma & ::= & \cdot \mid \Sigma, a{:}\mathrm{type} \mid \Sigma, c{:}A \\
\textit{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x{:}A
\end{array}
$$

We require that signatures and contexts declare each constant or variable at most once. Furthermore, we identify contexts that differ only in their order and promote ',' to denote disjoint set union. As usual we identify terms which differ only in the names of their bound variables. We restrict attention to well-typed terms, omitting the standard typing rules. We write the main typing judgment as $\Gamma \vdash M : A$, assuming a fixed signature Σ.

In applications such as logic programming or logical frameworks, λ-abstraction is used to represent binding operators in some object language. In such a situation the most appropriate notion of normal form is the long $\beta\eta$-normal form (which we call *canonical form*), since canonical forms are almost always the terms in bijective correspondence with the objects we are trying to represent. Every well-typed term in the simply-typed λ-calculus has a unique canonical form—a property which persists in the strict λ-calculus introduced in Section 3.

We denote existential variables of type A (also called logical variables, meta-variables, or pattern variables) by E_A, although we mostly omit the type A when it is clear from the context. We think of existential variables as syntactically distinct from bound variables or free variables declared in a context. A term possibly containing some existential variables is called a *pattern* if each occurrence of an existential variable appears in a subterm of the form $E\ x_1 \dots x_n$, where the arguments x_i are distinct occurrences of free or bound variables (but not existential variables). We call a term *ground* if it contains no existential variables. Note that it may still contain parameters.

Semantically, an existential variable E_A stands for all canonical terms M of type A in the empty context with respect to a given signature. We extend this to arbitrary well-typed patterns in the usual way, and write $\Gamma \vdash M \in \|N\| : A$ when a term M is a instance of a pattern N at type A containing only the parameters in Γ and no existential variables. In this setting, unification of two patterns without shared existential variables corresponds to an intersection of the set of terms they denote [Miller 1991; Pfenning 1991b]. This set is always either empty, or can be expressed again as the set of instances of a single pattern. That is, patterns admit most general unifiers.

The class of higher-order patterns inherits many properties from first-order terms. However, as we will see, it is *not* closed under complement, but a special subclass is. We call a canonical pattern $\Gamma \vdash M : A$ *fully applied* if each occurrence of an existential variable E under binders $y_1, \dots, y_m$ is applied to some permutation of the variables in Γ and $y_1, \dots, y_m$. Fully applied patterns play an important role in functional logic programming and rewriting [Hanus and Prehofer 1996], because any fully applied existential variable $\Gamma \vdash E\ x_1 \dots x_n : a$ denotes all canonical terms of type a with parameters from Γ. It is this property which makes complementation particularly simple.

Example 2.1. Consider the untyped λ-calculus:[2]

$$e ::= x \mid \Lambda x.\, e \mid e_1 \mathbin{@} e_2$$

We encode these expressions using the usual technique of higher-order abstract syntax as canonical forms over the following signature.

$$\Sigma = \mathit{exp} : \text{type}, \mathit{lam} : (\mathit{exp} \to \mathit{exp}) \to \mathit{exp}, \mathit{app} : \mathit{exp} \to \mathit{exp} \to \mathit{exp}$$

The representation function $\ulcorner \_ \urcorner$ is defined as follows:

$$\begin{aligned} \ulcorner x \urcorner &= x : \mathit{exp} \\ \ulcorner \Lambda x.\, e \urcorner &= \mathit{lam}\ (\lambda x{:}\mathit{exp}.\, \ulcorner e \urcorner) \\ \ulcorner e_1 \mathbin{@} e_2 \urcorner &= \mathit{app}\ \ulcorner e_1 \urcorner\, \ulcorner e_2 \urcorner \end{aligned}$$

The representation of an object-language β-redex then has the form

$$\ulcorner (\Lambda x.\, e) \mathbin{@} f \urcorner = \mathit{app}\ (\mathit{lam}\ (\lambda x{:}\mathit{exp}.\, \ulcorner e \urcorner))\ \ulcorner f \urcorner,$$

where $\ulcorner e \urcorner$ may have free occurrences of x. When written as a pattern with existential variables $E_{\mathit{exp} \to \mathit{exp}}$ and F_{exp} this is expressed as

$$\mathit{app}\ (\mathit{lam}\ (\lambda x{:}\mathit{exp}.\, E\ x)\ F).$$

Note that in the empty context this pattern is fully applied. Its complement with respect to the empty context contains every top-level abstraction plus every application where the first argument is not an abstraction.

$$\text{Not}(\mathit{app}\ (\mathit{lam}\ (\lambda x{:}\mathit{exp}.\, E\ x)\ F)) = \{\mathit{lam}\ (\lambda x{:}\mathit{exp}.\, H\ x), \mathit{app}\ (\mathit{app}\ H_1\ H_2)\ H_3\}$$

Here H, H_1, H_2, H_3 are fresh existential variables of appropriate type, namely $H : \mathit{exp} \to \mathit{exp}$ and $H_i : \mathit{exp}$.

For patterns that are not fully applied, the complement cannot be expressed as a finite set of patterns, as the following example illustrates.

Example 2.2. The encoding of an η-redex takes the form:

$$\ulcorner \Lambda x.\, (e \mathbin{@} x) \urcorner = \mathit{lam}\ (\lambda x{:}\mathit{exp}.\, \mathit{app}\ \ulcorner e \urcorner\ x)$$

where $\ulcorner e \urcorner$ may contain no free occurrence of x. The side condition is expressed in a pattern by introducing an existential variable E_{exp} which *does not* depend on x, that is

$$\mathit{lam}\ (\lambda x{:}\mathit{exp}.\, \mathit{app}\ E\ x).$$

Hence, its complement with respect to the empty context should contain, among others, also all terms

$$\mathit{lam}\ (\lambda x{:}\mathit{exp}.\, \mathit{app}\ (F\ x)\ (H\ x))$$

where $F : \mathit{exp} \to \mathit{exp}$ *must* depend on its argument x while $H : \mathit{exp} \to \mathit{exp}$ may or may not depend on x.

[2]We use Λ and @ to avoid confusion with λ and application in the language of patterns.

As the example above shows, the complement of patterns that are not fully applied cannot be represented as a finite set of patterns. Indeed, there is no finite set of patterns which has as its ground instances exactly those terms M which depend on a given variable x. This failure of closure under complementation cannot be avoided similarly to the way in which left-linearization bypasses the limitation to linear patterns and it needs to be addressed directly.

One approach is taken by Lugiez [1995]: he modifies the language of terms to permit occurrence constraints. For example $\lambda xyz.\ M\{1,3\}$ would denote a function which depends on its first and third argument. The technical handling of those objects then becomes awkward as they require specialized rules which are foreign to the issues of complementation.

Since our underlying λ-calculus is typed, we use typing to express that a function *must* depend on a variable x. Following standard terminology, we call such terms *strict in* x and the corresponding function $\lambda x{:}A.\ M$ a *strict function*. In the next section we develop such a λ-calculus and then generalize the complement algorithm to work on such terms.

3. STRICT TYPES

As we have seen in the preceding section, the complement of a partially applied pattern in the simply-typed λ-calculus cannot be expressed in a finitary manner within the same calculus. We thus generalize our language to include *strict* functions of type $A \xrightarrow{1} B$ (which are guaranteed to depend on their argument) and *invariant* functions of type $A \xrightarrow{0} B$ (which are guaranteed *not* to depend on their argument). Of course, any concretely given function either will or will not depend on its argument, but in the presence of higher-order functions and existential variables we still need the ability to remain uncommitted. Therefore our calculus also contains the full function space $A \xrightarrow{u} B$. We first concentrate on a version without existential variables. A similar calculus has been independently investigated by Wright [1992] and Baker-Finch [1993]; for a comparison see the end of Section 4.

$$
\begin{array}{rrcl}
\textit{Labels} & k & ::= & 1 \mid 0 \mid u \\
\textit{Types} & A & ::= & a \mid A_1 \xrightarrow{k} A_2 \\
\textit{Terms} & M & ::= & c \mid x \mid \lambda x^k{:}A.\ M \mid M_1\ M_2^k
\end{array}
$$

Note that there are three different forms of abstractions and applications, where the latter are distinguished by different labels on the argument. It is not really necessary to distinguish three forms of application syntactically, since the type of a function determines the status of its argument, but it is convenient for our purposes. A label u is called *undetermined*, otherwise it is *determined* and denoted by d.

We use a formulation of the typing judgment

$$\Gamma; \Omega; \Delta \vdash M : A$$

with three zones: Γ containing *unrestricted* hypotheses, Ω containing the *irrelevant* hypotheses, and Δ containing the *strict* hypotheses. We implicitly assume a fixed signature Σ which would otherwise clutter the presentation. Recall that Γ_1, Γ_2 is the union of two contexts that do not declare any common variables. Recall also that we consider contexts as sets, that is, exchange is left implicit. The typing rules are given in Figure 1.

$$\frac{c{:}A \in \Sigma}{\Gamma; \Omega; \cdot \vdash c : A}\ \texttt{Con}$$

$$\frac{}{(\Gamma, x{:}A); \Omega; \cdot \vdash x : A}\ \texttt{Id}^u \qquad \textit{no}\ \texttt{Id}^0\ \textit{rule} \qquad \frac{}{\Gamma; \Omega; x{:}A \vdash x : A}\ \texttt{Id}^1$$

$$\frac{(\Gamma, x{:}A); \Omega; \Delta \vdash M : B}{\Gamma; \Omega; \Delta \vdash \lambda x^u{:}A.\ M : A \stackrel{u}{\to} B}\ \stackrel{u}{\to} I$$

$$\frac{\Gamma; (\Omega, x{:}A); \Delta \vdash M : B}{\Gamma; \Omega; \Delta \vdash \lambda x^0{:}A.\ M : A \stackrel{0}{\to} B}\ \stackrel{0}{\to} I$$

$$\frac{\Gamma; \Omega; (\Delta, x{:}A) \vdash M : B}{\Gamma; \Omega; \Delta \vdash \lambda x^1{:}A.\ M : A \stackrel{1}{\to} B}\ \stackrel{1}{\to} I$$

$$\frac{\Gamma; \Omega; \Delta \vdash M : A \stackrel{u}{\to} B \qquad (\Gamma, \Delta); \Omega; \cdot \vdash N : A}{\Gamma; \Omega; \Delta \vdash M\ N^u : B}\ \stackrel{u}{\to} E$$

$$\frac{\Gamma; \Omega; \Delta \vdash M : A \stackrel{0}{\to} B \qquad (\Gamma, \Omega, \Delta); \cdot; \cdot \vdash N : A}{\Gamma; \Omega; \Delta \vdash M\ N^0 : B}\ \stackrel{0}{\to} E$$

$$\frac{(\Gamma, \Delta_N); \Omega; \Delta_M \vdash M : A \stackrel{1}{\to} B \qquad (\Gamma, \Delta_M); \Omega; \Delta_N \vdash N : A}{\Gamma; \Omega; (\Delta_M, \Delta_N) \vdash M\ N^1 : B}\ \stackrel{1}{\to} E$$

Fig. 1. Typing rules for $\Gamma; \Omega; \Delta \vdash M : A$

Our system is biased towards a bottom-up reading of the rules in that variables never disappear, i.e., they are always propagated from the conclusion to the premises, although their status might be changed.

Let us go through the typing rules in detail. The requirement for the strict context Δ to be empty in the $\texttt{Id}^u$ and $\texttt{Id}^1$ rules expresses that strict variables must be used, while undetermined variables in Γ or irrelevant variables in Ω can be ignored. Note that there is no rule for irrelevant variables, which expresses that they cannot be used. The introduction rules for undetermined, invariant, and strict functions simply add a variable to the appropriate context and check the body of the function. The difficult rules are the three elimination rules. First, the unrestricted context Γ is always propagated to both premises. This reflects that we place no restriction on the use of these variables.

Next we consider the strict context Δ: recall that this contains the variables which should occur strictly in a term. An undetermined function $M : A \stackrel{u}{\to} B$ may or may not use its argument. An occurrence of a variable in the argument to such a function can therefore not be guaranteed to be used. Hence we must require in the rule $\stackrel{u}{\to} E$ for an application $M\ N^u$ that all variables in Δ occur strictly in M. This ensures at least one strict occurrence in M and no further restrictions on occurrences of strict variables in the argument are necessary. This is reflected in the rule by adding Δ to the unrestricted context while checking the argument N. The treatment of the strict variables in the vacuous application $M\ N^0$ is similar.

$$\dfrac{\dfrac{\dfrac{}{y;\cdot;x \vdash x : A \xrightarrow{1} A \xrightarrow{1} B}\,\mathtt{Id}^1 \qquad \dfrac{}{x;\cdot;y \vdash y : A}\,\mathtt{Id}^1}{\cdot;\cdot;(x,y) \vdash x\ y^1 : A \xrightarrow{1} B}\,{\xrightarrow{1}}E \qquad \dfrac{}{(x,y);\cdot;\cdot \vdash y : A}\,\mathtt{Id}^u}{\cdot;\cdot;(x{:}A \xrightarrow{1} A \xrightarrow{1} B, y{:}A) \vdash (x\ y^1)\ y^1 : B}\,{\xrightarrow{1}}E$$

Fig. 2. First derivation of $\cdot;\cdot;(x{:}A \xrightarrow{1} A \xrightarrow{1} B, y{:}A) \vdash (x\ y^1)\ y^1 : B$

$$\dfrac{\dfrac{\dfrac{}{y;\cdot;x \vdash x : A \xrightarrow{1} A \xrightarrow{1} B}\,\mathtt{Id}^1 \qquad \dfrac{}{(x,y);\cdot;\cdot \vdash y : A}\,\mathtt{Id}^u}{y;\cdot;x \vdash x\ y^1 : A \xrightarrow{1} B}\,{\xrightarrow{1}}E \qquad \dfrac{}{x;\cdot;y \vdash y : A}\,\mathtt{Id}^1}{\cdot;\cdot;(x{:}A \xrightarrow{1} A \xrightarrow{1} B, y{:}A) \vdash (x\ y^1)\ y^1 : B}\,{\xrightarrow{1}}E$$

Fig. 3. Second derivation of $\cdot;\cdot;(x{:}A \xrightarrow{1} A \xrightarrow{1} B, y{:}A) \vdash (x\ y^1)\ y^1 : B$

In the case of a strict application $M\ N^1$ each strict variable should occur strictly in either M or N. We therefore split the context into Δ_M and Δ_N guaranteeing that each variable has at least one strict occurrence in M or N, respectively. However, strict variables can occur more than once, so variables from Δ_N can be used freely in M, and variables from Δ_M can occur freely in N. As before, we reflect this by adding these variables to the unrestricted context.

Finally we consider the irrelevant context Ω. Variables declared in Ω cannot be used *except* in the argument to an invariant function (which is guaranteed to ignore its argument). We therefore add the irrelevant context Ω to the unrestricted context when checking the argument of a vacuous application $M\ N^0$.

We now illustrate how the strict application rule non-deterministically splits contexts. Consider the typing problem $\cdot;\cdot;(x{:}A \xrightarrow{1} A \xrightarrow{1} B, y{:}A) \vdash (x\ y^1)\ y^1 : B$, related to the contraction principle. There are four ways to split the strict context for the outer application.

$$\begin{array}{ll}
\Delta_M = x{:}A \xrightarrow{1} A \xrightarrow{1} B, y{:}A & \Delta_N = \cdot \\
\Delta_M = x{:}A \xrightarrow{1} A \xrightarrow{1} B & \Delta_N = y{:}A \\
\Delta_M = y{:}A & \Delta_N = x{:}A \xrightarrow{1} A \xrightarrow{1} B \\
\Delta_M = \cdot & \Delta_N = x{:}A \xrightarrow{1} A \xrightarrow{1} B, y{:}A
\end{array}$$

Only the first two yield a valid derivation as depicted in Figures 2 and 3. Here we have dropped the types in the context.

Our strict λ-calculus satisfies the expected properties, culminating in the existence of canonical forms which is critical for the intended applications. First we remark that types are unique, although typing derivations may not.

THEOREM 3.1 UNIQUENESS OF TYPING. *Assume* $(\Gamma, \Omega, \Delta) = (\Gamma', \Omega', \Delta')$. *If* $\Gamma; \Omega; \Delta \vdash M : A$ *and* $\Gamma'; \Omega'; \Delta' \vdash M : A'$*, then* $A = A'$.

PROOF. By induction on the structure of the given derivation, exploiting uniqueness for declarations of variables and constants. □

We start addressing the structural properties of the contexts. Exchange is directly built into the formulation and will not be repeated. Note that our calculus is formulated entirely without structural rules, which now have to be shown to be admissible.

LEMMA 3.2 WEAKENING.

(1) *(Weakening^u) If* $\Gamma; \Omega; \Delta \vdash M : A$*, then* $(\Gamma, x{:}C); \Omega; \Delta \vdash M : A$.

(2) *(Weakening^0) If* $\Gamma; \Omega; \Delta \vdash M : A$*, then* $\Gamma; (\Omega, x{:}C); \Delta \vdash M : A$.

PROOF. By induction on the structure of the given derivations. □

The following properties allow us to lose track of strict and vacuous occurrences, if we are so inclined.

LEMMA 3.3 LOOSENING.

(1) *(Loosening^0) If* $\Gamma; (\Omega, x{:}C); \Delta \vdash M : A$*, then* $(\Gamma, x{:}C); \Omega; \Delta \vdash M : A$.

(2) *(Loosening^1) If* $\Gamma; \Omega; (\Delta, x{:}C) \vdash M : A$*, then* $(\Gamma, x{:}C); \Omega; \Delta \vdash M : A$.

PROOF. By induction on the structure of the given derivations. □

Next we come to the critical substitution properties. They verify the intended meaning of the hypothetical judgments and directly entail subject reduction (Theorem 3.5). To be consistent with the design of our typing rules, we formulate the substitution properties so that each of the given derivation depends on the same variables, although their status might be different (unrestricted, irrelevant, or strict). Note that this is possible only because we have included irrelevant hypotheses in our judgment.

LEMMA 3.4 SUBSTITUTION.

(1) *(Substitution^u) If* $(\Gamma, x{:}A); \Omega; \Delta \vdash M : C$ *and* $(\Gamma, \Delta); \Omega; \cdot \vdash N : A$*, then* $\Gamma; \Omega; \Delta \vdash [N/x]M : C$.

(2) *(Substitution^0) If* $\Gamma; (\Omega, x{:}A); \Delta \vdash M : C$ *and* $(\Gamma, \Delta, \Omega); \cdot; \cdot \vdash N : A$*, then* $\Gamma; \Omega; \Delta \vdash [N/x]M : C$.

(3) *(Substitution^1) If* $(\Gamma, \Delta_N); \Omega; (\Delta_M, x{:}A) \vdash M : C$ *and* $(\Gamma, \Delta_M); \Omega; \Delta_N \vdash N : A$*, then* $\Gamma; \Omega; (\Delta_M, \Delta_N) \vdash [N/x]M : C$.

PROOF. We proceed by mutual induction on the structure of the derivation $\mathcal{D}$ of $M : C$, using weakening and loosening as needed to match the form of the induction hypothesis. Each case is otherwise entirely straightforward. We show only one case in the proof of strict substitution (part 3). Here and in subsequent proofs we sometimes write $\mathcal{D} :: J$ if $\mathcal{D}$ is a derivation of judgment J instead of the two-dimensional notation $\begin{array}{c}\mathcal{D}\\ J\end{array}$.

Case. $\mathcal{D}$ ends in $\stackrel{1}{\to}E$. There are two sub-cases, depending on whether the declaration $x{:}A$ is strict in the left premise or right premise. We show the former.

$$\frac{\begin{array}{c}\mathcal{D}_1\\ (\Gamma, \Delta_N, \Delta_Q); \Omega; (\Delta_P, x{:}A) \vdash P : B \stackrel{1}{\to} C\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ (\Gamma, \Delta_N, \Delta_P, x{:}A); \Omega; \Delta_Q \vdash Q : B\end{array}}{(\Gamma, \Delta_N); \Omega; (\Delta_P, x{:}A, \Delta_Q) \vdash P\ Q^1 : C} \stackrel{1}{\to} E$$

$\mathcal{D}_1 :: (\Gamma, \Delta_N, \Delta_Q); \Omega; (\Delta_P, x{:}A) \vdash P : B \xrightarrow{1} C$	Subderivation
$\mathcal{E} :: (\Gamma, \Delta_P, \Delta_Q); \Omega; \Delta_N \vdash N : A$	Assumption
$(\Gamma, \Delta_Q); \Omega; (\Delta_P, \Delta_N) \vdash [N/x]P : B \xrightarrow{1} C$	By i.h. (3) on $\mathcal{D}_1, \mathcal{E}$
$(\Gamma, \Delta_Q, \Delta_N); \Omega; \Delta_P \vdash [N/x]P : B \xrightarrow{1} C$	By Loosening^1 Δ_N
$\mathcal{D}_2 :: (\Gamma, \Delta_N, \Delta_P, x{:}A); \Omega; \Delta_Q \vdash Q : B$	Subderivation
$\mathcal{E}' :: (\Gamma, \Delta_P, \Delta_Q, \Delta_N); \Omega; \cdot \vdash N : A$	By Loosening^1 Δ_N in $\mathcal{E}$
$(\Gamma, \Delta_N, \Delta_P); \Omega; \Delta_Q \vdash [N/x]Q : B$	By i.h. (1) on $\mathcal{D}_2, \mathcal{E}'$
$\Gamma; \Omega; (\Delta_P, \Delta_Q, \Delta_N) \vdash [N/x](P\ Q^1) : C$	By rule $\xrightarrow{1}E$

□

Weakening, loosening, and substitution directly imply the contraction property for all three kinds of hypotheses. Since we do not use contraction in this paper, we elide the formal statement and proof of this property.

The notions of reduction and expansion derive directly from the ordinary β and η rules.

$$(\lambda x^k{:}A.\ M)\, N^k \ \xrightarrow{\beta}\ [N/x]M$$

$$(M : A \xrightarrow{k} B) \ \xrightarrow{\bar{\eta}}\ \lambda x^k{:}A.\ M\ x^k$$

An application of η-expansion rules requires the term M to have the indicated type. The subject reduction and expansion theorems are an immediate consequence of the structural and substitution properties.

THEOREM 3.5 SUBJECT REDUCTION.
If $\Gamma; \Omega; \Delta \vdash M : A$ *and* $M \xrightarrow{\beta} M'$ *then* $\Gamma; \Omega; \Delta \vdash M' : A$.

PROOF. We proceed by cases and inversion followed by an appeal to the substitution property. We show only one case. Let $M = (\lambda x^1{:}B.\ P)\ Q^1 : A$ and $M' = [Q/x]P$.

$\Gamma; \Omega; \Delta \vdash (\lambda x^1{:}B.\ P)\ Q^1 : A$	Assumption
$\Delta = (\Delta_P, \Delta_Q)$, $\mathcal{E} :: (\Gamma, \Delta_P); \Omega; \Delta_Q \vdash Q : B$, and	
$(\Gamma, \Delta_Q); \Omega; \Delta_P \vdash \lambda x^1{:}B.\ P : B \xrightarrow{1} A$	By inversion
$\mathcal{D} :: (\Gamma, \Delta_Q); \Omega; (\Delta_P, x{:}B) \vdash P : A$	By further inversion
$\Gamma; \Omega; (\Delta_P, \Delta_Q) \vdash [Q/x]P : A$	By substitution^1 on $\mathcal{D}, \mathcal{E}$

□

Subject reduction continues to hold if we allow the reduction of an arbitrary subterm occurrence. We omit the obvious statement and formal proof of this fact.

THEOREM 3.6 SUBJECT EXPANSION.
If $\Gamma; \Omega; \Delta \vdash (M : A) \xrightarrow{k} B$ *and* $(M : A \xrightarrow{k} B) \xrightarrow{\bar{\eta}} M'$ *then* $\Gamma; \Omega; \Delta \vdash M' : A \xrightarrow{k} B$.

PROOF. Direct. We consider only the strict case ($k = 1$).

$\Gamma; \Omega; \Delta \vdash M : A \xrightarrow{1} B$	Assumption
$(\Gamma, x{:}A); \Omega; \Delta \vdash M : A \xrightarrow{1} B$	By weakening^u
$(\Gamma, \Delta); \Omega; x{:}A \vdash x : A$	By rule $\mathtt{Id}^1$
$\Gamma; \Omega; (\Delta, x{:}A) \vdash M\ x^1 : B$	By rule $\xrightarrow{1}E$

$\Gamma; \Omega; \Delta \vdash \lambda x^1{:}A.\ M\ x^1 : A \xrightarrow{1} B$ By rule $\xrightarrow{1} I$

□

The following lemma establishes a sort of consistency property of the type system, showing that a term M cannot be both strict and vacuous in a given variable. This will be central in the proof of disjointness of pattern complementation (Lemma 6.4).

LEMMA 3.7 EXCLUSIVITY. *It is not the case that both* $\Gamma_1; \Omega_1; (\Delta_1, x{:}C) \vdash M : A$ *and* $\Gamma_2; (\Omega_2, x{:}C); \Delta_2 \vdash M : A$.

PROOF. By induction on the structure of the derivation of $\Gamma_1; \Omega_1; (\Delta_1, x{:}C) \vdash M : A$, applying inversion on the derivation of $\Gamma_2; (\Omega_2, x{:}C); \Delta_2 \vdash M : A$ in each case. □

4. THE CANONICAL FORM THEOREM

In this section we establish the existence of canonical forms for the strict λ-calculus, i.e., β-normal η-long forms, which is crucial for our intended application. We prove this by Tait's method of *logical relations*; we essentially follow the account in [Pfenning 2001a], with a surprisingly little amount of generalization from simple to strict types, thanks to a simplified account of substitutions.

We start by presenting the inductive definition of *canonical* forms. It is realized by the two mutually recursive judgments depicted in Figure 4:

$$\begin{array}{ll} \Gamma; \Omega; \Delta \vdash M \downarrow A & M \text{ is atomic of type } A. \\ \Gamma; \Omega; \Delta \vdash M \Uparrow A & M \text{ is canonical of type } A. \end{array}$$

Informally, M is atomic (written $M \downarrow A$ for some A) if M consists of a variable applied to a sequence of arguments, where each of the arguments is canonical at appropriate type. A term M is canonical if M consists of a sequence of λ-abstractions followed by an atomic term of atomic type. We shall abbreviate judgments involving $\Uparrow$ and $\downarrow$ as $\Uparrow\!\!\downarrow$.

LEMMA 4.1 SOUNDNESS OF CANONICAL TERMS.
If $\Gamma; \Omega; \Delta \vdash M \Uparrow\!\!\downarrow A$, *then* $\Gamma; \Omega; \Delta \vdash M : A$.

PROOF. By induction on the structure of the derivation of $\Gamma; \Omega; \Delta \vdash M \Uparrow\!\!\downarrow A$. □

We describe an algorithm for conversion to canonical form in Figure 5. This algorithm is presented as a deductive system that can be used to construct a canonical form from an arbitrary well-typed term. Note that the algorithm does not need to keep track of occurrence constraints—they will be satisfied by construction (see Theorem 4.2). We write Ψ for a single context of distinct variable declarations whose status should be considered ambiguous since it is unnecessary to know whether they are unrestricted, irrelevant, or strict.

$$\begin{array}{ll} \Psi \vdash M \downarrow N : A & M \text{ has atomic form } N \text{ of type } A. \\ \Psi \vdash M \Uparrow N : A & M \text{ has canonical form } N \text{ at type } A. \end{array}$$

These utilize weak head reduction, which includes local reduction (β) and partial

$$\dfrac{c{:}A \in \Sigma}{\Gamma; \Omega; \cdot \vdash c \downarrow A}\,\texttt{cIdc}$$

$$\dfrac{}{(\Gamma, x{:}A); \Omega; \cdot \vdash x \downarrow A}\,\texttt{cId}^u \qquad \text{no } \texttt{cId}^0 \text{ rule} \qquad \dfrac{}{\Gamma; \Omega; x{:}A \vdash x \downarrow A}\,\texttt{cId}^1$$

$$\dfrac{\Gamma; \Omega; \Delta \vdash M \downarrow a}{\Gamma; \Omega; \Delta \vdash M \Uparrow a}\,\texttt{cAt}$$

$$\dfrac{(\Gamma, x{:}A); \Omega; \Delta \vdash M \Uparrow B}{\Gamma; \Omega; \Delta \vdash (\lambda x^u{:}A.\ M) \Uparrow A \xrightarrow{u} B}\,\texttt{c} \xrightarrow{u} I$$

$$\dfrac{\Gamma; \Omega; (\Delta, x{:}A) \vdash M \Uparrow B}{\Gamma; \Omega; \Delta \vdash (\lambda x^1{:}A.\ M) \Uparrow A \xrightarrow{1} B}\,\texttt{c} \xrightarrow{1} I$$

$$\dfrac{\Gamma; (\Omega, x{:}A); \Delta \vdash M \Uparrow B}{\Gamma; \Omega; \Delta \vdash (\lambda x^0{:}A.\ M) \Uparrow A \xrightarrow{0} B}\,\texttt{c} \xrightarrow{0} I$$

$$\dfrac{\Gamma; \Omega; \Delta \vdash M \downarrow A \xrightarrow{u} B \qquad (\Gamma, \Delta); \Omega; \cdot \vdash N \Uparrow A}{\Gamma; \Omega; \Delta \vdash M\ N^u \downarrow B}\,\texttt{c} \xrightarrow{u} E$$

$$\dfrac{\Gamma; \Omega; \Delta \vdash M \downarrow A \xrightarrow{0} B \quad (\Gamma, \Omega, \Delta); \cdot; \cdot \vdash N \Uparrow A}{\Gamma; \Omega; \Delta \vdash M\ N^0 \downarrow B}\,\texttt{c}\xrightarrow{0}E$$

$$\dfrac{(\Gamma, \Delta_N); \Omega; \Delta_M \vdash M \downarrow A \xrightarrow{1} B \qquad (\Gamma, \Delta_M); \Omega; \Delta_N \vdash N \Uparrow A}{\Gamma; \Omega; (\Delta_M, \Delta_N) \vdash M\ N^1 \downarrow B}\,\texttt{c}\xrightarrow{1}E$$

Fig. 4. Canonical forms: $\Gamma; \Omega; \Delta \vdash M \Uparrow\!\!\downarrow A$

$$\dfrac{c{:}A \in \Sigma}{\Psi \vdash c \downarrow c : A}\,\texttt{tcIdc} \qquad \dfrac{x{:}A \in \Psi}{\Psi \vdash x \downarrow x : A}\,\texttt{tcIdvar}$$

$$\dfrac{M \xrightarrow{\text{whr}} M' \qquad \Psi \vdash M' \Uparrow M'' : a}{\Psi \vdash M \Uparrow M'' : a}\,\texttt{tc} \xrightarrow{\text{whr}} \qquad \dfrac{\Psi \vdash M \downarrow N : a}{\Psi \vdash M \Uparrow N : a}\,\texttt{tcAtm}$$

$$\dfrac{\Psi, x{:}A \vdash M\ x^k \Uparrow N : B}{\Psi \vdash M \Uparrow (\lambda x^x{:}A.\ N) : A \xrightarrow{k} B}\,\texttt{tc} \xrightarrow{k} I \qquad \dfrac{\Psi \vdash M \downarrow P : A \xrightarrow{k} B \qquad \Psi \vdash N \Uparrow Q : A}{\Psi \vdash M\ N^k \downarrow P\ Q^k : B}\,\texttt{tc} \xrightarrow{k} E$$

Fig. 5. Conversion to canonical form: $\Psi \vdash M \Uparrow\!\!\downarrow N : A$

congruence (ν):

$$\dfrac{}{(\lambda x^k{:}A.\ M)\ N^k \xrightarrow{\text{whr}} [N/x]M}\,\beta^k \qquad \dfrac{M \xrightarrow{\text{whr}} Q}{M\ N^k \xrightarrow{\text{whr}} Q\ N^k}\,\nu^k$$

Operationally, we assume that M is given and we construct an N such that $M \xrightarrow{\text{whr}} N$ or fail. The judgments for conversion to canonical form can be interpreted as an algorithm in the following manner:

$$\begin{array}{ll}\Psi \vdash M \downarrow N : A & \text{Given } \Psi \text{ and } M \text{ construct } N \text{ and } A \\ \Psi \vdash M \Uparrow N : A & \text{Given } \Psi,\ M, \text{ and } A \text{ construct } N\end{array}$$

The main theorem of this section states that if $\Gamma; \Omega; \Delta \vdash M : A$ and $\Psi = (\Gamma, \Omega, \Delta)$ then the two judgments above will always succeed to construct an N and A, or N, respectively.

THEOREM 4.2 CONVERSION YIELDS CANONICAL TERMS.
If $(\Gamma, \Omega, \Delta) \vdash M \Updownarrow N : A$ *and* $\Gamma; \Omega; \Delta \vdash M : A$, *then* $\Gamma; \Omega; \Delta \vdash N \Updownarrow A$.

PROOF. By induction on the structure of the derivation of $(\Gamma, \Omega, \Delta) \vdash M \Updownarrow N : A$ and inversion on the given typing derivation in each case. □

In the construction of logical relations we will need a notion of context extension, $\Psi' \geq \Psi$ (Ψ' extends Ψ with zero or more declarations). It is clear that conversion to canonical form is not affected by weakening. We omit the formal statement of this property.

We can now introduce a unary *Kripke-logical relation*, in complete analogy with the usual definition for the simply-typed λ-calculus. At base type we postulate the property we are trying to show, namely existence of canonical forms. At higher type we reduce the property to lower types by quantifying over all possible elimination forms.

Definition 4.3 *Valid Terms.*

(1) $\Psi \vdash M \in [\![a]\!]$ iff $\Psi \vdash M \Uparrow N : a$, for some N.

(2) $\Psi \vdash M \in [\![A \xrightarrow{k} B]\!]$ iff for every $\Psi' \geq \Psi$ and every N, if $\Psi' \vdash N \in [\![A]\!]$, then $\Psi' \vdash M\ N^k \in [\![B]\!]$.

We say a term M is *valid* if $\Psi \vdash M \in [\![A]\!]$ for appropriate Ψ and A.

First we show that all valid terms have canonical forms. We prove at the same time that atomic terms are valid, both by induction on the structure of their types.

LEMMA 4.4 VALID TERMS HAVE CANONICAL FORMS.

(1) *If* $\Psi \vdash M \in [\![A]\!]$, *then* $\Psi \vdash M \Uparrow N : A$.

(2) *If* $\Psi \vdash M \downarrow N : A$, *then* $\Psi \vdash M \in [\![A]\!]$.

PROOF. By induction on A.

Case. $A = a$. Immediate from the definition of $[\![a]\!]$.

Case. $A = A_1 \xrightarrow{k} A_2$.

$$\begin{array}{lll}(1) & \Psi \vdash M \in [\![A_1 \xrightarrow{k} A_2]\!] & \text{Assumption} \\ & \Psi, x{:}A_1 \geq \Psi & \text{By definition of } \geq \\ & \Psi, x{:}A_1 \vdash x \downarrow x : A_1 & \text{By rule } \texttt{tcIdvar} \\ & \Psi, x{:}A_1 \vdash x \in [\![A_1]\!] & \text{By i.h. (2)} \\ & \Psi, x{:}A_1 \vdash M\ x^k \in [\![A_2]\!] & \text{By definition of } [\![\cdot]\!] \\ & \Psi, x{:}A_1 \vdash M\ x^k \Uparrow N : A_2 & \text{By i.h. (1)} \\ & \Psi \vdash M \Uparrow \lambda x^k{:}A_1.\ N : A_1 \xrightarrow{k} A_2 & \text{By rule } \texttt{tc} \xrightarrow{k} I\end{array}$$

(2) $\Psi \vdash M \downarrow M' : A_1 \xrightarrow{k} A_2$	Assumption
$\Psi' \geq \Psi$ and $\Psi' \vdash N \in [\![A_1]\!]$ for arbitrary Ψ' and N	New assumption
$\Psi' \vdash N \Uparrow N' : A_1$	By i.h. (1)
$\Psi' \vdash M \downarrow M' : A_1 \xrightarrow{k} A_2$	By weakening
$\Psi' \vdash M\ N^k \downarrow M'\ N'^k : A_2$	By rule $\mathtt{tc}\xrightarrow{k}E$
$\Psi' \vdash M\ N^k \in [\![A_2]\!]$	By i.h. (2)
$\Psi \vdash M \in [\![A_1 \xrightarrow{k} A_2]\!]$	By definition of $[\![\cdot]\!]$

□

The second major part states that every well-typed term is valid. For this we need closure of validity under head expansion.

Lemma 4.5 Closure under Head Expansion.
If $\Psi \vdash M' \in [\![A]\!]$ and $M \xrightarrow{whr} M'$, then $\Psi \vdash M \in [\![A]\!]$.

Proof. By induction on A:

Case. $A = a$. immediate by definition and rule $\mathtt{tc} \xrightarrow{\text{whr}}$.
Case. $A = A_1 \xrightarrow{k} A_2$.

$\Psi \vdash M' \in [\![A_1 \xrightarrow{k} A_2]\!]$	Assumption
$\Psi' \vdash N \in [\![A_1]\!]$ for arbitrary $\Psi' \geq \Psi$ and N	New assumption
$\Psi' \vdash M'\ N^k \in [\![A_2]\!]$	By definition of $[\![\cdot]\!]$
$M\ N^k \xrightarrow{\text{whr}} M'\ N^k$	By rule ν
$\Psi' \vdash M\ N^k \in [\![A_2]\!]$	By i.h. on A_2
$\Psi \vdash M \in [\![A_1 \xrightarrow{k} A_2]\!]$	By definition of $[\![\cdot]\!]$

□

Due to the need to β-reduce during conversion to canonical form, we need to introduce *substitutions*. We will not require substitutions to be well-typed, but they have to be valid in the sense that all substitution terms should be valid.

$$\textit{Substitutions}\ \theta\ ::= \epsilon \mid \theta, M/x$$

For $\theta = \theta', M/x$, we say that x is *defined* in θ and we write $\theta(x) = M$. We require all variables defined in a substitution to be distinct: we use $\mathrm{dom}(\theta)$ for the set of variables defined in θ. Furthermore, the co-domain of θ are the variables occurring in the substituting terms.

Next, we define the *application* of a substitution θ to a term M, denoted $[\theta]M$. We limit application of substitutions to objects whose free variables are in the domain of θ.

$$\begin{aligned} [\theta]c &= c \\ [\theta]x &= \theta(x) \\ [\theta](M\ N^k) &= ([\theta]M)\ ([\theta]N)^k \\ [\theta](\lambda x^k{:}A.\ M) &= \lambda x^k{:}A.\ [\theta, x/x]M \end{aligned}$$

In the last case we assume that x does not already occur in the domain or co-domain of θ. This can always be achieved by renaming of the bound variable.

We will also need to mediate between single substitutions stemming from β-reduction and simultaneous substitutions. We define how to *compose* a single substitution from a β-reduction with simultaneous substitutions, written as $[N/x]\theta$.

$$\begin{aligned} [N/x](\epsilon) &= \epsilon \\ [N/x](\theta, M/y) &= [N/x](\theta), ([N/x]M)/y \end{aligned}$$

Note that $[N/x]([\theta, x/x]M) = [\theta, N/x]M$ if x does not occur in the co-domain of θ. For a context $\Psi = x_1{:}A_1, \ldots, x_n{:}A_n$, we introduce the *identity* substitution on Ψ as $\mathrm{id}_\Psi = x_1/x_1, \ldots, x_n/x_n$. Clearly, $\mathrm{id}_\Psi M = M$ if the free variables of M are contained in Ψ.

We extend the notion of validity to substitutions as already indicated above: a substitution θ is valid for context Ψ if for every binding M/x such that $x{:}A$ is in Ψ we have M is in $[\![A]\!]$.

Definition 4.6 *Valid Substitutions.*

(1) $\Phi \vdash \theta \in [\![\cdot]\!]$ iff $\theta = \epsilon$.

(2) $\Phi \vdash \theta \in [\![\Psi', x{:}A]\!]$ iff $\theta = \theta', M/x$ such that $\Phi \vdash M \in [\![A]\!]$ and $\Phi \vdash \theta' \in [\![\Psi']\!]$.

We remark that contexts are not ordered, hence, for $\Psi = (\Gamma, \Omega, \Delta)$ we will identify, for example, $[\![\Psi, x{:}A]\!]$ with $[\![(\Gamma, x{:}A, \Omega, \Delta)]\!]$. Clearly, this view is legitimate in terms of the above definition, since validity of a substitution simply reduces to validity of the terms in it. It is easy to see that validity, both for terms and for substitutions, satisfies weakening. We omit the formal statement and proof of this property.

The next lemma is critical. It generalizes the statement that well-typed terms are valid by allowing for a valid substitution to be applied. This is necessary in order to proceed with the proof in the case of any of the three λ-abstractions.

LEMMA 4.7 WELL-TYPED TERMS ARE VALID.
If $\Gamma; \Omega; \Delta \vdash M : A$, then for every Ψ such that $\Psi \vdash \theta \in [\![(\Gamma, \Omega, \Delta)]\!]$ we have $\Psi \vdash [\theta]M \in [\![A]\!]$.

PROOF. By induction on the typing derivation $\mathcal{D}$ of $\Gamma; \Omega; \Delta \vdash M : A$.

Case.

$$\mathcal{D} = \frac{}{(\Gamma, x{:}A); \Omega; \cdot \vdash x : A}\ \mathtt{Id}^u$$

$\Psi \vdash \theta \in [\![(\Gamma, x{:}A, \Omega)]\!]$	Assumption
$\Psi \vdash \theta(x) \in [\![A]\!]$	By definition of $[\![\cdot]\!]$
$\Psi \vdash [\theta]x \in [\![A]\!]$	By definition of substitution

Case. $\mathcal{D}$ ends in $\mathtt{Id}^1$. As in the previous case.

Case. $\mathcal{D}$ ends in $\mathtt{Con}$. Immediate by Lemma 4.4(2) and definition of substitution.
Case.

$$\mathcal{D} = \frac{(\Gamma, x{:}A); \Omega; \Delta \vdash M : B}{\Gamma; \Omega; \Delta \vdash \lambda x^u{:}A.\ M : A \xrightarrow{u} B} \xrightarrow{u} I$$

$(\Gamma, x{:}A); \Omega; \Delta \vdash M : B$	Subderivation
$\Psi \vdash \theta \in [\![(\Gamma, \Omega, \Delta)]\!]$	Assumption
$\Psi' \vdash N \in [\![A]\!]$ for arbitrary $\Psi' \geq \Psi$ and N	New assumption
$\Psi' \vdash (\theta, N/x) \in [\![(\Gamma, x{:}A, \Omega, \Delta)]\!]$	By definition of $[\![\cdot]\!]$ and weakening
$\Psi' \vdash [\theta, N/x]M \in [\![B]\!]$	By i.h.
$\Psi' \vdash [N/x]([\theta, x/x]M) \in [\![B]\!]$	By property of substitution
$\Psi' \vdash (\lambda x^u{:}A.\ [\theta, x/x]M)N^u \in [\![B]\!]$	By Lemma 4.5
$\Psi' \vdash ([\theta](\lambda x^u{:}A.\ M))N^u \in [\![B]\!]$	By definition of substitution
$\Psi \vdash [\theta](\lambda x^u{:}A.\ M) \in [\![A \xrightarrow{u} B]\!]$	By definition of $[\![A \xrightarrow{u} B]\!]$

Cases. $\mathcal{D}$ ends in $\xrightarrow{0} I$ or $\xrightarrow{1} I$. Analogous to previous case.
Case.

$$\mathcal{D} = \frac{\Gamma; \Omega; \Delta \vdash M : A \xrightarrow{u} B \qquad (\Gamma, \Delta); \Omega; \cdot \vdash N : A}{\Gamma; \Omega; \Delta \vdash M\ N^u : B} \xrightarrow{u} E$$

$\Psi \vdash \theta \in [\![(\Gamma, \Omega, \Delta)]\!]$	Assumption
$\Gamma; \Omega; \Delta \vdash M : A \xrightarrow{u} B$	Subderivation
$\Psi \vdash [\theta]M \in [\![A \xrightarrow{u} B]\!]$	By i.h.
$(\Gamma, \Delta); \Omega; \cdot \vdash N : A$	Subderivation
$\Psi \vdash [\theta]N \in [\![A]\!]$	By i.h.
$\Psi \geq \Psi$	By definition of $\geq$
$\Psi \vdash ([\theta]M)([\theta]N)^u \in [\![B]\!]$	By definition of $[\![\cdot]\!]$
$\Psi \vdash [\theta](M\ N)^u \in [\![B]\!]$	By definition of substitution

Cases. $\mathcal{D}$ ends in $\xrightarrow{0} E$ or $\xrightarrow{1} E$. Analogous to the previous case.

□

From this central lemma, the canonical form theorem follows by noting that the identity substitution is valid.

Lemma 4.8 Validity of Identity. $\Psi \vdash id_\Psi \in [\![\Psi]\!]$.

Proof. By a straightforward induction on Ψ using Lemma 4.4(2). □

Theorem 4.9 Canonical Forms.
If $\Gamma; \Omega; \Delta \vdash M : A$*, then there exists an* N *such that* $(\Gamma, \Omega, \Delta) \vdash M \Uparrow N : A$ *and* $\Gamma; \Omega; \Delta \vdash N \Uparrow A$.

Proof. Direct from prior lemmas.

$\Gamma; \Omega; \Delta \vdash M : A$	Assumption
$(\Gamma, \Omega, \Delta) \vdash id_{(\Gamma,\Omega,\Delta)} \in [\![(\Gamma, \Omega, \Delta)]\!]$	By Lemma 4.8
$(\Gamma, \Omega, \Delta) \vdash [id_{(\Gamma,\Omega,\Delta)}]M \in [\![A]\!]$	By Lemma 4.7
$(\Gamma, \Omega, \Delta) \vdash M \in [\![A]\!]$	By identity substitution
$(\Gamma, \Omega, \Delta) \vdash M \Uparrow N : A$ for some N	By Lemma 4.4(1)
$\Gamma; \Omega; \Delta \vdash N \Uparrow A$	By Theorem 4.2

□

We close this section with some remarks on related work on strictness. Church original definition of the set Λ_I of (untyped) λ-terms [Church 1941] has this clause for abstraction:

$$\text{If } M \in \Lambda_I \text{ and } x \in FV(M), \text{ then } \lambda x.\, M \in \Lambda_I.$$

Therefore, in this language there cannot be any vacuous abstractions. The combinatorial counterpart of this calculus excludes $\mathbf{K}$ and consists of $\mathbf{I}, \mathbf{W}, \mathbf{B}, \mathbf{C}$. Those are the axioms of what Church called *weak implicational logic* [Church 1951], i.e., identity, contraction, prefixing and permutation. This establishes the link with an enterprise born from a very different origin, namely the relevance logic project [Anderson and Belnap 1975], which emerged in fact in the early sixties out of Anderson and Belnap's dissatisfaction with the so-called "*paradoxes of implication*", let it be material, intuitionistic, or strict (in the modal sense of Lewis and Langford).

Following Girard's and Belnap's suggestion [Belnap 1993], we will *not* refer to our calculus as *relevant*, but as *strict* logic, as the former may also satisfy other principles such as distributivity of implication over conjunction.

On an unrelated front, starting with Mycroft's seminal paper [Mycroft 1980], compile-time analysis of functional programs concentrated on strictness analysis in order to get the best out of call-by-value and call-by-need evaluation; first in terms of abstract interpretation, later by using non-standard types to represent these "intensional" properties of functions (see [Jensen 1991] for a comparison of these two techniques). However, earlier work such as [Tsung-Min and Mishra 1989] used non-standard primitive type to distinguish strict or non-strict terms, closed only under unrestricted function space. In the setting of functional programming, various different notions of strictness emerged. However, the absence of recursion and effects in our setting admits fewer distinctions.

Wright [1992] seems to be the first to have extended the Curry-Howard isomorphism to the implicational fragment of relevance logic and explicitly connected the two areas, although both [Belnap 1974] and [Helmann 1977] had previously recognized the link between strictness and relevance.

Baker-Finch [1993] presents a type assignment system that makes available strict, invariant and intuitionistic types. It is biased towards enforcing strictness information, which ultimately leads to a different expressive power from our calculus. There is only one context, where variables carry their occurrence status as a label. There is one identity rule, the strict one, so that e.g. $\lambda x.\, x : A \xrightarrow{u} A$ is not derivable, as it can be given the more stringent type $A \xrightarrow{1} A$. Let us consider the elimination rules for strict and irrelevant functions.

$$\frac{\Gamma \vdash M : A \xrightarrow{1} B \qquad \Gamma' \vdash N : A'}{\Gamma, \Gamma' \vdash M\ N : B}\ app \xrightarrow{1}$$

$$\frac{\Gamma \vdash M : A \xrightarrow{0} B \qquad \Gamma' \vdash N : A'}{\Gamma, \Gamma'[1 := 0] \vdash M\ N : B}\ app \xrightarrow{0}$$

A side condition $A' \leq A$ enforces the information ordering, so that for example $A' \xrightarrow{0} B \leq A \xrightarrow{u} B'$, provided that $A \leq A', B \leq B'$. This allows us to infer by strict application $\Gamma, \Gamma' \vdash M\ N : C$ from $\Gamma \vdash M : (A \xrightarrow{u} B) \xrightarrow{1} C$ and $\Gamma' \vdash N : A \xrightarrow{0} B$. The latter is instead forbidden in our system by the labeled reduction rules. The rationale on the relabeling operation in the rule *app* $\xrightarrow{0}$ is that A is not relevant to B, so all hypothesis should be deleted. Instead, in order to preserve every variable declaration, their strict label is changed into irrelevant. This would amount to moving the strict variables in the irrelevant context in our system. Note the difference with our rule, where the latter variables are moved in the unrestricted context. Moreover, having only one context, the author needs a strategy to deal with the same variable with different annotations; the solution is that while propagating premises top-down a binding $x^1{:}A$ supersedes $x^u{:}A$ which in turn supersedes $x^0{:}A$.

Wright [1996] introduces *Annotation Logic* as a general framework for resource-conscious logics. Its formulae have the form $A ::= X^k \mid A \xrightarrow{k} B$ for any annotation k and there are specific structural as well as annotation rules. The latter implement rules such as promotion or dereliction. By instantiation with different algebras of annotation, we get systems such as linear and strict logic as well as various other usage logics. An abstract normalization procedure is sketched, which however requires commutative conversions already in the purely implicational fragment.

In summary, none of the systems of strict function in the literature served our purpose, nor did any of the authors prove the existence of canonical forms that are critical for our application.

5. SIMPLE TERMS

Now that we have developed a calculus which is potentially strong enough to represent the complement of linear patterns, two questions naturally arise: how do we embed the original λ-calculus, and is the calculus now closed under complement? We require that our complement operator ought to satisfy the usual boolean rules for negation:

(1) (Exclusivity) It is not the case that some M is both a ground instance of N and of $\mathrm{Not}(N)$.
(2) (Exhaustivity) Every M is a ground instance of N or of $\mathrm{Not}(N)$.

Remember that when we refer to *ground instances* we mean instances without any existential variables. Parameters, on the other hand, can certainly occur.

Unfortunately, while the first property follows quite easily for a suitable algorithm, it turns out the second cannot be achieved for the full strict λ-calculus calculus as presented in the previous sections. The following counterexample is a pattern whose complement cannot be expressed within the language.

Example 5.1. Consider the signature $a{:}\mathrm{type}, b{:}a, c{:}a \xrightarrow{u} a$. Then in the context $x{:}a; \cdot; \cdot$ we have

$$\begin{aligned} \|E\ x^0\| &= \{b, c\ b^u, c\ (c\ b^u)^u, \ldots\} \\ \mathrm{Not}(\|E\ x^0\|) &= \{x, c\ x^u, c\ (c\ x^u)^u, \ldots\} \end{aligned}$$

It is easy to see that $\mathrm{Not}(\|E\ x^0\|)$ cannot be described by a finite set of patterns. The underlying problem is the undetermined status of the argument to $c{:}a \xrightarrow{u} a$ which means it can contain neither strict nor irrelevant variables while being allowed to contain unrestricted variables.

However, the main result of this section is that the complement algorithm presented in Definition 6.1 is sound and complete for the fragment which results from the natural embedding of the original simply-typed λ-calculus; this is sufficient for our intended applications. We will proceed in two phases. First we restrict ourselves to a class of terms (that we call *simple*) for which the crucial property of *tightening* (Lemma 5.5) can be established. Second we transform the complement problem so that each existential variable is applied to *all* parameters and bound variables in whose scope it appears. This improvement is mainly cosmetic and makes it easier to state and prove correctness for our algorithms.

Recall that we have introduced strictness to capture occurrence conditions on variables in canonical forms. This means that first-order constants (and by extension bound variables) should be considered *strict* functions of their argument, since these arguments will indeed occur in the canonical form. On the other hand, if we have a second order constant, we cannot restrict the argument function to be either strict or vacuous, since this would render our representation language too weak.

Example 5.2. Continuing Example 2.1, consider the representation of the **K** combinator:

$$\ulcorner \Lambda x.\ \Lambda y.\ x \urcorner \;=\; lam\ (\lambda x{:}exp.\ lam\ (\lambda y{:}exp.\ x))$$

Notice that the argument to the first occurrence of '*lam*' is a strict function, while the argument to the second occurrence is an invariant function. If we can give only one type to '*lam*' it must therefore be $(exp \overset{u}{\to} exp) \overset{1}{\to} exp$.

Generalizing this observation means that positive occurrence of function types are translated to strict functions, while the negative ones to undetermined functions. We can formalize this as an embedding of the simply-typed λ-calculus into a fragment of the strict calculus via two (overloaded) mutually recursive translations $()^-$ and $()^+$. First, the definition on types:

$$\begin{aligned} (A \to B)^+ &= A^- \overset{1}{\to} B^+ \\ (A \to B)^- &= A^+ \overset{u}{\to} B^- \\ a^- = a^+ &= a \end{aligned}$$

We extend it to atomic and canonical terms (including existential variables), signatures, and contexts; we therefore need the usual inductive definition of atomic and canonical terms in the simply-typed λ-calculus (see for example [Pfenning 2001a]), which can be obtained by dropping labels from the definition of canonical form in Figure 4. In addition, we allow well-typed applications $E_A\ x_1^{k_1} \dots x_n^{k_n}$ of base type as canonical terms. Recall that $x_1, \dots, x_n$ must be distinct bound variables or parameters. Note that the embedding $()^-$ is applied only to canonical terms, while $()^+$ is applied only to atomic terms.

$$\begin{aligned} (\lambda x{:}A.\ M)^- &= \lambda x^u{:}A^+.\ M^- \\ (E_A\ x_1 \dots x_n)^- &= F_{A^-}\ x_1^u \dots x_n^u \\ M^- &= M^+ \qquad\qquad \text{for } M \text{ of base type} \\ x^+ &= x \end{aligned}$$

$$
\begin{aligned}
c^+ &= c \\
(M\ N)^+ &= M^+\ (N^-)^1
\end{aligned}
$$

$$
\begin{aligned}
(\cdot)^+ &= \cdot \\
(\Gamma, x{:}A)^+ &= \Gamma^+, x{:}A^+ \\
(\Sigma, a{:}type)^+ &= \Sigma^+, a{:}type \\
(\Sigma, c{:}A)^+ &= \Sigma^+, c{:}A^+
\end{aligned}
$$

Example 5.3. Returning to Example 5.2:

$$(lam\ (\lambda x{:}exp.\ lam\ (\lambda y{:}exp.\ x)))^+ \ =\ lam\ (\lambda x^u{:}exp.\ lam\ (\lambda y^u{:}exp.\ x)^1)^1$$

The image of the embedding of the canonical forms of the simply-typed λ-calculus gives rise to the following fragment, where we allow existential variables to have arguments with arbitrary labels.

$$\textit{Simple Terms}\ \ M\ ::=\ \lambda x^u{:}A^+.\ M \mid h\ M_1^1 \ldots M_n^1 \mid E_A\ x_1^{k_1} \ldots x_n^{k_n}$$

It is possible to generalize this language further to allow arbitrary abstractions as well, but this is beyond the scope of the present paper (see the comment in the Section 9).

THEOREM 5.4 CORRECTNESS OF $()^{\pm}$.

(1) *If* $\Gamma \vdash M \Uparrow A$, *then* $\Gamma^+; \cdot; \cdot \vdash M^- \Uparrow A^-$.

(2) *If* $\Gamma \vdash M \downarrow A$, *then* $\Gamma^+; \cdot; \cdot \vdash M^+ \downarrow A^+$.

PROOF. By mutual induction on the structure of the derivations of $\Gamma \vdash M \Uparrow A$ and $\Gamma \vdash M \downarrow A$. $\square$

From now on we may hide the $()^1$ decoration from strict application of constants in examples. Moreover, we will shorten judgment $\mathcal{J}$ on simple terms of the form $\Psi; \cdot; \cdot \vdash \mathcal{J}$ to $\Psi \vdash \mathcal{J}$.

We can now prove the crucial tightening lemma. It expresses the property that every simple term with no existential variable is either strict or vacuous in a given undetermined variable.

LEMMA 5.5 TIGHTENING. *Let M be a simple term of type A with no existential variables.*

(1) *If* $(\Gamma, x{:}C); \Omega; \Delta \vdash M \downarrow A$ *then*
either $\Gamma; \Omega; (\Delta, x{:}C) \vdash M \downarrow A$ *or* $\Gamma; (\Omega, x{:}C); \Delta \vdash M \downarrow A$.

(2) *If* $(\Gamma, x{:}C); \Omega; \Delta \vdash M \Uparrow A$ *then*
either $\Gamma\ \Omega; (\Delta, x{:}C) \vdash M \Uparrow A$ *or* $\Gamma; (\Omega, x{:}C); \Delta \vdash M \Uparrow A$.

PROOF. By mutual induction on $\mathcal{D}_1 :: (\Gamma, x{:}C); \Omega; \Delta \vdash M \downarrow A$ and $\mathcal{D}_2 :: (\Gamma, x{:}C); \Omega; \Delta \vdash M \Uparrow A$. We show only one case.

Case.

$$\mathcal{D}_1 = \frac{(\Gamma, x{:}C, \Delta_N); \Omega; \Delta_M \vdash M \downarrow A \xrightarrow{1} B \quad (\Gamma, x{:}C, \Delta_M); \Omega; \Delta_N \vdash N \Uparrow A}{(\Gamma, x{:}C); \Omega; (\Delta_M, \Delta_N) \vdash M\ N^1 \downarrow B}\ \mathsf{c}\xrightarrow{1}E$$

There are four sub-cases, stemming from the two possibilities each for the two subderivations.

$$
\begin{array}{lll}
(1) & (\Gamma, \Delta_N); \Omega; (\Delta_M, x{:}C) \vdash M \downarrow A \xrightarrow{1} B & \text{Subcase of i.h.} \\
 & (\Gamma, \Delta_M); \Omega; (\Delta_N, x{:}C) \vdash N \Uparrow A & \text{Subcase of i.h.} \\
 & (\Gamma, \Delta_M, x{:}C); \Omega; \Delta_N \vdash N \Uparrow A & \text{By Loosening}^1\ x \\
 & \Gamma; \Omega; (\Delta_M, x{:}C, \Delta_N) \vdash M\ N^1 \downarrow B & \text{By rule } \mathsf{c}{\xrightarrow{1}}E \\
(2) & (\Gamma, \Delta_N); (\Omega, x{:}C); \Delta_M \vdash M \downarrow A \xrightarrow{1} B & \text{Subcase of i.h.} \\
 & (\Gamma, \Delta_M); (\Omega, x{:}C); \Delta_N \vdash N \Uparrow A & \text{Subcase of i.h.} \\
 & \Gamma; (\Omega, x{:}C); (\Delta_M, \Delta_N) \vdash M\ N^1 \downarrow B & \text{By rule } \mathsf{c}{\xrightarrow{1}}E \\
(3) & (\Gamma, \Delta_N); \Omega; (\Delta_M, x{:}C) \vdash M \downarrow A \xrightarrow{1} B & \text{Subcase of i.h.} \\
 & (\Gamma, \Delta_M); (\Omega, x{:}C); \Delta_N \vdash N \Uparrow A & \text{Subcase of i.h.} \\
 & (\Gamma, \Delta_M, x{:}C); \Omega; \Delta_N \vdash N \Uparrow A & \text{By Loosening}^0\ x \\
 & \Gamma; \Omega; (\Delta_M, x{:}C, \Delta_N) \vdash M\ N^1 \downarrow B & \text{By rule } \mathsf{c}{\xrightarrow{1}}E \\
(4) & \text{Symmetrical to (3).} &
\end{array}
$$

□

We remark that tightening fails to hold once we allow unrestricted function types in a negative position. For example, $(y{:}A \xrightarrow{u} B, x{:}A); \cdot; \cdot \vdash y\ x^u : B$ but both $y{:}A \xrightarrow{u} B; \cdot; x{:}A \not\vdash y\ x^u : B$ and $y{:}A \xrightarrow{u} B; x{:}A; \cdot \not\vdash y\ x^u : B$.

We also have the following related property.

Lemma 5.6 Irrelevance. *Let M be a simple term without existential variables.*

(1) If $\Gamma; (\Omega, x{:}C); \Delta \vdash M \Uparrow A$, then $\Gamma; \Omega; \Delta \vdash M \Uparrow A$.

(2) If $\Gamma; (\Omega, x{:}C); \Delta \vdash M \downarrow A$, then $\Gamma; \Omega; \Delta \vdash M \downarrow A$.

Proof. By mutual induction on the given derivations. □

Note that irrelevance holds for any strict canonical term, but it is false for terms containing redices. For example, for $c{:}B$ we have $\cdot; x{:}A; \cdot \vdash (\lambda y^0{:}A.\ c)\ x^0 : B$, but $\cdot; \cdot; \cdot \not\vdash (\lambda y^0{:}A.\ c)\ x^0 : B$.

For simple terms it is often more convenient to replace explicit reference to atomic forms by an n-ary version of $\mathsf{c}{\xrightarrow{1}}E$. This can easily be seen to cover all atomic forms for simple terms where the head h can be a variable x or constant c.

$$
\frac{\Psi \vdash h : A_1 \xrightarrow{1} \cdots \xrightarrow{1} A_n \xrightarrow{1} a \quad \Psi \vdash N_1 \Uparrow A_1 \quad \cdots \quad \Psi \vdash N_n \Uparrow A_n}{\Psi \vdash h\ N_1^1 \ldots N_n^1 \Uparrow a}\ \mathsf{c}{\xrightarrow{1}}E
$$

We can simplify the presentation of the algorithms for complement and later unification if we require any existential variable to be applied to every bound variable in its declaration context. This is possible for any simple linear pattern without changing the set of its ground instances. We just insert vacuous applications, which guarantee that the extra variables are not used.

In a slight abuse of notation we call the resulting patterns *fully applied*. This transformation is entirely straightforward and its correctness is easily established using Irrelevance (Lemma 5.6). We omit the formal details here, showing only an example.

$$\frac{\Gamma;\Omega;\Delta \vdash \Phi\ ok \qquad \Gamma;\Omega;\Delta \vdash M : a}{\Psi \vdash M \in \|E_A\ \Phi\| : a}\,\texttt{grFlx}$$

$$\frac{(\Psi, x{:}A) \vdash M \in \|N\| : B}{\Psi \vdash \lambda x^u{:}A.\ M \in \|\lambda x^u{:}A.\ N\| : A \xrightarrow{u} B}\,\texttt{grLam}$$

$$\frac{\Psi \vdash h : A_1 \xrightarrow{1} \cdots \xrightarrow{1} A_n \xrightarrow{1} a \qquad \Psi \vdash M_1 \in \|N_1\| : A_1 \ \cdots\ \Psi \vdash M_n \in \|N_n\| : A_n}{\Psi \vdash h\ M_1^1 \ldots M_n^1 \in \|h\ N_1^1 \ldots N_n^1\| : a}\,\texttt{grApp}$$

Fig. 6. Ground instance: $\Psi \vdash M \in \|N\| : A$

Example 5.7. Recall the simple pattern that encodes an object-level η-redex from Example 2.2,

$$lam\ (\lambda x^u{:}exp.\ app\ E\ x).$$

It is not fully applied, since E is not applied to x. This is crucial, since E is not allowed to depend on the bound variable x. In its fully applied form

$$lam\ (\lambda x^u{:}exp.\ app\ (E'\ x^0)\ x),$$

this occurrence condition is encoded by an irrelevant application of a fresh existential variable E' of type $exp \xrightarrow{0} exp$ to x. According to Lemma 5.6, this means that x cannot occur in the canonical form of $E'\ x^0$ for any instance of E'.

In the remainder of this paper we will assume that all existential variables are fully applied as defined above. We refer to a pattern $E\,x_1^{k_1} \ldots x_n^{k_n}$ as a *generalized variable*. Furthermore, we always sort the variables $x_1 \ldots x_n$ so that they come in some standard order; this simplifies the description of some of the algorithms on fully applied patterns. Following standard terminology we call atomic terms whose head is a bound variable or a parameter *rigid*, while terms whose head is an existential variable is called *flexible*.

Under these assumptions we can more formally specify the interpretation of terms with existential variables. We use Φ for sequences of distinct, labelled bound variables; if $x^k \in \Phi$, we set $\Phi(x) = k$. We say that $\Gamma;\Omega;\Delta \vdash \Phi\ ok$ if the following holds:

$$\begin{aligned}\Phi(x) = u \ &\Leftrightarrow\ x \in \mathrm{dom}(\Gamma)\\ \Phi(x) = 0 \ &\Leftrightarrow\ x \in \mathrm{dom}(\Omega)\\ \Phi(x) = 1 \ &\Leftrightarrow\ x \in \mathrm{dom}(\Delta)\end{aligned}$$

Note that Φ determines $\Gamma;\Omega;\Delta$ and vice versa whenever $\Gamma;\Omega;\Delta \vdash \Phi\ ok$.

Recall that every pattern can be seen as the intensional representation of the set of its instances with respected to a fixed signature Σ and a set of parameters declared in a context Ψ. The judgment in Figure 6, $\Psi \vdash M \in \|N\| : A$, formalizes the conditions for M canonical of type A to be a ground instance of a simple *linear* pattern N at type A.

Remark 5.8. Note that $\Psi \vdash M \in \|E_A\ \Phi\| : a$ means that M is indeed a ground instance of $E_A\ \Phi$. Conversely, if $\Phi = x_1^{k_1} \ldots x_n^{k_n}$ and $A = A_1 \xrightarrow{k_1^1} \cdots A_n \xrightarrow{k_n^n} a$ then

$$\frac{\mathrm{Not}_i(\Phi) \text{ defined}}{\Psi \vdash \mathrm{Not}(E\ \Phi) \Rightarrow Z\ \mathrm{Not}_i(\Phi) : a}\ \mathtt{NotFlx}^i$$

$$\frac{\Psi, x{:}A \vdash \mathrm{Not}(M) \Rightarrow N : B}{\Psi \vdash \mathrm{Not}(\lambda x^u{:}A.\ M) \Rightarrow \lambda x^u{:}A.\ N : A \xrightarrow{u} B}\ \mathtt{NotLam}$$

$$\frac{g \in \mathrm{dom}(\Sigma \cup \Psi), g : A_1 \xrightarrow{1} \ldots \xrightarrow{1} A_m \xrightarrow{1} a, h \neq g}{\Psi \vdash \mathrm{Not}(h\ M_1^1 \ldots M_n^1) \Rightarrow g\ (Z_1\ \Psi^u)^1 \ldots (Z_m\ \Psi^u)^1 : a}\ \mathtt{NotApp}_1$$

$$\frac{\Psi \vdash \mathrm{Not}(M_i) \Rightarrow N : A_i}{\Psi \vdash \mathrm{Not}(h\ M_1^1 \ldots M_n^1) \Rightarrow h\ (Z_1\ \Psi^u)^1 \ldots (Z_{i-1}\ \Psi^u)^1\ N^1\ (Z_{i+1}\ \Psi^u)^1 \ldots (Z_n\ \Psi^u)^1 : a}\ \mathtt{NotApp}_2^i$$

Fig. 7. Complement algorithm: $\Psi \vdash \mathrm{Not}(M) \Rightarrow N : A$

we set $E_A = \lambda x_1^{k_1}{:}A_1 \ldots \lambda x_n^{k_n}{:}A_n.\ M$

6. THE COMPLEMENT ALGORITHM

The idea of complementation for atomic terms and abstractions is quite simple and similar to the first-order case. For generalized variables we consider each argument in turn. If an argument variable is undetermined it does not contribute to the negation. If an argument variable is strict, then any term where this variable does not occur contributes to the negation. We therefore complement the corresponding label from 0 to 1 while all other arguments are undetermined. For vacuous argument variables we proceed dually.

In preparation for the rules, we observe that the complement operation on patterns behaves on labels like negation does on truth-values in Kleene's three-valued logic, in the sense of the following table:.

$$\mathrm{Not}(1) = 0 \qquad \mathrm{Not}(0) = 1 \qquad \mathrm{Not}(u) = u$$

We extend this definition to sequences of variables as they are used to codify occurrence constraints for existential variables.

$$\mathrm{Not}_i(x_1^{k_1} \ldots x_{i-1}^{k_{i-1}}\ x_i^d\ x_{i+1}^{k_{i+1}} \ldots x_n^{k_n}) \ = \ x_1^u \ldots x_{i-1}^u\ x_i^{\mathrm{Not}(d)}\ x_{i+1}^u \ldots x_n^u$$

Note that we require x_i to be determined ($d \in \{0, 1\}$) for Not_i to be defined, and that variables x_j for $j \neq i$ are all unrestricted on the right-hand side even though their status on the left-hand side varies.

Definition 6.1 *Higher-Order Pattern Complement.* For a linear simple pattern M such that $\Psi \vdash M \Uparrow A$, define $\Psi \vdash \mathrm{Not}(M) \Rightarrow N : A$ by the rules in Figure 7, where the Z's are fresh logic variables of appropriate type, $h \in \mathrm{dom}(\Sigma \cup \Psi)$ and $\Psi \vdash h : A_1 \xrightarrow{1} \ldots \xrightarrow{1} A_n \xrightarrow{1} a$. We write $Z\ \Psi^u$ as an abbreviation for $Z\ \Phi$ where $\Psi; \cdot; \cdot \vdash \Phi\ ok$.

Note that a given M may be related to several patterns N all of which belong to the complement of M. We therefore define $\Psi \vdash \mathrm{Not}(M) = \mathcal{N} : A$ if $\mathcal{N} = \{N \mid \Psi \vdash \mathrm{Not}(M) \Rightarrow N : A\}$.

We may drop the type information from the above judgment in examples and proofs; we will write $\Psi \vdash M \in \|\mathrm{Not}(N)\| : A$, when $\Psi \vdash \mathrm{Not}(N) = \mathcal{N}$ and $\Psi \vdash M \in \|\mathcal{N}\| : A$.

Example 6.2. Consider the following complement problems.

$$
\begin{aligned}
x{:}a, y{:}a \vdash \mathrm{Not}(E\ x^u\ y^1) &= \{F\ x^u\ y^0\} \\
x{:}a, y{:}a \vdash \mathrm{Not}(E\ x^0\ y^1) &= \{F\ x^1\ y^u, G\ x^u\ y^0\} \qquad (1)
\end{aligned}
$$

It is worthwhile to observe that the members of a complement set are not mutually disjoint, due to the indeterminacy of u. We can achieve exclusive patterns if we resolve this indeterminacy by considering for every x^u the two possibilities x^1, x^0. Thus, for example, the right-hand side of equation (1) can be rewritten as

$$\{F\ x^1 y^1, G\ x^1 y^0, H\ x^0 y^0\}.$$

It is clear that in the worst case scenario the number of patterns in a complement set is bounded by 2^n; hence the usefulness of this further step needs to be pragmatically determined.

We can now revisit the example of an η-redex in the untyped λ-calculus. To avoid too many indices on existential variables, we adopt a convention that the scope of existential variables is limited to each member of a complement set.

Example 6.3. Reconsider Example 2.2. Then we calculate:

$$
\begin{aligned}
\cdot \vdash\ & \mathrm{Not}(\mathit{lam}(\lambda x^u{:}\mathit{exp}.\ \mathit{app}\ (E\ x^0)\ x)) \\
&= \{\mathit{lam}(\lambda x^u{:}\mathit{exp}.\ \mathit{app}\ (Z\ x^1)\ (Z'\ x^u)), \\
&\quad \mathit{lam}(\lambda x^u{:}\mathit{exp}.\ \mathit{app}\ (Z\ x^u)\ (\mathit{app}\ (Z'\ x^u)\ (Z''\ x^u)), \\
&\quad \mathit{lam}(\lambda x^u{:}\mathit{exp}.\ \mathit{app}\ (Z\ x^u)\ (\mathit{lam}(\lambda y^u{:}\mathit{exp}.\ Z'\ x^u\ y^u)), \\
&\quad \mathit{lam}(\lambda x^u{:}\mathit{exp}.\ \mathit{lam}(\lambda y^u{:}\mathit{exp}.\ Z\ x^u\ y^u)), \\
&\quad \mathit{lam}(\lambda x^u{:}\mathit{exp}.\ x), \\
&\quad \mathit{app}\ Z\ Z'\}
\end{aligned}
$$

We now address the correctness of the complement algorithm with respect to the set-theoretic semantics. The proof obligation consists in proving that the former does behave as a complement operation on sets of patterns, that is, it satisfies disjointness and exhaustivity. Disjointness is the property that a set and its complement share no element; exhaustivity states that every element is in the set or its complement. Termination is obvious as the algorithm is syntax-directed and only finitely branching. We start with disjointness between a pattern and its complement.

LEMMA 6.4 DISJOINTNESS OF COMPLEMENTATION.
Let $\Psi \vdash N \Uparrow A$ be a simple linear pattern. Then for every Q such that $\Psi \vdash \mathrm{Not}(N) \Rightarrow Q : A$, it is not the case that both $\Psi \vdash M \in \|N\| : A$ and $\Psi \vdash M \in \|Q\| : A$.

PROOF. By induction on the structure of $\mathcal{D} :: \Psi \vdash \mathrm{Not}(N) \Rightarrow Q : A$.

Case. $\mathcal{D}$ ends in $\mathrm{Not}\texttt{Flx}^i$.

$\Psi \vdash M \in \|E\ \Phi\| : a$	Assumption
$\Psi \vdash M \in \|Z\ \mathrm{Not}_i(\Phi)\| : a$	Assumption
$\Phi(x_i) = 1$ or $\Phi(x_i) = 0$	Since $\mathrm{Not}_i(\Phi)$ defined

Subcase. $\Phi(x_i) = 1$

$\Gamma; \Omega; (\Delta, x_i{:}A) \vdash M : a$ — By inversion on $M \in \|E\ \Phi\|$

$(\Gamma, \Omega, \Delta); x_i{:}A; \cdot \vdash M : a$ — By inversion on $M \in \|Z\ \mathrm{Not}_i(\Phi)\|$

$\bot$ — By exclusivity (Lemma 3.7)

Subcase. $\Phi(x_i) = 0$ is symmetrical.

Case. $\mathcal{D}$ ends in $\mathtt{NotApp}_1$.

$\Psi \vdash M \in \|h\ N_1^1 \ldots N_n^1\| : a$ — Assumption

$\Psi \vdash M \in \|g\ (Z_1\ \Psi^u)^1 \ldots (Z_m\ \Psi^u)^1\| : a$ for $g \neq h$ — Assumption

$M = h \cdots$ — By inversion on `grApp`

$M = g \cdots$ — By inversion on `grApp`

$\bot$ — Since $g \neq h$

Case. $\mathcal{D}$ ends in $\mathtt{NotApp}_2^i$.

$\Psi \vdash M \in \|h\ N_1^1 \ldots N_n^1\| : a$ — Assumption

$\Psi \vdash M \in \|h\ (Z_1\ \Psi^u)^1 \ldots (Z_{i-1}\ \Psi^u)^1\ Q^1\ (Z_{i+1}\ \Psi^u)^1 \ldots (Z_n\ \Psi^u)^1\| : a$ and

$\Psi \vdash \mathrm{Not}(N_i) \Rightarrow Q : A_i$ — Assumption

$M = h\ M_1 \ldots M_n$ and

$\Psi \vdash M_i \in \|N_i\| : A_i$ — By inversion

$\Psi \vdash M_i \in \|Q\| : A_i$ — By inversion

$\bot$ — By i.h.

Case. $\mathcal{D}$ ends in $\mathtt{NotLam}$.

$\Psi \vdash \mathrm{Not}(\lambda x^u{:}A.\ N) \Rightarrow \lambda x^u{:}A.\ Q : A \xrightarrow{u} B$ — This case

$\Psi, x{:}A \vdash \mathrm{Not}(N) \Rightarrow Q : B$ — Subderivation

$\Psi \vdash \lambda x^u{:}A.\ M \in \|\lambda x^u{:}A.\ N\| : A \xrightarrow{u} B$ — Assumption

$\Psi \vdash \lambda x^u{:}A.\ M \in \|\lambda x^u{:}A.\ Q\| : A \xrightarrow{u} B$ — Assumption

$\Psi, x{:}A \vdash M \in \|N\| : B$ — By inversion

$\Psi, x{:}A \vdash M \in \|Q\| : B$ — By inversion

$\bot$ — By i.h.

□

Note that disjointness is based on exclusivity (Lemma 3.7), which holds for *any* strict term—it does not require simple terms. Next, we turn to the other direction. First a lemma concerning the special case of generalized variables.

Lemma 6.5 Exhaustivity for Flexible Patterns.
For every closed M such that $\Psi \vdash M \Uparrow a$, either $\Psi \vdash M \in \|E_A\ \Phi\| : a$ or $\Psi \vdash M \in \|Z\ \mathrm{Not}_i(\Phi)\| : a$ for some i.

Proof. Assume $\Psi \vdash M \Uparrow a$. Then by iterated applications of Lemma 5.5 there exist Ω and Δ such that $\Psi = \Omega, \Delta$ and $\cdot; \Omega; \Delta \vdash M \Uparrow a$.

Case. For every $x \in \mathrm{dom}(\Omega)$ we have $\Phi(x) \in \{0, u\}$ and for every $x \in \mathrm{dom}(\Delta)$ we have $\Phi(x) \in \{1, u\}$. Then $\Psi \vdash M \in \|E\ \Phi\|$.

Case. For some $x_i \in \mathrm{dom}(\Omega)$ we have $\Phi(x_i) = 1$.
Then $\Psi \vdash M \in \|Z\ x_1^u \ldots x_{i-1}^u\ x_i^1\ x_{i+1}^u \ldots x_n^u\|$ and therefore $\Psi \vdash M \in \|Z\ \mathrm{Not}_i(\Phi)\|$.

Case. For some $x_i \in \mathrm{dom}(\Delta)$ we have $\Phi(x_i) = 0$.
Then $\Psi \vdash M \in \|Z\ x_1^u \ldots x_{i-1}^u\ x_i^0\ x_{i+1}^u \ldots x_n^u\|$ and therefore $\Psi \vdash M \in \|Z\ \mathrm{Not}_i(\Phi)\|$.

□

We are now ready to prove exhaustivity of complementation.

LEMMA 6.6 EXHAUSTIVITY OF COMPLEMENTATION.
Assume $\Psi \vdash N \Uparrow A$ is a simple linear pattern. Then for every closed M such that $\Psi \vdash M \Uparrow A$, either $\Psi \vdash M \in \|N\| : A$ or there is a Q such that $\Psi \vdash \mathrm{Not}(N) \Rightarrow Q : A$ and $\Psi \vdash M \in \|Q\| : A$.

PROOF. By induction on the structure of $\mathcal{D} :: \Psi \vdash N \Uparrow A$.

Case. $\mathcal{D}$ ends in `cPat`. Then the claim follows immediately by Lemma 6.5.

Case. $\mathcal{D}$ ends in $\mathtt{c} \overset{u}{\to} I$. The i.h. yields the two sub-cases.

Subcase. $\Psi, x{:}A \vdash M \in \|N\| : B$.

$\Psi \vdash \lambda x^u{:}A.\ M \in \|\lambda x^u{:}A.\ N\| : A \overset{u}{\to} B$ By rule `grLam`

Subcase. $\Psi, x{:}A \vdash \mathrm{Not}(N) \Rightarrow Q : B$ and $\Psi, x{:}A \vdash M \in \|Q\| : B$ for some Q.

$\Psi \vdash \mathrm{Not}(\lambda x^u{:}A.\ N) \Rightarrow \lambda x^u{:}A.\ Q : A \overset{u}{\to} B$ By rule `NotLam`

$\Psi \vdash \lambda x^u{:}A.\ M \in \|\lambda x^u{:}A.\ Q\| : A \overset{u}{\to} B$ By rule `grLam`

Case.

$$\mathcal{D} = \cfrac{\Psi \vdash h : A_1 \overset{1}{\to} \cdots \overset{1}{\to} A_n \overset{1}{\to} a \quad \Psi \vdash N_1 \Uparrow A_1 \quad \cdots \quad \Psi \vdash N_n \Uparrow A_n}{\Psi \vdash h\ N_1^1 \ldots N_n^1 \Uparrow a}\ \mathtt{c}\overset{1}{\to}E$$

First, assume $M = g\ M_1^1 \ldots M_m^1$, for $g \in \mathrm{dom}(\Sigma \cup \Psi)$, $h \neq g$. Then

$\Psi \vdash \mathrm{Not}(h\ N_1^1 \ldots N_n^1) \Rightarrow g\ (Z_1\ \Psi^u)^1 \ldots (Z_m\ \Psi^u)^1 : a$ By rule $\mathtt{NotApp}_1$

$\Psi \vdash M_i \in \|Z_i\ \Psi^u\| : A_i$ for all $1 \leq i \leq m$ By rule `grFlx`

$\Psi \vdash g\ M_1^1 \ldots M_m^1 \in \|g\ (Z_1\ \Psi^u)^1 \ldots (Z_n\ \Psi^u)^1\| : a$ By rule `grApp`

Otherwise, assume $M = h\ M_1^1 \ldots M_n^1$. Again, the i.h. yields two sub-cases.

Subcase. $\Psi \vdash M_i \in \|N_i\| : A_i$, for all $1 \leq i \leq n$.

$\Psi \vdash h\ M_1^1 \ldots M_n^1 \in \|h\ N_1^1 \ldots N_n^1\| : a$ By rule `grApp`

Subcase. $\Psi \vdash \mathrm{Not}(N_i) \Rightarrow Q : A_i$ and $\Psi \vdash M_i \in \|Q\| : A_i$, for some Q.

$\Psi \vdash M_j \in \|Z_j\ \Psi^u\| : A_j$ for all $j \neq i$, $1 \leq j \leq n$ By rule `grFlx`

$\Psi \vdash \mathrm{Not}(h\ M_1^1 \ldots M_n^1)$
$\quad \Rightarrow h\ (Z_1\ \Psi^u)^1 \ldots (Z_{i-1}\ \Psi^u)^1\ Q^1\ (Z_{i+1}\ \Psi^u)^1 \ldots (Z_n\ \Psi^u)^1 : a$ By rule $\mathtt{NotApp}_2^i$

$\Psi \vdash h\ M_1^1 \ldots M_n^1 \in \|h\ (Z_1\ \Psi^u)^1 \ldots (Z_{i-1}\ \Psi^u)^1\ Q^1\ (Z_{i+1}\ \Psi^u)^1 \ldots (Z_m\ \Psi^u)^1\| : a$
By rule `grApp`.

□

The correctness of the algorithm for pattern complement follows directly from the preceding two lemmas.

THEOREM 6.7 CORRECTNESS OF PATTERN COMPLEMENT.
Assume N is a simple linear pattern such that $\Psi \vdash N : A$. Then for every closed M with $\Psi \vdash M \Uparrow A$, $\Psi \vdash M \in \|\mathrm{Not}(N)\| : A$ iff $\Psi \not\vdash M \in \|N\|$.

It is easy to see that simple linear patterns are closed under complementation.

THEOREM 6.8 CLOSURE UNDER COMPLEMENTATION. *Assume M is a simple linear pattern with $\Psi \vdash M \Uparrow A$. Then $\Psi \vdash \mathrm{Not}(M) \Rightarrow N : A$ implies N is a simple linear pattern and $\Psi \vdash N \Uparrow A$.*

PROOF. By induction on the structure of the derivation of $\Psi \vdash \mathrm{Not}(M) \Rightarrow N : A$. □

7. UNIFICATION OF SIMPLE PATTERNS

As we observed earlier, we can solve a relative complement problem by pairing complementation with intersection. We therefore address the task of giving an algorithm for unification of linear simple patterns. We start by determining when two labels are compatible:

$$\begin{aligned} 1 \cap 1 = u \cap 1 = 1 \cap u &= 1 \\ 0 \cap 0 = u \cap 0 = 0 \cap u &= 0 \\ u \cap u &= u \end{aligned}$$

Recall that Φ is a list of labelled bound variables. We call Φ_1 and Φ_2 *compatible* if they contain the same variables in the same order, but with possibly different labels. We can extend the intersection operations to compatible lists.

$$\begin{aligned} \cdot \cap \cdot \;&=\; \cdot \\ (\Phi, x^k) \cap (\Phi', x^{k'}) \;&=\; (\Phi \cap \Phi', x^{k \cap k'}) \quad \text{if } k \cap k' \text{ is defined.} \end{aligned}$$

For contexts Γ_1 and Γ_2 that may have variable declarations in common, we write $\Gamma_1 \cap \Gamma_2$ and $\Gamma_1 \cup \Gamma_2$ for set-theoretic union and intersection. In both cases we assume that a variable x declared in both Γ_1 and Γ_2 must be assigned the same type in both contexts.

Remark 7.1. Assume Φ_1 and Φ_2 are compatible and $\Phi_1 \cap \Phi_2$ is defined. Then $\Gamma_1; \Omega_1; \Delta_1 \vdash \Phi_1$ *ok* and $\Gamma_2; \Omega_2; \Delta_2 \vdash \Phi_2$ *ok* implies that $\Delta_1 \cap \Omega_2 = \Delta_2 \cap \Omega_1 = \emptyset$. Moreover, $(\Gamma_1 \cap \Gamma_2); (\Omega_1 \cup \Omega_2); (\Delta_1 \cup \Delta_2) \vdash (\Phi_1 \cap \Phi_2)$ *ok*. From that it follows that $\Psi \vdash M \in \|E_A\ (\Phi_1 \cap \Phi_2)\| : a$ iff $(\Gamma_1 \cap \Gamma_2); (\Omega_1 \cup \Omega_2); (\Delta_1 \cup \Delta_2) \vdash M : a$.

Definition 7.2 *Higher-Order Pattern Intersection.* Assume M and N are linear simple patterns without shared existential variables such that $\Psi \vdash M \Uparrow A$ and $\Psi \vdash N \Uparrow A$. We define $\Psi \vdash M \cap N \Rightarrow Q : A$ by the rules in Figure 8, where the H's are fresh variables of appropriate type. We omit two rules, $\cap\mathtt{RF}^c$ and $\cap\mathtt{RF}^y$, that are symmetric to $\cap\mathtt{FR}^c$ and $\cap\mathtt{FR}^y$.

The rules $\cap\mathtt{FR}^c$ and $\cap\mathtt{RF}^c$ have the following proviso: for all $1 \le i \le n$, $\mathrm{dom}(\Phi_i) = \mathrm{dom}(\Phi)$ and

$$\begin{aligned} \forall x. \Phi(x) = 0 \supset \forall i.\ \Phi_i(x) = 0 \\ \forall x. \Phi(x) = u \supset \forall i.\ \Phi_i(x) = u \\ \forall x. \Phi(x) = 1 \supset \exists i.\ \Phi_i(x) = 1 \wedge \forall j.\ j \neq i \supset \Phi_j(x) = u \end{aligned}$$

The rules $\cap\mathtt{FR}^y$ and $\cap RF^y$ are subject to the proviso:

$$\begin{aligned} \forall x. \Phi(x) = 0 \supset \forall i.\ \Phi_i(x) = 0 \\ \forall x. \Phi(x) = u \supset \forall i.\ \Phi_i(x) = u \end{aligned}$$

$$\frac{}{\Psi \vdash (E_1\ \Phi_1) \cap (E_2\ \Phi_2) \Rightarrow H\ (\Phi_1 \cap \Phi_2) : a}\ \cap\mathsf{FF}$$

no rule for flex/flex same

$$\frac{c \in \mathrm{dom}(\Sigma) \quad \Psi \vdash (H_1\ \Phi_1) \cap M_1 \Rightarrow N_1 : A_1 \cdots \Psi \vdash (H_n\ \Phi_n) \cap M_n \Rightarrow N_n : A_n}{\Psi \vdash (E\ \Phi) \cap (c\ M_1^1 \ldots M_n^1) \Rightarrow c\ N_1^1 \ldots N_n^1 : a}\ \cap\mathsf{FR}^c$$

$$\frac{y \in \mathrm{dom}(\Psi) \quad \Psi \vdash (H_1\ \Phi_1) \cap M_1 \Rightarrow N_1 : A_1 \cdots \Psi \vdash (H_n\ \Phi_n) \cap M_n \Rightarrow N_n : A_n}{\Psi \vdash (E\ \Phi) \cap (y\ M_1^1 \ldots M_n^1) \Rightarrow y\ N_1^1 \ldots N_n^1 : a}\ \cap\mathsf{FR}^y$$

$$\frac{h \in \mathrm{dom}(\Psi \cup \Sigma) \quad \Psi \vdash M_1 \cap N_1 \Rightarrow Q_1 : A_1 \cdots \Psi \vdash M_n \cap N_n \Rightarrow Q_n : A_n}{\Psi \vdash (h\ M_1^1 \ldots M_n^1) \cap (h\ N_1^1 \ldots N_n^1) \Rightarrow h\ Q_1^1 \ldots Q_n^n : a}\ \cap\mathsf{RR}$$

$$\frac{\Psi, x{:}A \vdash M \cap N \Rightarrow Q : B}{\Psi \vdash (\lambda x^u{:}A.\ M) \cap (\lambda x^u{:}A.\ )N \Rightarrow \lambda x^u{:}A.\ Q : A \xrightarrow{u} B}\ \cap\mathsf{L}$$

Fig. 8. Unification algorithm: $\Psi \vdash M \cap N \Rightarrow Q : A$

$$\begin{aligned} \forall x. x \neq y \wedge \Phi(x) = 1 \supset \exists i.\ \Phi_i(x) = 1 \wedge \forall j.\ j \neq i \supset \Phi_j(x) = u \\ \Phi(y) = 1 \supset \forall i. \Phi_i(y) = u \end{aligned}$$

Finally define $\Psi \vdash M \cap N = \mathcal{Q} : A$ if $\mathcal{Q} = \{Q \mid \Psi \vdash M \cap N \Rightarrow Q : A\}$.

Some remarks are in order:

—In rule $\cap\mathsf{FF}$ we can assume Φ_1 and Φ_2 are compatible lists of variables, since generalized variables are fully applied and their arguments are in a standard order.

—Since patterns are linear and M and N share no pattern variables, the flex/flex case arises only with distinct variables. This also means we do not have to apply substitutions or perform the customary occurs-check.

—In the flex/rigid and rigid/flex rules, the proviso enforces the typing discipline since each strict variable x must be strict in some premise. If instead y is the projected variable, the modified condition on y takes into account that the head of an application constitutes a strict occurrence; moreover, since y did occur, it is set to u in the rest of the computation, as there are no more requirements on that variable.

—The symmetric rules take the place of an explicit exchange rule that is problematic with respect to termination.

The following example illustrates how the flex/rigid rules, in this case $\cap\mathsf{FR}^c$, make unification on simple patterns finitary. We describe a *unification problem* by omitting the eventually computed solution as $\Psi \vdash M \cap N : A$.

Example 7.3. Consider the unification problem

$$x{:}a \vdash E\ x^1 \cap c\ (F\ x^u)^1\ (F'\ x^u)^1 : a$$

Since x is strict in the left-hand side, there are two ways to apply the $\cap\mathsf{FR}^c$ rule, leading to the following subproblems:

$$\begin{array}{ll} 1.\ x{:}a \vdash E'\ x^1 \cap\ F\ x^u : a & x{:}a \vdash E''\ x^u \cap\ F'\ x^u : a \\ 2.\ x{:}a \vdash E'\ x^u \cap\ F\ x^u : a & x{:}a \vdash E''\ x^1 \cap\ F'\ x^u : a \end{array}$$

Hence the result:

$$x{:}A \vdash E\ x^1 \cap c\ (F\ x^u)^1\ (F'\ x^u)^1 = \{c\ (H\ x^1)^1\ (H'\ x^u)^1, c\ (H\ x^u)^1\ (H'\ x^1)^1\}$$

Note that, similarly to complementation, intersection returns a set of patterns with common terms; again it is possible, in a post-processing phase to make the result exclusive.

The following example illustrates the additional proviso on $\cap\mathtt{FR}^y$

Example 7.4. The unification problem

$$y{:}a \xrightarrow{1} a \xrightarrow{1} a \vdash E\ y^0 \cap y\ (F\ y^1)^1\ (F'\ y^u)^1 : a$$

has no solution, whereas

$$y{:}a \xrightarrow{1} a \xrightarrow{1} a \vdash E\ y^1 \cap y\ (F\ y^1)^1\ (F'\ y^0)^1 = \{y\ (H\ y^1)^1\ (H'\ y^0)^1\} : a$$

This first lemma will be needed to handle the case for unification of generalized variables.

Lemma 7.5. *Assume Φ_1 and Φ_2 are compatible and $\Phi_1 \cap \Phi_2$ is defined. Assume furthermore that $\Gamma_1; \Omega_1; \Delta_1 \vdash \Phi_1$ ok and $\Gamma_2; \Omega_2; \Delta_2 \vdash \Phi_2$ ok. Then $\Gamma_1; \Omega_1; \Delta_1 \vdash M : A$ and $\Gamma_2; \Omega_2; \Delta_2 \vdash M : A$ iff $(\Gamma_1 \cap \Gamma_2); (\Omega_1 \cup \Omega_2); (\Delta_1 \cup \Delta_2) \vdash M : A$.*

Proof. From left to right by induction on the size of $(\Gamma_1 \cup \Gamma_2) \setminus (\Gamma_1 \cap \Gamma_2)$, using tightening (Lemma 5.5). From right to left by appropriate appeals to loosening (Lemma 3.3). □

We introduce two n-ary strict application rules which, for the special case of simple terms, capture the notion of atomic forms more compactly than the previous definition. The rules differ only in whether the head h of the atomic term is a strict variable or unrestricted. These will be needed in the proof of Lemma 7.6 and Lemma 7.7.

$$\frac{(\Gamma, \Delta_i^u); \Omega; \Delta_i^1 \vdash M_i : A_i \quad 1 \leq i \leq n}{\Gamma; \Omega; \Delta \vdash h\ M_1^1 \ldots M_n^1 : B} \xrightarrow{1}E^u$$

where $h\ : A_1 \xrightarrow{1} \ldots \xrightarrow{1} A_n \xrightarrow{1} b$ in $\mathrm{dom}(\Gamma \cup \Sigma)$ and

(1) $\forall x \in \mathrm{dom}(\Delta).\ \exists! i : 1 \leq i \leq n.\ x \in \mathrm{dom}(\Delta_i^1)$.
(2) $\forall i : 1 \leq i \leq n.\ (\Delta_i^u, \Delta_i^1) = \Delta$.

$$\frac{(\Gamma, \Delta_i^u); \Omega; \Delta_i^1 \vdash M_i : A_i \quad 1 \leq i \leq n}{\Gamma; \Omega; \Delta \vdash y\ M_1^1 \ldots M_n^1 : B} \xrightarrow{1}E^1$$

where $y : A_1 \xrightarrow{1} \ldots \xrightarrow{1} A_n \xrightarrow{1} b \in \mathrm{dom}(\Delta)$ and

(1) $\forall x \in \mathrm{dom}(\Delta), x \neq y.\ \exists! i : 1 \leq i \leq n.\ x \in \mathrm{dom}(\Delta^1_i)$.
(2) $\forall i : 1 \leq i \leq n.\ (\Delta^u_i, \Delta^1_i) = \Delta$.
(3) $\forall i : 1 \leq i \leq n.\ y \in \mathrm{dom}(\Delta^u_i)$.

It is straightforward, but tedious to show that these rules can replace the rules for atomic terms. The curious reader is invited to consult [Momigliano 2000a] for details.

We are now ready to address correctness of unification. First we show that our algorithm only computes unifiers, then that the set of unifiers we compute is most general.

LEMMA 7.6 INTERSECTION COMPUTES UNIFIERS. *For any simple linear pattern N_1 and N_2 without shared variables such that $\Psi \vdash N_1 \Uparrow A$ and $\Psi \vdash N_2 \Uparrow A$, for every N such that $\Psi \vdash N_1 \cap N_2 \Rightarrow N$ if $\Psi \vdash M \in \|N\| : A$, then $\Psi \vdash M \in \|N_1\| : A$ and $\Psi \vdash M \in \|N_2\| : A$.*

PROOF. By induction on the structure of $\mathcal{D} :: \Psi \vdash N_1 \cap N_2 \Rightarrow N$ and inversion on $\mathcal{D}' :: \Psi \vdash M \in \|N\| : A$. We show only some of the cases; the others are analogous.

Case. $\mathcal{D}$ ends in $\cap\mathtt{FF}$:

$\Psi \vdash (E_1\ \Phi_1) \cap (E_2\ \Phi_2) \Rightarrow H\ (\Phi_1 \cap \Phi_2) : a$	Assumption
$\Psi \vdash M \in \|H\ (\Phi_1 \cap \Phi_2)\| : a$	Assumption
$\Gamma_i; \Omega_i; \Delta_i \vdash \Phi_i\ ok$ for $i = 1, 2$ for some Γ_i, Ω_i, Δ_i	Determined from Φ_i
$(\Gamma_1 \cap \Gamma_2); (\Omega_1 \cup \Omega_2); (\Delta_1 \cup \Delta_2) \vdash \Phi_1 \cap \Phi_2\ ok$	By Remark 7.1
$\Gamma_i; \Omega_i; \Delta_i \vdash M : a$	By Lemma 7.5 ($\leftarrow$)
$\Psi \vdash M \in \|N_i\| : a$	By rule $\mathtt{grFlx}$

Case. $\mathcal{D}$ ends in $\cap\mathtt{FR}^c$.

$\mathcal{D} :: \Psi \vdash (E\ \Phi) \cap (c\ Q^1_1 \ldots Q^1_n) \Rightarrow c\ N^1_1 \ldots N^1_n : a$	Assumption
$\mathcal{D}_i :: \Psi \vdash (E\ \Phi_i) \cap Q_i \Rightarrow N_i : A_i$, for all $1 \leq i \leq n$	Subderivations
$\Psi \vdash c\ M^1_1 \ldots M^1_n \in \|c\ N^1_1 \ldots N^1_n\| : a$	Assumption
$\Psi \vdash M_i \in \|N_i\| : A_i$	By inversion
$\Psi \vdash M_i \in \|Q_i\| : A_i$ and $\Psi \vdash M_i \in \|E_i\ \Phi_i\| : A_i$	By i.h. on $\mathcal{D}_i$
$(\Gamma_i, \Delta^u_i); \Omega; \Delta^1_i \vdash \Phi_i\ ok$ and $(\Gamma_i, \Delta^u_i); \Omega; \Delta^1_i \vdash M_i : A_i$	By rule $\mathtt{grFlx}$
$\Gamma; \Omega; \Delta \vdash c\ M^1_1 \ldots M^1_n : a$	By rule $\xrightarrow{1} E^u$
$\Psi \vdash c\ M^1_1 \ldots M^1_n \in \|E\ \Phi\| : a$	By rule $\mathtt{grFlx}$
$\Psi \vdash c\ M^1_1 \ldots M^1_n \in \|c\ Q^1_1 \ldots Q^1_n\| : a$	By rule $\mathtt{grApp}$

□

The second part consists of showing that any unifier of two patterns is an instance of an element from the computed set of unifiers.

LEMMA 7.7 INTERSECTIONS ARE MOST GENERAL. *For any simple linear patterns N_1 and N_2 without shared variables such that $\Psi \vdash N_1 \Uparrow A$ and $\Psi \vdash N_2 \Uparrow A$, if $\Psi \vdash M \in \|N_1\| : A$ and $\Psi \vdash M \in \|N_2\| : A$, then there is N such that $\Psi \vdash N_1 \cap N_2 \Rightarrow N : A$ and $\Psi \vdash M \in \|N\| : A$.*

PROOF. By simultaneous induction on the structure of $\mathcal{D}_1 :: \Psi \vdash M \in \|N_1\| : A$ and $\mathcal{D}_2 :: \Psi \vdash M \in \|N_2\| : A$.

Case. $\mathcal{D}_1, \mathcal{D}_2$ end in `grFlx`:

$\Gamma_i;\Omega_i;\Delta_i \vdash \Phi_i\ ok$ and $\Gamma_i;\Omega_i;\Delta_i \vdash M : a$ for $i = 1,2$ — Subderivations
$\Phi_1 \cap \Phi_2$ is defined — By exclusivity (Lemma 3.7)
$\Psi \vdash (E_1\ \Phi_1) \cap (E_2\ \Phi_2) \Rightarrow H\ (\Phi_1 \cap \Phi_2) : a$ — By rule $\cap$FF
$(\Gamma_1 \cap \Gamma_2);(\Omega_1 \cup \Omega_2);(\Delta_1 \cup \Delta_2) \vdash M : a$ — By Lemma 7.5($\rightarrow$)
$(\Gamma_1 \cap \Gamma_2);(\Omega_1 \cup \Omega_2);(\Delta_1 \cup \Delta_2) \vdash \Phi_1 \cap \Phi_2\ ok$ — By Remark 7.1
$\Psi \vdash M \in \|H\ (\Phi_1 \cap \Phi_2)\| : a$ — By rule `grFlx`

Case. $\mathcal{D}_1$ ends in `grFlx` and $\mathcal{D}_2$ ends in `grApp`: there are two cases depending on whether the head of N_2 is a constant or a parameter.

Subcase. The head of N_2 is a constant c.

$\Psi \vdash M \in \|c\ Q_1^1 \dots Q_n^1\| : a$ — Assumption
$M = c\ M_1^1 \dots M_n^1$ and $\mathcal{D}_i^2 :: \Psi \vdash M_i \in \|Q_i\| : A_i$ for all $1 \le i \le n$ — Subderivation
$\Psi \vdash c\ M_1^1 \dots M_n^1 \in \|E\ \Phi\| : a$ — Assumption
$\Gamma;\Omega;\Delta \vdash c\ M_1^1 \dots M_n^1 : a$ and $\Gamma;\Omega;\Delta \vdash \Phi\ ok$ — By inversion on rule `grFlx`
$(\Gamma, \Delta_i^u);\Omega;\Delta_i^1 \vdash M_i : A_i$ for some Δ_i^u, Δ_i^u satisfying (1) and (2)
By inversion on rule $\xrightarrow{1}E^u$
$\mathcal{D}_i^1 :: \Psi \vdash M_i \in \|E_i\ \Phi_i\| : A_i$ for Φ_i such that $(\Gamma, \Delta_i^u);\Omega;\Delta_i^1 \vdash \Phi_i\ ok$
By rule `grFlx`
$\mathcal{D}_i :: \Psi \vdash (E_i\ \Phi_i) \cap Q_i \Rightarrow N_i : A_i$ and $\Psi \vdash M_i \in \|N_i\| : A_i$ — By i.h. on $\mathcal{D}_i^1, \mathcal{D}_i^2$
$\mathcal{D} :: \Psi \vdash (E\ \Phi) \cap (c\ Q_1^1 \dots Q_n^1) \Rightarrow c\ N_1^1 \dots N_n^1 : a$ — By rule $\cap\mathtt{FR}^c$
$\Psi \vdash c\ M_1^1 \dots M_n^1 \in \|c\ N_1^1 \dots N_n^n\| : a$ — By rule `grApp`

Subcase. Proceed as above, but using inversion on rule $\xrightarrow{1}E^1$

Case. $\mathcal{D}_2$ ends in `grFlx` and $\mathcal{D}_1$ ends in `grApp`: symmetrical to the above.

Case. $\mathcal{D}_1, \mathcal{D}_2$ end in `grLam`: straightforward by induction hypothesis.

Case. $\mathcal{D}_1, \mathcal{D}_2$ end in `grApp`: a straightforward appeal to the induction hypothesis as in the above case.

□

The correctness of the algorithm for pattern intersection follows directly from the preceding two lemmas.

THEOREM 7.8 CORRECTNESS OF PATTERN INTERSECTION.
Assume N_1 and N_2 are simple linear patterns without shared variables such that $\Psi \vdash N_1 \Uparrow A$ and $\Psi \vdash N_2 \Uparrow A$. Then $\Psi \vdash M \in \|N_1\| : A$ and $\Psi \vdash M \in \|N_2\| : A$ iff $\Psi \vdash M \in \|N_1 \cap N_2\| : A$.

Also note that the intersection of linear simple patterns is again a simple linear pattern.

THEOREM 7.9 CLOSURE UNDER INTERSECTION. *Assume M and N are simple linear patterns with $\Psi \vdash M \Uparrow A$ and $\Psi \vdash N \Uparrow A$. Then $\Psi \vdash M \cap N \Rightarrow Q : A$ implies that Q is a simple linear pattern and $\Psi \vdash Q \Uparrow A$.*

PROOF. By induction on the structure of the derivation of $\Psi \vdash M \cap N \Rightarrow Q : A$. □

8. THE ALGEBRA OF LINEAR SIMPLE PATTERNS

An interesting and natural question is whether complementation is involutive. The answer is of course positive, since the latter is a boolean property and the complement operation has been shown to satisfy "tertium non datur" and the principle of non-contradiction. However, the reader should keep in mind that the *representation* of the set $\mathrm{Not}(\mathrm{Not}(N))$ may be different from $\{N\}$, even though the two sets are guaranteed to have the same set of ground instances. Since on finite set of patterns we also have intersection and set-theoretic union, we obtain a boolean algebra. For the sake of readability, we introduce the following notation: $\mathrm{Pat}_A(\Psi)$ denotes the finite set of linear simple patterns M with $\Psi \vdash M : A$. In the following, we also drop the type information and overload the singleton pattern notation.

Definition 8.1. For $\mathcal{M}, \mathcal{N} \in \mathrm{Pat}_A(\Psi)$, define:

$$\mathcal{M} \cap \mathcal{N} = \bigcup_{M \in \mathcal{M}, N \in \mathcal{N}} M \cap N$$

$$\mathrm{Not}(\mathcal{M}) = \bigcap_{M \in \mathcal{M}} \mathrm{Not}(M)$$

Those operations on sets of patterns satisfy the same properties that singleton intersection and complementation do.

Corollary 8.2 Correctness of Set Intersection.
For $\mathcal{N}_1, \mathcal{N}_2 \in \mathrm{Pat}_A(\Psi)$, $\Psi \vdash M \in \|\mathcal{N}_1\| : A$ *and* $\Psi \vdash M \in \|\mathcal{N}_2\| : A$ *iff* $\Psi \vdash M \in \|\mathcal{N}_1 \cap \mathcal{N}_2\| : A$.

Corollary 8.3 Correctness of Set Complement.
For $\mathcal{N} \in \mathrm{Pat}_A(\Psi)$, $\Psi \vdash M \in \|\mathrm{Not}(\mathcal{N})\| : A$ *iff* $\Psi \not\vdash M \in \|\mathcal{N}\| : A$

As we have remarked earlier, we can define the relative complement operation by using complement and intersection. Its correctness follows immediately from the correctness of pattern set intersection and complement.

Definition 8.4 *Relative Complement.*
Given $\mathcal{M}, \mathcal{N} \in \mathrm{Pat}_A(\Psi)$, we define $\mathcal{M} - \mathcal{N} = \mathcal{M} \cap \mathrm{Not}(\mathcal{N})$.

The properties above mean that we can organize, for a given signature Σ, context Ψ, and a type A, finite sets of simple linear patters into a Boolean algebra by taking equality as extensional identity on sets of terms without existential variables. In symbols, for $\mathcal{N}_1, \mathcal{N}_2 \in \mathrm{Pat}_A(\Psi)$:

$$\mathcal{N}_1 \simeq \mathcal{N}_2 \text{ iff } \|\mathcal{N}_1\| = \|\mathcal{N}_2\|$$

Under this interpretation, the **0** element is the empty set and the **1** element the singleton set containing the η-expansion of a generalized existential variable of the appropriate type that may depend on all variables in the context Ψ.

$$\begin{aligned} \mathbf{0} &= \emptyset \\ \mathbf{1} &= \{\lambda x_1^u{:}A_1 \ldots \lambda x_n^u{:}A_n.\, E\ \Psi^u\ x_1^u \ldots x_n^u\} \end{aligned}$$

where $A = A_1 \overset{u}{\to} \cdots A_n \overset{u}{\to} a$.

THEOREM 8.5. *Consider the algebra* $\langle \mathrm{Pat}_A(\Psi), \cup, \cap, \mathrm{Not}, \mathbf{1}, \mathbf{0}\rangle$. *Then the following holds:*

(1) $\mathcal{M} \cap \mathcal{M} \simeq \mathcal{M}$.

(2) $\mathcal{M} \cap \mathcal{N} \simeq \mathcal{N} \cap \mathcal{M}$.

(3) $\mathcal{M} \cap (\mathcal{N} \cup \mathcal{P}) \simeq (\mathcal{M} \cap \mathcal{N}) \cup (\mathcal{M} \cap \mathcal{P})$.

(4) $\mathcal{M} \cap (\mathcal{N} \cap \mathcal{P}) \simeq (\mathcal{M} \cap \mathcal{N}) \cap \mathcal{P}$.

(5) $\mathrm{Not}(\mathrm{Not}(\mathcal{M})) \simeq \mathcal{M}$.

(6) $\mathrm{Not}(\mathbf{1}) \simeq \mathbf{0}$.

(7) $\mathrm{Not}(\mathbf{0}) \simeq \mathbf{1}$.

PROOF. From Corollaries 8.2 and 8.3 and the fact that $\cup$ is set-theoretic. □

COROLLARY 8.6. *The algebra of finite sets of simple linear patterns is boolean.*

It is notable that the $\cup$ operator must be set-theoretic union rather than anti-unification or generalization, as traditional in lattice-theoretic investigations of the algebra of terms [Lassez et al. 1988]. The problem is the intrinsically classical nature of complementation which is not compatible with the very irregular structure of the lattice of terms where the smallest upper bound is interpreted as anti-unification.

We end this section showing how pattern complement can be used as a building block of our main application, that is a clause complement algorithm [Barbuti et al. 1990]. In (higher-order) logic programming, in fact, pattern complement is a necessary component in any algorithm to synthesize the negation of a given program. This synthesis includes two basic operations: negation to compute the complements of heads of clauses in the definition of a predicate, and intersection to combine results of negating individual clause heads. In this paper we have provided algorithms to compute both. A full development for the higher-order case can be found in [Momigliano 2000a].

Example 8.7. We can combine Example 2.1 and 2.2 and consider the following trivial program, which encodes when an object-level lambda term is a $\beta\eta$-redex:

$$\begin{array}{lll} \mathit{betardx} & : & \mathit{isredx}\ (\mathit{app}\ (\mathit{lam}\ (\lambda x^u{:}\mathit{exp}.\ E\ x^u))\ F). \\ \mathit{etardx} & : & \mathit{isredx}\ (\mathit{lam}\ (\lambda x^u{:}\mathit{exp}.\ \mathit{app}\ (E\ x^0)\ x)). \end{array}$$

We can compute the complement of both heads, as follows:

$$
\begin{array}{l}
\mathrm{Not}\{app\ (lam\ (\lambda x^u{:}exp.\ E\ x^u))\ F, lam(\lambda x^u{:}exp.\ app\ (E\ x^0)\ x)\} \\
= \mathrm{Not}(app\ (lam\ (\lambda x^u{:}exp.\ E\ x^u))\ F) \cap \mathrm{Not}(lam(\lambda x^u{:}exp.\ app\ (E\ x^0)\ x)) \\
= \{lam\ (\lambda x^u{:}exp.\ H\ x^u), app\ (app\ H\ H')\ H''\} \\
\quad \cap \{lam\ (\lambda x^u{:}exp.\ app\ (H\ x^1)\ (H'\ x^u)), \\
\qquad lam\ (\lambda x^u{:}exp.\ app\ (H\ x^u)\ (app\ (H'\ x^u)\ (H'' x^u))), \\
\qquad lam\ (\lambda x^u{:}exp.\ app\ (H\ x^u)\ (lam\ (\lambda y^u{:}exp.\ H'\ x^u\ y^u))), \\
\qquad lam\ (\lambda x^u{:}exp.\ lam\ (\lambda y^u{:}exp.\ H\ x^u\ y^u)), \\
\qquad lam\ (\lambda x^u{:}exp.\ x), \\
\qquad app\ H\ H'\} \\
= \{lam\ (\lambda x^u{:}exp.\ app\ (H\ x^1)\ (H'\ x^u)), \\
\quad lam\ (\lambda x^u{:}exp.\ app\ (H\ x^u)\ (app\ (H'\ x^u)\ (H''\ x^u))), \\
\quad lam\ (\lambda x^u{:}exp.\ app\ (H\ x^u)\ (lam\ (\lambda y^u{:}exp.\ H'\ x^u\ y^u))) \\
\quad lam\ (\lambda x^u{:}exp.\ lam\ (\lambda y^u{:}exp.\ H'\ x^u\ y^u)), \\
\quad lam\ (\lambda x^u{:}exp.\ x), \\
\quad app\ (app\ H\ H')\ H''\}
\end{array}
$$

This yields the negation of that program, that is the complementary clauses:

$$
\begin{array}{lll}
nb_1 & : & non\_isredx\ (lam\ (\lambda x^u{:}exp.\ app\ (H\ x^1)\ (H'\ x^u))). \\
nb_2 & : & non\_isredx\ (lam\ (\lambda x^u{:}exp.\ app\ (H\ x^u)\ (app\ (H'\ x^u)\ (H''\ x^u)))). \\
nb_3 & : & non\_isredx\ (lam\ (\lambda x^u{:}exp.\ app\ (H\ x^u)\ (lam\ (\lambda y^u{:}exp.\ H'\ x^u\ y^u)))). \\
nb_4 & : & non\_isredx\ (lam\ (\lambda x^u{:}exp.\ lam\ (\lambda y^u{:}exp.\ H\ x^u\ y^u))). \\
nb_5 & : & non\_isredx\ (lam\ (\lambda x^u{:}exp.\ x)). \\
nb_6 & : & non\_isredx\ (app\ (app\ H\ H')\ H'').
\end{array}
$$

9. CONCLUSIONS

In this paper we have been concerned with the relative complement problem for higher-order patterns. As we have seen, the complement operation does not generalize easily from the first-order case. Indeed, the complement of a partially applied higher-order pattern cannot be described by a pattern, or even a by finite set of patterns. The formulation of the problem suggests that we should consider a λ-calculus with an internal notion of *strictness* so that we can directly express that a term must depend on a given variable. We have developed such a calculus and we have shown that via a suitable embedding in our calculus the complement of a linear pattern is a finite set of linear patterns and unification of two patterns is decidable and leads to a finite set of most general unifiers. Moreover, they form a boolean algebra under set-theoretic union, intersection (implemented via unification) and the complement operation.

The latter item brings up the question if we can actually decide extensional equality between, and membership of terms in, finite sets of simple terms. For membership, one can see that $\Psi \vdash M \in \|N_1, \ldots, N_n\|$ iff M unifies with some N_i. As far as equality is concerned between say $\|M_1, \ldots, M_m\|$ and $\|N_1, \ldots, N_n\|$ calculate the two relative complements $\{M_1, \ldots, M_m\} - \{N_1, \ldots, N_n\}$ and $\{N_1, \ldots, N_n\} - \{M_1, \ldots, M_m\}$ and then check if they are both empty. An emptiness check would rely on the decidability of inhabitation in the underlying calculus. We conjecture this question to be decidable for the strict λ-calculus and we plan to address this question in future work.

Our main application is the transformational approach to negation in higher-order logic programming [Barbuti et al. 1990], where pattern complement and unification is a necessary component. We plan to extend the results to dependent types to endow intentionally weak frameworks such as Twelf [Schürmann and Pfenning 1998] with a logically meaningful notion of negation along these lines.

It may be argued that the restriction to simple terms is somewhat ad hoc. Ideally, one would have a complement algorithm for the full strict lambda-calculus (including vacuous types). Yet, this seems to be ill-defined, because "occurrence" no longer has the desired meaning once we lift the principle that constructors should be strict in their argument. As we have remarked earlier, it is possible to describe complement and unification algorithms for a larger fragment than treated here by allowing arbitrary abstractions, if we adhere to the above strictness assumption for constructors. The technical development is not difficult but entails a proliferation of rules to cover the new abstraction cases, as well as the duplication of all rules concerning strict application in versions similar to the $\stackrel{1}{\to}E^u$ and $\stackrel{1}{\to}E^1$ typing rules.

Finally, it is our contention that the strict λ-calculus that we have introduced has independent interest in the investigation of sub-structural logics. Our type system is simple and uniform and arguably more elegant than those ones presented in the literature (see the earlier discussion of related work at the end of Section 4). Moreover, the explicit introduction of the notion of *vacuous* or *irrelevant* variables can be useful in a variety of contexts. In fact, the second author has suggested some unexpected usage of those variables in type theory for uses in reasoning about staged computation [Pfenning 2000] and proof compression in logical frameworks [Pfenning 2001b]. Furthermore, extending a linear λ-calculus with vacuous variables permits more programs under type assignment; for example a term such as $\lambda x.\ \lambda y.\ x \otimes (\lambda w.\ y)\ x$, which is traditionally considered *not* linear, can be given the linear type $A \multimap B \multimap (A \otimes B)$. This carries over to the study of *explicit substitutions* in resource-conscious λ-calculi [Ghani et al. 1998] where it might clarify the logical status of the *extension operator*.

ACKNOWLEDGMENTS

We would like to thank Roberto Virga, who discovered an error in an earlier version of this paper, and Iliano Cervesato and Carsten Schürmann for several discussions and comments on a draft of this paper.

REFERENCES

Anderson, A. and Belnap, N. 1975. *Entailment. The Logic of Relevance and Necessity*. Vol. 1. Princeton University Press.

Baker-Finch, C. A. 1993. Relevance and contraction: A logical basis for strictness and sharing analysis. Tech. Rep. ISE RR 34/94, University of Canberra.

Barbuti, R., Mancarella, P., Pedreschi, D., and Turini, F. 1990. A transformational approach to negation in logic programming. *Journal of Logic Programming 8*, 201–228.

Belnap, N. 1974. Functions which really depend on their argument. Manuscript.

Belnap, N. D. 1993. Life in the undistributed middle. In *Substructural Logics*, K. Došen and P. Schroeder-Heister, Eds. Oxford University Press, 31–42.

Church, A. 1941. *The Calculi of Lambda Conversion*. Princeton University Press.

CHURCH, A. 1951. The weak theory of implication. In *Kontrolliertes Denken*, A.Menne, A. Wilhelmy, and H. Angsil, Eds. Kommissions-Verlag Karl Alber, Munich, 22–27.

COMON, H. 1991. Disunification: a survey. In *Computational Logic: Essays in Honor of Alan Robinson*, J.-L. Lassez and G.Plotkin, Eds. MIT Press, Cambridge, MA, 322–359.

DESPEYROUX, J., PFENNING, F., AND SCHÜRMANN, C. 1997. Primitive recursion for higher-order abstract syntax. In *Proceedings of the Third International Conference on Typed Lambda Calculus and Applications (TLCA'97)*, R. Hindley, Ed. Springer-Verlag LNCS, Nancy, France, 147–163. An extended version is available as Technical Report CMU-CS-96-172, Carnegie Mellon University.

GHANI, N., PAIVA, V. D., AND RITTER, E. 1998. Linear explicit substitutions. Tech. Rep. CSR-98-2, University of Birmingham, School of Computer Science. Mar.

HANUS, M. AND PREHOFER, C. 1996. Higher-order narrowing with definitional trees. In *Proc. Seventh International Conference on Rewriting Techniques and Applications (RTA'96)*. Springer LNCS 1103, 138–152.

HELMANN, G. 1977. Completeness of the normal typed fragment of the λ-system u. *Journal of Philosophical Logic 6,* 2, 33–46.

JENSEN, T. P. 1991. Strictness Analysis in Logical Form. In *Functional Programming Languages and Computer Architectures*, J. Hughes, Ed. Lecture Notes in Computer Science, vol. 523. Springer, Berlin, Harvard, Massachusetts, USA, 352–366.

KUNEN, K. 1987. Answer sets and negation-as-failure. In *Proceedings of the Fourth International Conference on Logic Programming (ICLP '87)*, J.-L. Lassez, Ed. MIT Press, Melbourne, Australia, 219–228.

LASSEZ, J.-L., MAHER, M., AND MARRIOT, K. 1988. Unification revisited. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, Los Altos,CA.

LASSEZ, J.-L. AND MARRIOT, K. 1987. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning 3,* 3 (Sept.), 301–318.

LUGIEZ, D. 1995. Positive and negative results for higher-order disunification. *Journal of Symbolic Computation 20,* 4 (Oct.), 431–470.

MILLER, D. 1991. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation 1,* 4, 497–536.

MOMIGLIANO, A. 2000a. Elimination of negation in a logical framework. Ph.D. thesis, Department of Philosophy, Carnegie Mellon University. Available as Technical Report CMU-CS-00-175.

MOMIGLIANO, A. 2000b. Elimination of negation in a logical framework. In *Proceedings of the 14th Annual Conference of the European Association for Computer Science Logic (CSL'00)*, P. G. Clote and H. Schwichtenberg, Eds. Springer-Verlag LNCS, Fischbachau, Germany, 411–426.

MYCROFT, A. 1980. The theory and practice of transforming call-by-need to call-by-value. In *Proceedings of the 4th International Symposium on Programming.* Lecture Notes in Computer Science, vol. 83. Springer Verlag, 269–281.

NIPKOW, T. 1991. Higher-order critical pairs. In *Sixth Annual IEEE Symposium on Logic in Computer Science*, G. Kahn, Ed. Amsterdam, The Netherlands, 342–349.

NIPKOW, T. 1993. Orthogonal higher-order rewrite systems are confluent. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, M. Bezem and J. Groote, Eds. Utrecht, The Netherlands, 306–317.

PFENNING, F. 1991a. Logic programming in the LF logical framework. In *Logical Frameworks*, G. Huet and G. Plotkin, Eds. Cambridge University Press, 149–181.

PFENNING, F. 1991b. Unification and anti-unification in the Calculus of Constructions. In *Sixth Annual IEEE Symposium on Logic in Computer Science.* Amsterdam, The Netherlands, 74–85.

PFENNING, F. 2000. Reasoning about staged computation. In *Proceedings of the International Workshop on Semantics, Applications, and Implementation of Program Generation (SAIG 2000)*, W. Taha, Ed. Springer-Verlag LNCS 1924, Montreal, Canada, 5–6. Abstract of invited talk.

PFENNING, F. 2001a. *Computation and Deduction.* Cambridge University Press. In preparation. Draft from April 1997 available electronically.

PFENNING, F. 2001b. Intensionality, extensionality, and proof irrelevance in modal type theory. In *Proceedings of the 16th Annual Symposium on Logic in Computer Science (LICS'01)*, J. Halpern, Ed. IEEE Computer Society Press, Boston, Massachusetts, 221–230. Extended version available as Technical Report CMU-CS-01-116, Department of Computer Science, Carnegie Mellon University, September 2001.

SCHÜRMANN, C. AND PFENNING, F. 1998. Automated theorem proving in a simple meta-logic for LF. In *Proceedings of the 15th International Conference on Automated Deduction (CADE-15)*, C. Kirchner and H. Kirchner, Eds. Springer-Verlag LNCS 1421, Lindau, Germany, 286–300.

TSUNG-MIN, K. AND MISHRA, P. 1989. Strictness Analysis: A New Perspective Based on Type Inference. In *FPCA '89, Functional Programming Languages and Computer Architecture*. ACM Press, New York, London, UK, September 11–13.

WRIGHT, D. A. 1992. Reduction types and intensionality in the lambda-calculus. Ph.D. thesis, University of Tasmania.

WRIGHT, D. A. 1996. Linear, strictness and usage logics. In *Proceedings of Conference on Computing: The Australian Theory Symposium*, M. E. Houle and P. Eades, Eds. Australian Computer Science Communications, Townsville, 73–80.

Submitted September 2001