Soundness and Principal Contexts for a Shallow Polymorphic Type System based on Classical Logic

Alexander J. Summers

Department of Computing, Imperial College London, 180 Queens Gate, South Kensington, London

Abstract

In this paper we investigate how to adapt the well-known notion of *ML-style polymorphism* (shallow polymorphism) to a term calculus based on a Curry-Howard correspondence with classical sequent calculus, namely, the $\mathcal{X}^i$-calculus. We show that the intuitive approach is unsound, and pinpoint the precise nature of the problem. We define a suitably refined type system, and prove its soundness. We then define a notion of *principal contexts* for the type system, and provide an algorithm to compute these, which is proved to be sound and complete with respect to the type system. In the process, we formalise and prove correctness of *generic unification*, which generalises Robinson's unification to shallow-polymorphic types.

Key words: Curry-Howard, classical logic, generic unification, principal types, cut elimination

1. Introduction

Polymorphism is a powerful aspect of most modern programming languages. It is a mechanism for allowing a program to be applied in various contexts which each expect different types, and allows flexibility and reuse of code. In a non-polymorphic programming language for example, even if a function's behaviour is independent of the type of its argument, it must be redefined for each such type.

In recent years, there has been a wealth of work sparked by the seminal paper of Griffin [1], concerning the extension of the Curry-Howard Correspondence [2, 3, 4] to various classical logics (e.g., [5, 6, 7, 8]). In particular, the reduction

Email addresses: `alexander.summers@inf.ethz.ch` (Alexander J. Summers)
URL: `http://people.inf.ethz.ch/summersa` (Alexander J. Summers)

behaviour of calculi based on classical logic has been shown to have strong and natural correspondences with control operators in functional languages (e.g., [9, 10, 11, 12, 13]). In this way, proof reductions can be given a strong computational interpretation, and it seems feasible in principle that a calculus based on a Curry-Howard correspondence with a classical logic could be used as the basis for a practical programming language.

In this paper we investigate how to adapt the well-known notion of ML-style polymorphism (which we call *shallow polymorphism* in this paper) to a term calculus based on classical logic, namely, the $\mathcal{X}^i$-calculus [14]. The problems here are two-fold: firstly what difference does the extension to a classical logic setting make, and secondly, how should polymorphism be implemented in the unusual setting of the sequent calculus? We will show that the logical basis of the sequent calculus (as opposed to variants of the λ-calculus, which correspond to logic in a natural deduction style) provides a clearer understanding of the issues involved.

We will first review the key aspects of the Hindley-Milner approach, and then examine how they can be brought to the more-general setting of the $\mathcal{X}^i$-calculus. This turns out to be non-trivial; not only do some aspects require extra machinery to be adapted naturally to the sequent calculus setting, but the intuitive approach fails for the general system; witness reduction is violated by the more-general reductions possible in the $\mathcal{X}^i$-calculus. This problem was not identified in the published work by the author [15], in which a witness reduction result for this type system is erroneously claimed. We examine here this problem, and identify three sufficient conditions for such a polymorphic type system to *fail* to be sound in this way. We compare with examples in the literature; in particular the unsoundness of ML when extended with various non-functional concepts, such as exceptions and control operators. By exploring the exact cause of the unsoundness, we show how the type system can be amended in a novel way, which is more general than one of the standard approaches in the context of ML.

Having obtained a suitably general polymorphic type system, we then move on to the question of principal types. We show that we can define a notion of principal types (which is more-suitably named "principal contexts" for this calculus) similar to the well-known result for ML. We define a principal typing algorithm, and prove soundness and completeness of the algorithm. In order to define the algorithm correctly, we define the concept of *generic unification*, whose naïve definition is shown to be incomplete. We define this operation precisely, and prove a most-general-unifier property, generalising the classical unification results of Robinson [16].

Finally, we discuss the definition of a naturally-arising 'dual' notion of poly-

morphism, using existential quantification ($\exists$) rather than universal. Although in a programming setting based on intuitionistic logic (as λ-calculus and indeed ML can be seen to be) the addition of existential quantification does not make any new programs typeable, in a classical logic setting there are programs which can be made typeable in this way. We also discuss the idea of a type system combining *both* kinds of quantification at once. Such a system can type even more programs: the roles of the quantifiers can be seen to be complimentary, but requires a non-trivial extension of our principal contexts results, and is left for future work.

This paper builds on and corrects the work previously presented in [15], and is based on otherwise unpublished material from the author's PhD thesis [14].

1.1. Overview

Section 2 covers the background material for the rest of the paper. We begin with a brief overview of some notation and conventions in section 2.1, which is followed by a review of the basic definitions of ML and shallow polymorphism in section 2.2. Section 2.3 discusses principal types. Section 2.4 provides an introduction to the $\mathcal{X}^i$-calculus, which we use as the basis for our work. Section 3 is the body of the paper. It begins with a discussion of the intuitive, unsound approach to a polymorphic type system in the classical logic setting (section 3.1). We then discuss the precise source of the unsoundness, and define an improved type system in Section 3.2. Section 3.4 deals with the problem of principal contexts for this new type system. We discuss some possible extensions and future work in section 4 and conclude in section 5. Detailed proofs of all of the important results can be found in Appendix B.

2. Background

2.1. Notation

We give here a brief summary of the notations and conventions employed in this paper, particularly regarding the introduction of binders within our type language, and the handling of substitutions, renamings etc., within types.

The binders we will be employing within types come in the form of (second order) logical quantifiers: specifically $\forall$ (universal quantification). We will usually refer to types containing these symbols as *quantified types*. We choose to distinguish between bound and free 'type variables', using different notation and language to describe each. The 'free type variables' are referred to as *atomic types*, and are represented by $\varphi, \varphi', \varphi_1, \varphi_2$ etc.. The *bound type variables* are chosen from the latter part of the uppercase Roman alphabet; i.e. W, X, Y, Z, X_1, X_2 etc..

For example, we can write both $\varphi \to \varphi$ and $\forall X.(X \to X)$ as types. We believe that maintaining a clear distinction between these two notions is both natural (since their meanings and behaviours are quite different) and illustrative. We believe the clear distinction to be necessary for the presentation of our technical results, which mostly require a careful treatment of quantified types.

We use the early part of the uppercase Roman alphabet ($A, B, C, D, E, F, A', A_1$ etc.) to represent Curry types[1]. In order to describe quantified types separately from Curry types, we use the overlined version of this notation, i.e., the symbols $\overline{A}, \overline{B}, \overline{C}$ etc. (note however that we may also write $\overline{A}$ for Curry Types). Types which may be quantified are referred to as *generic types*.

When introducing bindings, we require operations to rename free atomic types φ with bound type variables X, and we write this operation as $\overline{A}[X/\varphi]$. Dually, we also require the replacement of bound type variables X with (Curry) types B, which we write as $\overline{A}[B/X]$. Note that these operations are kept distinct from the usual Curry substitutions, which replace atomic types with Curry types. We assume that all of these operations bind tighter than any logical connectives in the types: for example, $\forall X.A[X/\varphi]$ should be read as $\forall X.(A[X/\varphi])$.

Since we will frequently be concerned with the question of which atomic types occur within a type, it is convenient to define the set $atoms(\overline{A})$ to be the set of all atomic types in a type $\overline{A}$ (note: this does not include bound type variables). We can then write $\varphi \in atoms(\overline{A})$ to state that an atomic type occurs within a (generic) type $\overline{A}$. For convenience, we allow ourselves to write this as $\varphi \in \overline{A}$ when this does not cause confusion. We extend this notation to contexts in the obvious way, e.g., $\varphi \in \langle \Gamma; \underline{\Delta} \rangle$ means that there exists a statement in $\langle \Gamma; \Delta \rangle$ featuring a type $\overline{A}$ such that $\varphi \in \overline{A}$.

When discussing generic types, i.e. types of the form $\forall X_1.\forall X_2.\ldots.\forall X_n.A$, we find it convenient to introduce the $\overrightarrow{\ }$ notation, e.g., $\overrightarrow{\forall X_i}.A$. We do not explicitly quantify over the subscripts, but it is always intended that a subscript i, j, k etc. is bound under the corresponding $\overrightarrow{\ }$. We extend this notation slightly informally to facilitate the statement and proof of our results, by using it as a shorthand for repetition in other statements. For example, we write $\{\overrightarrow{\varphi_i}\}$ for the set $\{\varphi_1, \varphi_2, \ldots\}$, and we write $\varphi_i \in \overline{A}$ to mean "$\varphi_1 \in \overline{A}$ and $\varphi_2 \in \overline{A}$ etc.."

As usual, we assume all binders in types are α-converted appropriately to avoid clashes, and that all substitutions and renamings which cross binders are capture-avoiding.

[1] As usual, Curry types are those built from atomic types and the binary $\to$ operator.

2.2. The Hindley-Milner Type System for ML

The archetypal example of a shallow-polymorphic type system is the Hindley-Milner [17, 18] type system, which underlies the type system for the ML programming language. The main advantages of this approach over that of System F [19, 20], for example, are practical: type checking and type assignment (within certain constraints, as we will explain) are decidable, and can be implemented by relatively straightforward algorithms [21]. In contrast, it has been shown that the corresponding problems are undecidable for System F [22].

We briefly recall here the basic definitions, and refer the reader to Damas and Milner's work [21] for the details.

Definition 2.1 (ML Syntax)**.** *The syntax of* ML terms *is defined by:*

$$M, N ::= x \mid M\ N \mid \lambda x.M \mid \texttt{Fix}\, g.M \mid \mathit{let}\ x = M\ \texttt{in}\ N$$

The construct $\texttt{Fix}\, g.M$ *is included to allow typeable recursion in the calculus. For simplicity in our discussions of polymorphism we choose to study the subset of ML expressions without* `Fix`*, and will hereafter only consider ML expressions within this subset.*

ML values *are defined by:* $V ::= x \mid \lambda x.M$

Definition 2.2 (ML Reductions)**.** *The reduction relation in ML is the transitive, compatible closure of the following two rules:*

$$\begin{array}{rcl} (\lambda x.M)V & \rightarrow_{\mathrm{ML}} & M[V/x] \\ (\mathit{let}\ x = V\ \texttt{in}\ M) & \rightarrow_{\mathrm{ML}} & M[V/x] \end{array}$$

We choose to present types and type assignment rules using the approach of Damas and Milner [21], as this gives a clearer treatment than that of [18].

Definition 2.3 (Generic Types [21][2])**.** *The set of* generic types *is built from the usual Curry types by allowing any number (possibly zero) of* $\forall$ *quantifiers to be*

[2]Damas and Milner call these "type schemes" rather than "generic types" [21]. We also distinguish between atomic types and bound type variables, whereas they choose not to. In fact, in [18] the set of Curry types is also extended with type constants, to represent concrete types such an integers, booleans, etc. However, this is more a practical consideration, and we leave them out in these discussions for simplicity. Since, as we shall see, our type derivations are closed under substitution on atomic types, we can imagine extending these substitutions to replace atomic types with concrete types; everything works out in the same way.

built on the outside. We recall below the definition of Curry types A, B (extended with occurrences of type variables X), and then define generic types $\overline{A}$:

$$\begin{aligned} A, B &::= \varphi \mid X \mid (A \to B) \\ \overline{A} &::= \forall X_1.\forall X_2.\ldots\forall X_n.A \quad (n \geq 0) \end{aligned}$$

A generic type $\overline{A}$ is well-formed *if it contains no free type variables (type variables X not occurring under a corresponding $\forall X.$ binding). We will assume all types to be well-formed in this paper, unless otherwise stated.*

Note that in the case $n = 0$ in the definition of generic types, we assert that any Curry type A is a generic type itself. We use the symbol Γ to represent a basis of assumptions, as before. We write $\Gamma \vdash_{\text{ML}} M : \overline{A}$ to mean 'there is a type derivation assigning the type (scheme) $\overline{A}$ to the term M under the basis of assumption Γ'. The form of these type derivations is defined as follows:

Definition 2.4 ([21])**.** ML-type assignment *is defined by the following derivation rules.*

$$\frac{}{\Gamma, x : \overline{A} \vdash_{\text{ML}} x : \overline{A}}\ (ax) \qquad \frac{\Gamma \vdash_{\text{ML}} M : \overline{A} \quad \Gamma, x : \overline{A} \vdash_{\text{ML}} N : B}{\Gamma \vdash_{\text{ML}} \mathit{let}\ x = M\ \texttt{in}\ N : B}\ (\mathit{let})$$

$$\frac{\Gamma, x : A \vdash_{\text{ML}} M : B}{\Gamma \vdash_{\text{ML}} \lambda x.M : A \to B}\ (\to\mathcal{I}) \qquad \frac{\Gamma \vdash_{\text{ML}} M : A \to B \quad \Gamma \vdash_{\text{ML}} N : A}{\Gamma \vdash_{\text{ML}} M\ N : B}\ (\to\mathcal{E})$$

$$\frac{\Gamma \vdash_{\text{ML}} M : \overline{A}}{\Gamma \vdash_{\text{ML}} M : \forall X.\overline{A}[X/\varphi]}\ (\forall\mathcal{I})^{*} \qquad \frac{\Gamma \vdash_{\text{ML}} M : \forall X.\overline{A}}{\Gamma \vdash_{\text{ML}} M : \overline{A}[B/X]}\ (\forall\mathcal{E})$$

* *if φ is not free in Γ.*

As a standard example, consider the term $(\lambda z.zz)(\lambda y.y)$. This remains untypeable in ML, just as it is in the λ-calculus, because z is bound in a lambda-abstraction over the self application zz. The self application requires z to be given two types, of the form $A \to B$ and A (for some Curry types A and B), whereas the lambda abstraction forces z to take a unique Curry type. Using the *let* construct, it is possible to form the term $\mathit{let}\ z = \lambda y.y\ \texttt{in}\ zz$, which will reduce in the same way as our original term. However, it is typeable in the ML system, because the type $\forall X.(X \to X)$ is derivable for $\lambda y.y$, and different instances of this type can be taken for the two occurrences of z (e.g. $(\varphi \to \varphi) \to (\varphi \to \varphi)$ and $(\varphi \to \varphi)$ respectively):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y:\varphi' \vdash_{\mathrm{ML}} y:\varphi'}\,(ax)}{\vdash_{\mathrm{ML}} \lambda y.y:\varphi'\to\varphi'}\,(\to\mathcal{I})}{\vdash_{\mathrm{ML}} \lambda y.y:\forall X.(X\to X)}\,(\forall\mathcal{I}) \quad \dfrac{\dfrac{}{z:\forall X.(X\to X)\vdash_{\mathrm{ML}} z:\forall X.(X\to X)}\,(ax)}{z:\forall X.(X\to X)\vdash_{\mathrm{ML}} z:(\varphi\to\varphi)\to(\varphi\to\varphi)}\,(\forall\mathcal{E}) \quad \dfrac{\dfrac{}{z:\forall X.(X\to X)\vdash_{\mathrm{ML}} z:\forall X.(X\to X)}\,(ax)}{z:\forall X.(X\to X)\vdash_{\mathrm{ML}} z:\varphi\to\varphi}\,(\forall\mathcal{E})}{z:\forall X.(X\to X)\vdash_{\mathrm{ML}} z\,z:\varphi\to\varphi}\,(\to\mathcal{E})}{\vdash_{\mathrm{ML}} \mathit{let}\ z=\lambda y.y\ \mathtt{in}\ z\,z:\varphi\to\varphi}\,(\mathit{let})$$

Although ML admits less polymorphism than System F does, it has the advantage of being very practical: in particular, it has a principal type property. Milner presents an algorithm (called $\mathcal{W}$) that takes as input a pair of basis and term (Γ, M) and returns a pair of substitution and type (S, A), representing the most general typing for the term (if one exists) using an instantiation of the basis.

The formal results concerning the algorithm depend on the following definition (essentially from [21]):

Definition 2.5 (Generic Instance)**.** *A generic type* $\overline{A} = \forall\overrightarrow{X_i}.A$ *has a* generic instance $\overline{B} = \forall\overrightarrow{Y_j}.A'$ *if there exist types* $\overrightarrow{B_i}$ *and atomic types* $\overrightarrow{\varphi_j}$ *such that*[3] $A' = A[\overrightarrow{B_i/X_i}]\,[\overrightarrow{Y_j/\varphi_j}]$*, and* $\varphi_j \notin \overline{A}$.

We write $\overline{A}\geq\overline{B}$ *in this case, read "*$\overline{B}$ *is a generic instance of* $\overline{A}$*".*

Considering the types as logical formulae, in Natural Deduction terms this definition essentially says that $\overline{B}\leq\overline{A}$ if and only if we can derive $\overline{B}$ from $\overline{A}$ using a series of $(\forall\mathcal{E})$ steps, followed by a series of $(\forall\mathcal{I})$ steps. Equivalently in terms of the sequent calculus, $\overline{B}\leq\overline{A}$ if and only if the sequent $\overline{A} \vdash \overline{B}$ is derivable using only the $\forall$-fragment of the logic (i.e., the rules $(\forall\mathcal{I})$,$(\forall\mathcal{E})$ and (ax)). This notion of derivability gives an intuition as to why $\overline{B}$ may be considered 'smaller' or 'less general' than $\overline{A}$.

This is made formal by the following results:

Theorem 2.6 (Properties of the algorithm $\mathcal{W}$)**.**

Soundness: *If* $\mathcal{W}(\langle\Gamma, M\rangle) = \langle S, A\rangle$ *then* $(S\ \Gamma) \vdash_{\mathrm{ML}} M : A$.

Completeness: *If, for a basis* Γ *and term* M*, there exist* S *and* A *such that* $(S\ \Gamma) \vdash_{\mathrm{ML}} M : A$ *then there exist substitutions* S_1 *and* S_2 *and a type* B *such that* $\mathcal{W}(\langle\Gamma, M\rangle) = \langle S_1, B\rangle$ *and* $(S\ \Gamma) = (S_2 \circ S_1\ \Gamma)$ *and* $(S_2 \circ S_1\ \overline{B})\geq\overline{A}$.

[3]Note that the "types" following are not well-formed types, but e.g., A' forms a part of a well-formed type $(\overline{B})$. The equality we write here just means syntactic equality on these portions of types.

2.3. *Principal Types and Principal Typings*

It is worth making clear at this point what we mean by a "principal type property". Wells [22] wrote a paper specifically addressing this point, in which definitions are given for "principal types" and "principal typings". For a type system to have a "principal typing property" there must be an algorithm which, given any term of the syntax, either determines that the term is not typeable at all or else derives *all* of the information used in a typing judgement for the term (other than the term itself), in a *most-general* way. What this information exactly is, and what the notion of 'most general' means depends on the specific calculus and type system. For example, the simply-typed lambda calculus has a principal typing property, for which 'most general' essentially means "can be obtained by applying substitutions and adding extra (redundant) information to the context Γ (weakening)". On the other hand, ML, equipped with the shallow polymorphic type assignment described above, *does not* have a principal typing property [23]. Informally, this essentially is because, given a term M with free variables, it is not possible to determine the most general 'amount' of polymorphism to assume for the types of the free variables. In most cases, the stronger the assumptions made in Γ, the stronger the derived type for M, and vice versa. Instead, ML has a weaker property, which is referred to as a *principal types* property. Essentially, this says that if one *fixes* an initial basis of assumptions Γ, as well as a term M, then one can compute the most general pair of substitution S and generic type $\overline{A}$ (if such a pair exist) such that $(S\ \Gamma) \vdash_{\mathrm{ML}} M : \overline{A}$. Since a substitution S cannot affect the quantified (bound) parts of the types in Γ, it can be understood that the initial Γ determines exactly the polymorphic behaviour which will be assumed for the free variables of M. This is what the algorithm $\mathcal{W}$ achieves.

2.4. *The $\mathcal{X}^i$-calculus*

In this paper we will work largely with the $\mathcal{X}^i$-calculus, which is an untyped term calculus based on a Curry-Howard Correspondence with classical sequent calculus. As such, the reduction rules of the $\mathcal{X}^i$-calculus correspond (in the typeable cases) with the process of *cut elimination* [24]. The calculus is essentially an *untyped* variant of one of the term representations for sequent calculus proofs used in Urban's PhD thesis [8], and the reduction behaviour is that described by Urban's cut elimination. The calculus is named after the $\mathcal{X}$-calculus of van Bakel et. al. [25], but is presented with propagation of cuts as an 'implicit' operation (similar to the treatment of substitution in the λ-calculus). The notation we use is not based on Urban's prefix notation, but rather the infix notation of [25], since

this makes more explicit the input/output symmetry of the calculus, in particular in the case of cuts.

Since the sequents of classical sequent calculus have multiple formulas on *both* sides of the sequent, when defining a term inhabitation for the logic it is natural to have two alphabets of names to index them. We see those names indexing formulas on the left of the sequent as inputs, and call them *sockets*, and those on the right as outputs, and call them *plugs* [25].

Definition 2.7 ($\mathcal{X}^i$-Terms[4])**.** *The terms of the $\mathcal{X}^i$-calculus (ranged over by P,Q,R, etc.) are defined by the following syntax, where x, y range over the infinite set of* sockets *and α, β over the infinite set of* plugs *(sockets and plugs together form the set of* connectors*).*

$$\begin{array}{rcccccc} P, Q & ::= & \langle x.\alpha \rangle & \mid \widehat{x}P\widehat{\alpha}\cdot\beta & \mid P\widehat{\alpha}\,[y]\,\widehat{x}Q & \mid P\widehat{\alpha} \dagger \widehat{x}Q \\ & & \text{capsule} & \text{export} & \text{import} & \text{cut} \end{array}$$

The $\hat{\cdot}$ symbolises that the connector underneath is bound in the attached subterm—a bound socket is written as a prefix to the term, whereas a bound plug is written as a suffix. For example in the import $P\widehat{\beta}\,[y]\,\widehat{x}Q$, occurrences of β are bound in the subterm P and occurrences of x are bound in Q. A connector which does not occur under a binder is said to be free. We will use $fp(P)$ to denote the free plugs of P, and similarly $fs(P)$ for free sockets.

The *capsule* is the basic building-block of the calculus, which consists of a single input-output pair (which may be thought of as two ends of the same connection). An *export* is a construct reminiscent of function abstraction; it binds an input (socket) x and an output (plug) α (whose combination may loosely be considered a function from x to α), and provides this newly-constructed "function" on the output plug β. An *import* is the dual notion to an export, and provides a pair of terms, one with a bound plug, and one with a bound socket. The idea is that an appropriate export term can be inserted between the two terms in an import, providing a way of connecting them together. The *cut* is the active construct in the syntax, which attempts to connect the output plugs named α in P with the input sockets named x in Q. In order to understand the reduction behaviour of

[4] A more-general version of the calculus presented here occurs in the PhD thesis of the author [14], in which the underlying logic includes negation as well as implication as primitive connectives. This allows for completeness in a logical sense, and a more-symmetric computational content. However, for the purposes of this paper, the distinction is not particularly relevant, and we drop negation in order to simplify the presentation.

this calculus[5], and also to aid various discussions later on, it is useful to be able to describe the location of occurrences of a free connector in a term. To this end, we make use of the following definitions:

Definition 2.8 (Exhibiting and Introducing a Connector)**.** *For any $\mathcal{X}^i$-term P and socket x , we say P* exhibits *x if there is an occurrence of x at the top-level of P's syntactic structure (i.e., not located within a subterm of P).*

We say that P introduces *x if $x \in fs(P)$ but, for all proper subterms P' of P, $x \notin fs(P')$ (alternatively, x occurs* uniquely *at the top-level of P's syntactic structure).*

We define exactly the same notions for plugs α instead of sockets x.

For example, $\widehat{x}\langle x.\alpha\rangle\widehat{\beta}\cdot\alpha$ exhibits α but does not introduce α (since there is a further occurrence of α within a subterm). Also, $\langle x.\alpha\rangle$ introduces (and therefore exhibits) both x and α.

The most important use for these definitions is in understanding the behaviour of the cut-elimination procedure. A cut $P\widehat{\alpha} \dagger \widehat{x}Q$ in which P introduces α and Q introduces x can always be removed (and possibly replaced by new cuts between the subterms of P and Q) - this is the ultimate goal of the cut elimination procedure. These rules (which are called the *logical* reduction rules) are described as follows.

Definition 2.9 (Logical Rules)**.** *The logical rules are presented by:*

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

The logical rules are *only* applicable in the special case of a cut whose subterms both introduce the appropriate connector. In all other cases, a cut is reduced by 'seeking out' the positions in its subterms where the appropriate connectors are exhibited. For example, if P does not introduce α, then a cut $P\widehat{\alpha} \dagger \widehat{x}Q$ can be reduced by pushing copies of the cut with Q through the structure of P, depositing a cut at the level of each occurrence of α in P. A similar behaviour is possible

[5]For a more-detailed explanation, please see citevanBakelLengrandLescanne'05,Summers'08.

when x is not introduced in Q. This reduction behaviour is referred to as (*left*- or *right*-)*propagation*.

In contrast to the $\mathcal{X}$-calculus, we present propagation as a meta-operation, external to the calculus itself, in much the same way as substitution is treated in the λ-calculus[6]. We introduce the notation $P\{\alpha \leftrightarrow \widehat{x}Q\}$ to denote the result of left-propagation, which propagates through the structure of the term P, connecting each occurrence of α with a new cut with Q, via x. The notation $Q\{P\widehat{\alpha} \leftrightarrow x\}$ is used for the analogous right-propagation operation. Note that this notation is not a part of the syntax of the calculus; rather it denotes the *result* of evaluating the associated operations. These are defined as follows:

Definition 2.10 (Propagation Operations)**.** *The operation $P\{\alpha \leftrightarrow \widehat{x}Q\}$ is defined recursively over the structure of P, as follows:*

$$
\begin{aligned}
\langle y.\alpha\rangle\{\alpha \leftrightarrow \widehat{x}Q\} &= \langle y.\alpha\rangle\widehat{\alpha} \dagger \widehat{x}Q \\
\langle y.\beta\rangle\{\alpha \leftrightarrow \widehat{x}Q\} &= \langle y.\beta\rangle && \beta \neq \alpha \\
(\widehat{y}P\widehat{\beta}\cdot\alpha)\{\alpha \leftrightarrow \widehat{x}Q\} &= (\widehat{y}(P\{\alpha \leftrightarrow \widehat{x}Q\})\widehat{\beta}\cdot\alpha)\widehat{\alpha} \dagger \widehat{x}Q \\
(\widehat{y}P\widehat{\beta}\cdot\gamma)\{\alpha \leftrightarrow \widehat{x}Q\} &= \widehat{y}(P\{\alpha \leftrightarrow \widehat{x}Q\})\widehat{\beta}\cdot\gamma, && \gamma \neq \alpha \\
(P\widehat{\beta}\,[z]\,\widehat{y}R)\{\alpha \leftrightarrow \widehat{x}Q\} &= (P\{\alpha \leftrightarrow \widehat{x}Q\})\widehat{\beta}\,[z]\,\widehat{y}(R\{\alpha \leftrightarrow \widehat{x}Q\}) \\
(P\widehat{\beta} \dagger \widehat{y}\langle y.\alpha\rangle)\{\alpha \leftrightarrow \widehat{x}Q\} &= (P\{\alpha \leftrightarrow \widehat{x}Q\})\widehat{\beta} \dagger \widehat{x}Q \\
(P\widehat{\beta} \dagger \widehat{y}R)\{\alpha \leftrightarrow \widehat{x}Q\} &= (P\{\alpha \leftrightarrow \widehat{x}Q\})\widehat{\beta} \dagger \widehat{y}(R\{\alpha \leftrightarrow \widehat{x}Q\}), && R \neq \langle y.\alpha\rangle
\end{aligned}
$$

The operation $Q\{P\widehat{\alpha} \leftrightarrow x\}$ is defined recursively over the structure of Q, as follows:

$$
\begin{aligned}
\langle x.\beta\rangle\{P\widehat{\alpha} \leftrightarrow x\} &= P\widehat{\alpha} \dagger \widehat{x}\langle x.\beta\rangle \\
\langle y.\beta\rangle\{P\widehat{\alpha} \leftrightarrow x\} &= \langle y.\beta\rangle, && y \neq x \\
(\widehat{y}Q\widehat{\beta}\cdot\gamma)\{P\widehat{\alpha} \leftrightarrow x\} &= \widehat{y}(Q\{P\widehat{\alpha} \leftrightarrow x\})\widehat{\beta}\cdot\gamma \\
(Q\widehat{\beta}\,[x]\,\widehat{y}R)\{P\widehat{\alpha} \leftrightarrow x\} &= P\widehat{\alpha} \dagger \widehat{x}((Q\{P\widehat{\alpha} \leftrightarrow x\})\widehat{\beta}\,[x]\,\widehat{y}(R\{P\widehat{\alpha} \leftrightarrow x\})) \\
(Q\widehat{\beta}\,[z]\,\widehat{y}R)\{P\widehat{\alpha} \leftrightarrow x\} &= (Q\{P\widehat{\alpha} \leftrightarrow x\})\widehat{\beta}\,[z]\,\widehat{y}(R\{P\widehat{\alpha} \leftrightarrow x\}), \; z \neq x \\
(\langle x.\beta\rangle\widehat{\beta} \dagger \widehat{y}R)\{P\widehat{\alpha} \leftrightarrow x\} &= P\widehat{\alpha} \dagger \widehat{y}(R\{P\widehat{\alpha} \leftrightarrow x\}) \\
(Q\widehat{\beta} \dagger \widehat{y}R)\{P\widehat{\alpha} \leftrightarrow x\} &= (Q\{P\widehat{\alpha} \leftrightarrow x\})\widehat{\beta} \dagger \widehat{y}(R\{P\widehat{\alpha} \leftrightarrow x\}), && Q \neq \langle x.\beta\rangle
\end{aligned}
$$

[6] $\mathcal{X}$ can be seen as the 'explicit' (i.e., propagation is included explicitly in the reduction rules) version of the $\mathcal{X}^i$ calculus, just as $\lambda\mathbf{x}$ can be seen as the 'explicit' version of the λ-calculus. In terms of Urban's work, the $\mathcal{X}^i$ calculus is essentially the untyped version of his $\rightarrow_{aux}$ cut elimination procedure, while $\mathcal{X}$ can be equally compared with the 'localised version', $\rightarrow_{loc}$ [8].

The propagation operations are used to define the two *propagation rules* for this calculus.

Definition 2.11 (Propagation Rules)**.** *We define two* cut-propagation *rules.*

$$(\textit{prop}\text{-L}) : P\widehat{\alpha} \dagger \widehat{x}Q \rightarrow P\{\alpha \leftrightarrow \widehat{x}Q\} \textit{ if } P \textit{ does not introduce } \alpha$$
$$(\textit{prop}\text{-R}) : P\widehat{\alpha} \dagger \widehat{x}Q \rightarrow Q\{P\widehat{\alpha} \leftrightarrow x\} \textit{ if } Q \textit{ does not introduce } x$$

Hereafter, we will write $\rightarrow$ for the reflexive, transitive, compatible reduction relation generated by the logical and propagation rules.

2.4.1. Type Assignment for $\mathcal{X}^i$

Since $\mathcal{X}^i$ is the untyped analogue of a typed term assignment for sequent proofs, it comes with a natural notion of type-assignment. The type system that we present in this section corresponds with a simple sequent calculus for the restriction of classical logic to the two connectives implication ($\rightarrow$) and negation ($\neg$). The sequent calculus on which the type system is based is a variant of Kleene's G3, in which structural rules are treated implicitly. Arbitrary weakenings are allowed at the leaves of a derivation (in the (*ax*) rules), while contraction is treated implicitly per rule; if a statement is introduced to a context in which it is already present, it is simply merged. Gentzen's original formulation also included *exchange* rules, for reordering the statements on the left and right of a sequent; in our setting we treat these collections of statements as (unordered) sets.

Definition 2.12 (Types and Contexts)**.**

1. *The set of* Curry types $\mathcal{T}$*, ranged over by* A, B*, is defined over a set of* atomic types $\mathcal{V} = \{\varphi_1, \varphi_2, \varphi_3, \ldots\}$ *by the grammar:*

$$A, B ::= \varphi \mid A \rightarrow B$$

2. *A* left context Γ *is a partial mapping from sockets to types, denoted as a finite set of* statements $x{:}A$*, such that the* subjects *of the statements (the sockets) are distinct. We write* $\Gamma, x{:}A$ *for* $\Gamma \cup \{x{:}A\}$*. When writing a context as* $\Gamma, x{:}A$*, we indicate that either* Γ *is not defined on* x *or contains the same statement* $x{:}A$*. We write* $\Gamma \backslash x$ *(read as "*Γ without x*") for the context from which the statement concerning* x*, if any, has been removed.*
   Right contexts Δ*, and the notations* $\alpha{:}A, \Delta$ *and* $\Delta \backslash \alpha$ *are defined in a similar way.*

3. *A pair* $\langle \Gamma; \Delta \rangle$ *is usually referred to simply as a* context*, and is a shorthand for the sequent (with labelled formulas)* $\Gamma \vdash \Delta$*. We will sometimes also refer to left/right contexts simply as contexts, when it is clear to do so.*

Armed with these definitions, we can define the simple type assignment system for the calculus.

Definition 2.13 (Typing for $\mathcal{X}^i$)**.**

1. Type judgements *are expressed via a ternary relation* $P :\cdot\ \Gamma \vdash \Delta$*, where* Γ *is a left context,* Δ *is a right context, and* P *is an* $\mathcal{X}^i$*-term. We say that* P *is the* witness *of this judgement.*
2. Type assignment *is defined by the following sequent calculus:*

$$\frac{}{\langle x.\alpha \rangle :\cdot\ \Gamma, x : A \vdash \alpha : A, \Delta}\ (ax) \qquad \frac{P :\cdot\ \Gamma \vdash \alpha : A, \Delta \quad Q :\cdot\ \Gamma, x : A \vdash \Delta}{P\widehat{\alpha} \dagger \widehat{x}Q :\cdot\ \Gamma \vdash \Delta}\ (cut)$$

$$\frac{P :\cdot\ \Gamma, x : A \vdash \alpha : B, \Delta}{\widehat{x}P\widehat{\alpha}\cdot\beta :\cdot\ \Gamma \vdash \beta : A \to B, \Delta}\ (\to\mathcal{R}) \qquad \frac{P :\cdot\ \Gamma \vdash \alpha : A, \Delta \quad Q :\cdot\ \Gamma, y : B \vdash \Delta}{P\widehat{\alpha}\,[x]\widehat{y}Q :\cdot\ \Gamma, x : A \to B \vdash \Delta}\ (\to\mathcal{L})$$

We write $P :\cdot\ \Gamma \vdash \Delta$ *if there exists a derivation using the above rules that has this judgement in the bottom line.*

It is easy to show that a judgement $P :\cdot\ \Gamma \vdash \Delta$ includes types for (at least) the free connectors in P. In terms of the Curry-Howard Correspondence, P represents the syntactic structure of a proof of the sequent $\Gamma \vdash \Delta$, so P is in fact a witness to this sequent being provable in the underlying logic. Note that there is no notion of a type for P itself; rather, the whole context $\langle \Gamma; \Delta \rangle$ describes a consistent way of assigning types to P's connectors.

It is important to emphasise that the typing rules include a notion of implicit contraction; if a new statement is introduced on the bottom line of a rule, but it was already present in the context, then it is simply merged. We do not consider duplicate statements, as we consider contexts to be unordered sets. This also implies that a typing rule cannot be applied if it would result in the addition of a statement $x : A$ to a context Γ, say, in which x was already assigned a different type.

$$
\dfrac{
\dfrac{\dfrac{\dfrac{}{\langle y.\pi\rangle \;\vdots\cdot\; y:\varphi \vdash_{\mathrm{SP}} \pi:\varphi}\,(ax)}{\widehat{y}\langle y.\pi\rangle\widehat{\pi}\cdot\theta \;\vdots\cdot\; \vdash_{\mathrm{SP}} \theta:\varphi\to\varphi}\,(\to\mathcal{R})}{\widehat{y}\langle y.\pi\rangle\widehat{\pi}\cdot\theta \;\vdots\cdot\; \vdash_{\mathrm{SP}} \theta:\forall X.(X\to X)}\,(\forall\mathcal{R})
\qquad
\dfrac{\dfrac{\dfrac{\dfrac{}{\langle x.\gamma\rangle \;\vdots\cdot\; x:A\to A \vdash_{\mathrm{SP}} \gamma:A\to A}\,(ax) \qquad \dfrac{}{\langle p.\alpha\rangle \;\vdots\cdot\; p:A\to A \vdash_{\mathrm{SP}} \alpha:A\to A}\,(ax)}{\langle x.\gamma\rangle\widehat{\gamma}\,[x]\,\widehat{p}\langle p.\alpha\rangle \;\vdots\cdot\; x:(A\to A)\to(A\to A),\, x:A\to A \vdash_{\mathrm{SP}} \alpha:A\to A}\,(\to\mathcal{L})}{\langle x.\gamma\rangle\widehat{\gamma}\,[x]\,\widehat{p}\langle p.\alpha\rangle \;\vdots\cdot\; x:(A\to A)\to(A\to A),\, x:\forall X.(X\to X) \vdash_{\mathrm{SP}} \alpha:A\to A}\,(\forall\mathcal{L})}{\langle x.\gamma\rangle\widehat{\gamma}\,[x]\,\widehat{p}\langle p.\alpha\rangle \;\vdots\cdot\; x:\forall X.(X\to X) \vdash_{\mathrm{SP}} \alpha:A\to A}\,(\forall\mathcal{L})
}{
(\widehat{y}\langle y.\pi\rangle\widehat{\pi}\cdot\theta)\widehat{\theta} \dagger \widehat{x}(\langle x.\gamma\rangle\widehat{\gamma}\,[x]\,\widehat{p}\langle p.\alpha\rangle) \;\vdots\cdot\; \vdash_{\mathrm{SP}} \alpha:A\to A
}\,(cut)
$$

Figure 1: Example of shallow-polymorphic type assignment in $\mathcal{X}^i$

Example 2.14. *If the judgement $P \;\vdots\cdot\; x:A \vdash \alpha:B,\beta:A$ had been derived, and one wished to apply the $(\to\mathcal{R})$ rule to this statement, binding the connectors x and α, it would not be possible for the connector exhibited in the premise to be β, since this would mean β was assigned both type A and type $A\to B$. Put more succinctly, the $\mathcal{X}^i$-term $\widehat{x}\langle x.\beta\rangle\widehat{\alpha}\cdot\beta$ is not typeable in the type system presented above.*

We have the following result for the simple type system:

Theorem 2.15 (Witness Reduction)**.** *If $P \;\vdots\cdot\; \Gamma \vdash \Delta$, and $P \to Q$, then $Q \;\vdots\cdot\; \Gamma \vdash \Delta$.*

Proof. $\mathcal{X}^i$-terms to which types have been assigned correspond to sequent proofs and can be equivalently represented in the term calculus of Urban; this result then follows from the soundness of the cut elimination procedure of Urban [8]. □

3. Universal Shallow Polymorphism for $\mathcal{X}^i$

3.1. The Intuitive, Unsound Approach

The key to the use of polymorphism in ML is in the *let* construct, which is interpreted as a substitution both syntactically (according to its reduction rule) and semantically (see [18]). The polymorphism present in the (*let*)-rule essentially gives a way of typing the substitution about to take place such that multiple occurrences of the name to replace need not all be typed in the same way. The *let*-construct is a necessary extension to the syntax for a shallow polymorphic approach (short of allowing polymorphism to be used directly with abstractions and applications, which leads to System F), since there is nothing in the syntax of the λ-calculus to represent these substitutions.

In the $\mathcal{X}^i$-calculus, there is a construct already present which can be seen to encode substitution. The cut $P\widehat{\alpha} \dagger \widehat{x}Q$ can, by right-evaluation, approximately simulate the substitution of P for the occurrences of x in Q. This observation led to

the investigation of a notion of shallow polymorphic type-assignment for the $\mathcal{X}^i$-calculus. Following what seems to be the analogous approach to ML, one adds generic types to the type language, which are allowed to be used for the typing of cuts and axioms (but not the other syntax constructs), and the standard logical rules for $\forall$ (this time for the sequent calculus) are added to the type assignment rules.

Definition 3.1 (Naïve Shallow Polymorphic Type Assignment for $\mathcal{X}^i$ [15])**.** *Types A, B and type-schemes $\overline{A}$ are defined as follows:*

$$\begin{aligned} A, B &::= \varphi \mid X \mid (A \to B) \\ \overline{A} &::= \forall X_1.\forall X_2.\ldots.\forall X_n.A \quad (n \geq 0) \end{aligned}$$

The shallow polymorphic type assignment for $\mathcal{X}^i$ is defined by the following rules (where $\overline{A}$ represents a generic type of Definition 2.3):

$$\frac{}{\langle x.\alpha\rangle \;\vdots\cdot\; \Gamma, x:\overline{A} \vdash_{\text{NSP}} \alpha:\overline{A}, \Delta}\,(ax) \qquad \frac{P \;\vdots\cdot\; \Gamma \vdash_{\text{NSP}} \alpha:\overline{A}, \Delta \quad Q \;\vdots\cdot\; \Gamma, x:\overline{A} \vdash_{\text{NSP}} \Delta}{P\widehat{\alpha} \dagger \widehat{x}Q \;\vdots\cdot\; \Gamma \vdash_{\text{NSP}} \Delta}\,(cut)^1$$

$$\frac{P \;\vdots\cdot\; \Gamma \vdash_{\text{NSP}} \alpha:A, \Delta \quad Q \;\vdots\cdot\; \Gamma, x:B \vdash_{\text{NSP}} \Delta}{P\widehat{\alpha}[y]\widehat{x}Q \;\vdots\cdot\; \Gamma, y:A{\to}B \vdash_{\text{NSP}} \Delta}\,(\to\mathcal{L})^1 \qquad \frac{P \;\vdots\cdot\; \Gamma, x:A \vdash_{\text{NSP}} \alpha:B, \Delta}{\widehat{x}P\widehat{\alpha}\cdot\beta \;\vdots\cdot\; \Gamma \vdash_{\text{NSP}} \beta:A{\to}B}\,(\to\mathcal{R})^1$$

$$\frac{P \;\vdots\cdot\; \Gamma, x:\overline{A}[B/X] \vdash_{\text{NSP}} \Delta}{P \;\vdots\cdot\; \Gamma, x:\forall X.\overline{A} \vdash_{\text{NSP}} \Delta}\,(\forall\mathcal{L}) \qquad \frac{P \;\vdots\cdot\; \Gamma \vdash_{\text{NSP}} \alpha:\overline{A}, \Delta}{P \;\vdots\cdot\; \Gamma \vdash_{\text{NSP}} \alpha:\forall X.\overline{A}[X/\varphi], \Delta}\,(\forall\mathcal{R})^2$$

[1]*: if $x \notin \Gamma$ and $\alpha \notin \Delta$.* [2]*: if φ does not occur in Γ, Δ.*

We include a notion of implicit contraction in the above rules, so that if a derivation rule introduces a statement which was already present in the context, it is simply merged.

Notice that generic types are not used in the $(\to\mathcal{R})$ or $(\to\mathcal{L})$ rules. This enforces the restriction that the $\forall$-symbol may not appear underneath an '$\to$' in a type, and is similar to the way the $(\to I)$ and $(\to E)$ rules are treated in ML.

A subtle problem occurs in defining a shallow polymorphic type assignment in this way, which suggests a possible relaxation of Definition 2.12 to allow multiple statements in a context with the same subject. The mechanism for taking instances of a generic type employs the $(\forall\mathcal{L})$ rule, which can be seen to allow instances of the $\forall$ formula to be taken further up in a derivation. However, because the instance $\overline{A}[B/X]$ appears also on the left-hand side of the sequent, and is labelled with the same name (socket), this eliminates the possibility of further instances

being taken further up in the same 'branch' of the derivation - the statement $\forall X.\overline{A}$ may not remain in the upper sequent of the rule, since we insist in Definition 2.12 that the *subjects* of the statements in a context are distinct. Thinking in terms of the logical proofs however, the subjects of the statements are not a consideration - sequent proofs need not always be annotated (depending on the presentation of the logic) and would certainly allow a use of the $(\forall\mathcal{L})$ rule to include an implicit contraction. For example, in the following proof thel formula $\forall X.(X{\to}X)$ is used in two $(\forall\mathcal{L})$ rules:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(A{\to}A) \vdash (A{\to}A)}\,(\mathrm{Ax}) \qquad \dfrac{}{(A{\to}A) \vdash (A{\to}A)}\,(\mathrm{Ax})}{(A{\to}A), (A{\to}A){\to}(A{\to}A) \vdash A{\to}A}\,(\to\mathcal{L})}{\forall X.(X{\to}X), (A{\to}A){\to}(A{\to}A) \vdash A{\to}A}\,(\forall\mathcal{L})}{\forall X.(X{\to}X) \vdash A{\to}A}\,(\forall\mathcal{L})$$

This might correspond to a type derivation in a shallow polymorphic system, (where we use B as a shorthand for the formula $(A{\to}A)$), as in the following derivation:

Example 3.2.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\langle x.\alpha\rangle :\cdot\ x : B \vdash \alpha : B}\,(\mathrm{Ax}) \qquad \dfrac{}{\langle y.\beta\rangle :\cdot\ y : B \vdash \beta : B}\,(\mathrm{Ax})}{\langle x.\alpha\rangle\widehat{\alpha}\,[x]\,\widehat{y}\langle y.\beta\rangle :\cdot\ x : B, x : B{\to}B \vdash \beta : B}\,(\to\mathcal{L})}{\langle x.\alpha\rangle\widehat{\alpha}\,[x]\,\widehat{y}\langle y.\beta\rangle :\cdot\ x : \forall X.(X{\to}X), x : B{\to}B \vdash \beta : B}\,(\forall\mathcal{L})}{\langle x.\alpha\rangle\widehat{\alpha}\,[x]\,\widehat{y}\langle y.\beta\rangle :\cdot\ x : \forall X.(X{\to}X) \vdash \beta : B}\,(\forall\mathcal{L})$$

This is a type derivation we would like to be legal in this system, since we can view this as part of the type derivation for a term analogous to *let* $x = \lambda y.y$ `in` xx, which we wish to be able to type (c.f. Figure 1). It is possible to work around this problem, by adjusting the set of rules so that instances can be taken implicitly of a quantified formula. In fact, this solution will be employed in the next section, for reasons which will become clear. However, for the moment we explore a more basic solution, which yields a type-system whose underlying derivations are still standard logical proofs.

To deal with the problem of instantiating quantified types in this system, we initially considered relaxing Definition 2.12, allowing multiple statements in a

context with the same subject. This seems at first glance a risky move, but hopefully the example above has shown that it allows intuitively sound derivations to be constructed. In order to retain soundness, we needed to be careful that whenever a connector (plus/socket) is bound, some statements involving the connector do not remain in the context. We therefore insisted that whenever the rules ($\rightarrow\mathcal{R}$), ($\rightarrow\mathcal{L}$) and (*cut*) were employed, the connectors mentioned in the top line of the rule (which are bound in the construction of the respective terms) had a unique statement in the rule. This enforces that all the types for a connector disappear from the contexts when the connector is bound. We also insisted that a derivation is not complete unless the subjects of the statements in the *final* sequent are unique (so the relaxation is only usable temporarily within a derivation). As a consequence of these restrictions, if several statements with the same subject (but different types) are used in a derivation, it will be necessary for the $\forall$ rules to be applied until the types of these statements match, and they are contracted into a single statement. Until this takes place, it will be impossible to either bind the connector (plug/socket) concerned, or complete the derivation.

This is the type system which was presented in [15], in which a notion of principal contexts (with respect to an initial context) was also defined, in the spirit of the principal types property for ML. As we shall explain next, while this type system seems in many ways analagous to the way polymorphism is introduced to ML, in our more general setting (and particularly in the presence of classical logic), this approach is unsound.

3.1.1. Failure of Subject Reduction

Unfortunately, the 'intuitive' approach outlined in the previous section does not guarantee subject reduction (although it was originally believed to do so [15]). The problem is due to the interaction between the use of implicit (i.e., not represented syntactically in the calculus) polymorphism in the type derivation, and the ability to perform left propagation reductions. In particular, since the implicit quantifier rules can occur at any point in a derivation, a cut may be left-propagated 'through' an occurrence of the ($\forall\mathcal{R}$) rule used to type the left-hand subterm. In order to construct a new type derivation for the resulting term, we need to be able to 'relocate' the occurrence of the ($\forall\mathcal{R}$) rule, to be applied further up on the derivation. This is not always possible, because the side-condition of the rule is not always satisfied in this new position. We can make this clearer with an example.

Example 3.3 (Failure of Subject Reduction)**.** *Define* $P = \widehat{x}(\widehat{y}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma)\widehat{\beta}\cdot\gamma$. *This term can be assigned the same contexts as the identity, in the type system presented*

above:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\langle x.\alpha\rangle \;\vdots\cdot\; x:\varphi, y:\varphi \vdash_{\text{NSP}} \alpha:\varphi, \beta:\varphi}\,(ax)}{\widehat{y}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma \;\vdots\cdot\; x:\varphi \vdash_{\text{NSP}} \beta:\varphi, \gamma:\varphi\to\varphi}\,(\to\mathcal{R})}{\widehat{x}(\widehat{y}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma)\widehat{\beta}\cdot\gamma \;\vdots\cdot\; \emptyset \vdash_{\text{NSP}} \gamma:\varphi\to\varphi}\,(\to\mathcal{R})}{\widehat{x}(\widehat{y}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma)\widehat{\beta}\cdot\gamma \;\vdots\cdot\; \emptyset \vdash_{\text{NSP}} \gamma:\forall X.(X\to X)}\,(\forall\mathcal{R})$$

Therefore, if we place this term in a cut which 'applies it to itself' (i.e. in an ML sense, we construct let $z = P$ `in` $z\ z$*, cf. Example 3.2), then the resulting term can be typed as follows:*

$$\dfrac{\overbrace{P \;\vdots\cdot\; \emptyset \vdash_{\text{NSP}} \gamma:\forall X.(X\to X)}^{\textit{as above}} \quad \overbrace{\langle z.\delta\rangle\widehat{\delta}\,[z]\,\widehat{w}\langle w.\epsilon\rangle \;\vdots\cdot\; z:\forall X.(X\to X) \vdash_{\text{NSP}} \epsilon:\varphi'\to\varphi'}^{\textit{Example 3.2}}}{P\widehat{\gamma} \dagger \widehat{z}(\langle z.\delta\rangle\widehat{\delta}\,[z]\,\widehat{w}\langle w.\epsilon\rangle) \;\vdots\cdot\; \emptyset \vdash_{\text{NSP}} \epsilon:\varphi'\to\varphi'}\,(cut)$$

However, this term can be shown to reduce as follows:

$$\begin{aligned}
&(\widehat{x}(\widehat{y}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma)\widehat{\beta}\cdot\gamma)\widehat{\gamma} \dagger \widehat{z}(\langle z.\delta\rangle\widehat{\delta}\,[z]\,\widehat{w}\langle w.\epsilon\rangle)\\
\to\ &(\widehat{x}((\widehat{y}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma)\widehat{\gamma} \dagger \widehat{z}(\langle z.\delta\rangle\widehat{\delta}\,[z]\,\widehat{w}\langle w.\epsilon\rangle))\widehat{\beta}\cdot\gamma)\widehat{\gamma} \dagger \widehat{z}(\langle z.\delta\rangle\widehat{\delta}\,[z]\,\widehat{w}\langle w.\epsilon\rangle) \quad (\textit{prop}\text{-L})\\
\to\ &\widehat{x}\langle x.\epsilon\rangle\widehat{\beta}\cdot\epsilon
\end{aligned}$$

The resulting term is not typeable in this system. In fact, the problem came right in the first step, when the cut was propagated to the left, through the structure of the term P. In the typing derivation for P, the crucial $(\forall\mathcal{R})$ *rule comes right at the very end. But, when propagating a copy of this cut inside the structure of P, in order to maintain the same quantified type for the new cut there must be a similar occurrence of the* $(\forall\mathcal{R})$ *rule on this copy; i.e., the rule needs to be moved upwards in the derivation with the cut. This is not possible; at this point x is still a free socket in the context, carrying the type* φ *which is to be generalised by the* $(\forall\mathcal{R})$ *rule.*

In short, the condition on the $(\forall\mathcal{R})$ *rule is not necessarily preserved by moving it further up the typing derivation, and so, when cuts are left-propagated 'past' an occurrence of the* $(\forall\mathcal{R})$ *rule, it is not always possible to rebuild the same quantifier rule in a suitable new position. In general, this means that a type derivation cannot always be reconstructed.*

With hindsight, the failure of subject reduction is not that surprising. It is well-known that the original ML approach to polymorphism is unsound in the presence

of various extensions to the language, such as references, exceptions and the control operator call/cc [26]. Calculi based on classical logic can also be closely related to functional calculi extended with control operators, and we believe that (for example), the version of ML with call/cc included could also be encoded into the $\mathcal{X}^i$-calculus. Therefore, the polymorphic type-system presented above must almost inevitably be unsound. Furthermore, Fujita has shown that a similar unsoundness arises in the context of an alternative calculus based on classical natural deduction [27]. However, we believe that the source of the unsoundness is actually much clearer in the sequent calculus setting: it is clear that the attempted left propagation of a cut 'past' an occurrence of $(\forall\mathcal{R})$ in the left-hand typing derivation is the exact source of the problem. In fact, we can describe the essence of this problem by observing that the presence of the following three aspects will guarantee such an unsoundness:

1. Implicit universal quantification.
2. Call-by-value reductions (not necessarily *only* these reductions, but their inclusion in the calculus).
3. Ability to express/encode classical structural rules (e.g., contraction) manipulating statements on the right of a typing sequent.

Our counter-example depends on the presence of these three features. Implicit quantification allows reduction to 'ignore' the quantifier steps which are violated in the example. Left-propagation of a cut which could be right-propagated (i.e., a call-by-value reduction) ensures that such a violation cannot be 'fixed' in the reduct (i.e., there is in general no way of typing the reduct by, for example, resorting to non-quantified types). Finally, (implicit) right-contraction in the typing is used to cause the failure of the side-condition on the $(\forall\mathcal{R})$ after left-propagation is performed.

It is interesting to note that examples exist in the literature of proposed calculi and type systems which include each possible pair of *two* out of the three ingredients for unsoundness described. The ML calculus has implicit universal quantification, and call-by-value reductions, but no classical logic features such as right-contraction in the type system. Parigot's presentation of the $\lambda\mu$-calculus in [7] includes implicit universal quantification, and the ability to (indirectly) express right-contraction in the type system, however the reduction rules are essentially restricted to call-by-name reductions. Ong and Stewart's definition of call-by-value $\lambda\mu$-calculus [12] includes call-by-value reductions, and still permits right-contraction to be expressed in the type system, but no rules for polymor-

phism are included. Therefore, in each of these works, one of the three 'ingredients' described above is missing, and so the unsoundness we are concerned with is avoided.

There are three main approaches described in the literature for dealing with this unsoundness in the context of ML:

1. Introducing a separate class of ('imperative') atomic types [28], which must be used whenever an 'imperative' feature such as call/cc is to be typed, and may not be generalised using the $(\forall\mathcal{R})$ rule. In our setting it is less obvious how to understand this solution, but it amounts essentially to permitting polymorphic types only on cuts where the left-hand subterm satisfies certain properties (we conjecture that these properties amount to the subterm representing a proof valid in *minimal* logic, but this idea is not explored here).
2. Restricting reductions to a call-by-name strategy [29]. It turns out that the problematic cases cannot be reached by call-by-name reductions. The reason for this can be fairly clearly seen in the context of the $\mathcal{X}^i$-calculus and the counter-example presented above; restricting to call-by-name amounts to insisting that cuts be propagated preferentially to the right, and only left-propagated if the socket bound in the right-hand subterm is introduced. In such a situation, any cut which can be typed with a quantified type may also be typed without quantification; this is because the uniqueness of the socket means that the ability to take multiple instances of the quantified type is irrelevant. Therefore, by the time a cut is left-propagated, we may depend essentially on the subject reduction property of the simple type system.
3. Restricting the form of let-bound terms to bind only values [30] (i.e., only allow terms of the form $\mathit{let}\ x = V\ \mathtt{in}\ M$. Again, the soundness of this approach can be seen clearly in our setting; restricting to values here amounts to restricting the left-hand subterm of cuts $P\widehat{\alpha} \dagger \widehat{x}Q$ to the cases where P introduces α. In such a system, no left-propagation reductions can ever take place, and so the problematic scenario is avoided.

Unfortunately, given that our original aim was to eventually define a type system in which both existential and universal quantification could be employed, none of the solutions above seem very desirable. We will explain why this is so for each in turn:

1. This approach breaks the logical foundation of the type system, and, while practical in the ML setting, it is not clear how it would be adapted to deal with existential quantification, and whether a useful system would result.

2. Unfortunately, just as a call-by-name reduction strategy is required to make implicit universal quantification safe, a call-by-value strategy would be needed to ensure subject reduction for a system with similar existential polymorphism. Thus, no reduction strategy would work for a system with both kinds of polymorphism.
3. Similarly, in order to ensure that subject reduction held for a system with both kinds of quantifiers, one would need to restrict the system to allow both kinds of quantifiers in the typing of cuts $P\widehat{\alpha} \dagger \widehat{x}Q$ only in the case when both P introduces α and Q introduces x. Ensuring this condition was met and preserved by reduction would result in a system with almost no useful polymorphism - these cuts can be typed just as well in the simple (non-polymorphic) type system.

By examining the problematic cases in more detail, we were able to come up with a fourth solution (which is in fact, a generalisation of the restriction to values, above). This is, to restrict the points in a derivation where polymorphic generalisation (i.e., the $(\forall\mathcal{R})$ rule in the previous type system) may be employed. In order to avoid the unsoundness described, we only allow generalisation of a statement immediately when it is introduced into the derivation. For example, when the $(\rightarrow\mathcal{R})$ rule is applied, the type for the exhibited plug may be generalised, but if it is not, then it cannot be later on in the derivation. The advantages of our solution are that it imposes fewer restrictions on the type system than the restriction to values (more terms are typeable), and that it does not eliminate in principle the possibility of a useful extended system based on both existential and universal quantification.

The observation we have gained from the sequent calculus setting is that the unsoundness of the naïve system is directly caused by the left-propagation of cuts $P\widehat{\alpha} \dagger \widehat{x}Q$ past occurrences of the $(\forall\mathcal{R})$ rule in the typing derivation for P. We note that this can only happen because, in general, it is allowed for such occurrences to exist in positions far devoid from the points in the derivation (and term) where occurrences of α are exhibited. Since it is these points which the left-propagation of the cut reaches, the cut can of course pass over occurrences of the $(\forall\mathcal{R})$ rule on the way. The third solution listed above ([30]) can be understood then as removing the possibility of such 'gaps' between the occurrences of α and the occurrences of $(\forall\mathcal{R})$ applied to the type of α; by insisting on the strong requirement that P introduces α, i.e., that there is exactly one occurrence of α in P, and that it is at the top-level, any $(\forall\mathcal{R})$ rules to be used in typing the cut must also occur at this top level. However, we observe that it would suffice to guarantee the weaker property,

that there are no 'gaps' between each occurrence of α and the polymorphic rules applied to the type for the occurrence; this guarantees that a cut 'seeking out' the occurrences of α need never cross over such rules.

Consider the following $\mathcal{X}^i$-term, for example (where the subterm P is left unspecified):

$$((\widehat{x}\langle x.\alpha\rangle\widehat{\alpha}\cdot\beta)\widehat{\epsilon}\,[i]\;\widehat{j}(\widehat{x}\langle x.\alpha\rangle\widehat{\alpha}\cdot\beta))\widehat{\beta}\,\dagger\,\widehat{z}P$$

The left-hand subterm of the cut contains two copies of the identity function $\widehat{x}\langle x.\alpha\rangle\widehat{\alpha}\cdot\beta$, both of which exhibit occurrences of the output β (the other names within the terms are also identical, but since these are bound it is only for comparison). The two 'copies' of the identity are independent of one another (the surrounding import does not bind any plugs/sockets in the subterms, and acts as a 'dummy context' for this example). Since each copy can be given the polymorphic type $\forall X.(X\rightarrow X)$, it seems reasonable for the cut to employ this type, also. Furthermore, since the $(\forall\mathcal{R})$ rule applications needed to derive the type $\forall X.(X\rightarrow X)$ can be located at each of the points where β is exhibited, there is no need to risk the possibility of the cut 'crossing' these rules by left-propagation. Essentially, if the polymorphic generalisation steps in a derivation can be located at the same syntactic level as the connector whose type they apply to is exhibited, the derivation is safe from the potential unsoundness described above. This notion is tricky to formalise in a type system with rules for manipulating quantifiers independent of the other rules in the type system (e.g., the $(\forall\mathcal{L})$ and $(\forall\mathcal{R})$ rules in the system presented above). However, since we are now proposing that such rules be employed only at the points where the corresponding connectors are introduced, we can instead present a system with the polymorphism steps 'built in' to the other rules. This will be presented next.

3.2. An Improved Shallow Polymorphic Type System

We write *typeof* $x\,\Gamma$ and *typeof* $\alpha\,\Delta$ to denote functions which look up the type assigned to the connector by the context, and if none is defined, return a fresh atomic type. For example, if $\Gamma = \{x:\overline{A}, y:B\}$ then *typeof* $x\,\Gamma = \overline{A}$, while, for $y \neq z \neq x$, *typeof* $z\,\Gamma = \varphi$ for some fresh atomic type φ.

We now extend Definition 2.5 to allow the comparison of (right) contexts as follows:

Definition 3.4 (Generic Instance for Contexts)**.** *We extend the notion of generic instance to (right)-contexts* Δ_1,Δ_2 *as follows:* $\Delta_1 \geq \Delta_2 \quad \Leftrightarrow \quad (\alpha \in \Delta_1 \Rightarrow \alpha \in \Delta_2$ & (*typeof* $\alpha\,\Delta_1) \geq$ (*typeof* $\alpha\,\Delta_2$).

It is also useful to have an explicit notation for a 'closure' relation on types, which characterises the behaviour of the $\forall\mathcal{R}$ rule. This rule can be used to replace types with more general (larger, in the $\geq$ relation) forms, provided this is sound with respect to the context in which it is used. Thus this relation depends not only on the types which are changed, but also on the types present in the rest of the context (c.f. the condition on the $\forall\mathcal{R}$ rule). We introduce a relation on types which coincides with any number of valid $\forall\mathcal{R}$ steps being applied to the same statement $\alpha : \overline{A}$ say, in a context $\langle\Gamma;\Delta\rangle$.

Definition 3.5 (Closures and fresh instances)**.** 1. *For any generic types $\overline{A}, \overline{B}$ and context $\langle\Gamma;\Delta\rangle$, we say $\overline{A}$* closes to *$\overline{B}$* in *$\langle\Gamma;\Delta\rangle$, and write $\overline{A} \lhd_{\langle\Gamma;\Delta\rangle} \overline{B}$, if and only if there exist $\overrightarrow{X_i}$ and $\overrightarrow{\varphi_i}$ such that $\overline{B} = \forall\overrightarrow{X_i}.\overline{A}[\overrightarrow{X_i/\varphi_i}]$, where $\varphi_i \notin \langle\Gamma;\Delta\rangle$ and $\varphi_i \notin \overline{B}$.*

2. *For any generic type $\overline{A} = \overrightarrow{\forall X_i}.A$, we define $freshInst(\overline{A}) = A[\overrightarrow{\varphi_i/X_i}]$ where the $\overrightarrow{\varphi_i}$ are fresh atomic types.*

We have the following results for these definitions:

Proposition 3.6. 1. *$\leq$ is a preorder on generic types.*

2. *For any contexts $\langle\Gamma;\Delta\rangle$, $\lhd_{\langle\Gamma;\Delta\rangle}$ is a partial order on generic types.*
3. *For any contexts $\langle\Gamma;\Delta\rangle$ and generic types $\overline{A}, \overline{B}, \overline{C}$, if $\overline{A} \geq \overline{B}$ and $\overline{B} \lhd_{\langle\Gamma;\Delta\rangle} \overline{C}$ and $\overline{A}$ is the type for some connector in $\langle\Gamma;\Delta\rangle$, then $\overline{A} \geq \overline{C}$.*
4. *For any generic types $\overline{A}, \overline{B}$ and substitution S, if $\overline{A} \geq \overline{B}$ then $(S\ \overline{A}) \geq (S\ \overline{B})$.*
5. *For any generic types $\overline{A}, \overline{B}$, context $\langle\Gamma;\Delta\rangle$ and substitution S, if $\overline{A} \lhd_{\langle\Gamma;\Delta\rangle} \overline{B}$ then there exists a substitution S' such that $dom(S') \subseteq (atoms(\overline{A}) \backslash atoms(\langle\Gamma;\Delta\rangle))$ and $(S'\ \overline{B}) = \overline{B}$ and $(S \circ S'\ \overline{A}) \lhd_{\langle(S\ \Gamma);(S\ \Delta)\rangle} (S\ \overline{B})$.*
6. *For any type A, generic types $\overline{A}$ and $\overline{B}$, and context $\langle\Gamma;\Delta\rangle$, if $A \lhd_{\langle\Gamma;\Delta\rangle} \overline{A}$ and $\overline{A} \geq \overline{B}$ then there is a substitution S with $dom(S) \subseteq (atoms(A) \backslash atoms(\langle\Gamma;\Delta\rangle))$ and $(S\ A) \lhd_{\langle\Gamma;\Delta\rangle} \overline{B}$.*
7. *For any type $\overline{A} = \overrightarrow{\forall X_i}.A$ and Curry type B, if $A' = freshInst(\overline{A}) = A[\overrightarrow{\varphi_i/X_i}]$ and $\overline{A} \geq B$ then there exists a substitution S such that $dom(S) = \{\overrightarrow{\varphi_i}\}$ and $(S\ A') = B$.*

Proof. See Proof B.1 in Section B. □

We are now able to define our amended type system. The essence of the system is to "bake in"[7] the quantifier rules at the points where the corresponding

[7] Thanks to Gavin Bierman for this terminology!

connectors are exhibited in the derivation. This avoids the possibility for cuts to propagate through the quantifier rules, since they occur exactly at the points at which cuts are deposited and propagation halts[8].

Definition 3.7 (Improved Shallow Polymorphic Type Assignment for $\mathcal{X}^i$)**.** *The (sound) shallow polymorphic type assignment for $\mathcal{X}^i$ is defined by the following rules (where $\overline{A}$ represents a generic type of Definition 2.3):*

$$\frac{}{\langle x.\alpha\rangle \;\vdots\cdot\; \Gamma, x:\overline{A} \vdash_{\mathrm{SP}} \alpha:\overline{B}, \Delta}\,(ax)^1 \qquad \frac{P \;\vdots\cdot\; \Gamma, x:A \vdash_{\mathrm{SP}} \alpha:B, \Delta}{\widehat{x}P\widehat{\alpha}\cdot\beta \;\vdots\cdot\; \Gamma \vdash_{\mathrm{SP}} \beta:\overline{C}, \Delta}\,(\rightarrow\mathcal{R})^2$$

$$\frac{P \;\vdots\cdot\; \Gamma \vdash_{\mathrm{SP}} \alpha:A, \Delta \quad Q \;\vdots\cdot\; \Gamma, y:B \vdash_{\mathrm{SP}} \Delta}{P\widehat{\alpha}\,[x]\widehat{y}Q \;\vdots\cdot\; \Gamma, x:\overline{C} \vdash_{\mathrm{SP}} \Delta}\,(\rightarrow\mathcal{L})^3 \quad \frac{P \;\vdots\cdot\; \Gamma \vdash_{\mathrm{SP}} \alpha:\overline{A}, \Delta \quad Q \;\vdots\cdot\; \Gamma, x:\overline{A} \vdash_{\mathrm{SP}} \Delta}{P\widehat{\alpha} \dagger \widehat{x}Q \;\vdots\cdot\; \Gamma \vdash_{\mathrm{SP}} \Delta}\,(cut)$$

[1] $\overline{A} \geq \overline{B}$. [2] $(A \rightarrow B) \lhd_{\langle\Gamma;\Delta\rangle} \overline{C}$. [3] $\overline{C} \geq (A \rightarrow B)$.

In comparison with the previous (unsound) proposal, this type system can be seen as a restriction in which the (now implicit) uses of quantifier rules, which could previously occur at any apparently valid point in a type derivation, are now restricted to be applied in precise positions. In fact, all such quantifier rules are implicitly applied immediately after the statement which they affect is introduced into the context. For example, an occurrence of the $(\forall\mathcal{L})$ rule in the naïve type system, which bound a statement originally introduced by an occurrence of the $(\rightarrow\mathcal{L})$ rule, is (in the new type system) implicitly included in the new version of the $(\rightarrow\mathcal{L})$ rule, by allowing the type for x in the premise to be a generic instance of the type for x in the conclusion of the rule. This essentially permits any number of implicit applications of the $(\forall\mathcal{L})$ rule here.

Note that in the new type system, the problematic term from Example 3.3 can no longer be typed. This is because the term $P = \widehat{x}(\widehat{y}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma)\widehat{\beta}\cdot\gamma$ can no-longer be typed with a polymorphic type for γ. Any type derivation for this term must

[8]Note that there exist presentations of ML which "bake in" the quantifier rules as well. However, the analogous system in our work would bake all quantifier rules in at the point of a cut occurrence in a derivation. This does not avoid the potential unsoundness of cuts being propagated past these occurrences (and indeed, does not affect the natural unsoundness of the ML type system in the presence of control operators [26].

have the following form:

$$\dfrac{\dfrac{\dfrac{}{\langle x.\alpha\rangle \;:\cdot\; \Gamma, y:C, x:A \vdash_{\mathrm{SP}} \alpha:B, \beta:D, \Delta}\,(ax)}{\widehat{y}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma \;:\cdot\; \Gamma, x:A \vdash_{\mathrm{SP}} \gamma:\overline{E}, \beta:D, \Delta}\,(\rightarrow\mathcal{R})}{\widehat{x}(\widehat{y}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma)\widehat{\beta}\cdot\gamma \;:\cdot\; \Gamma \vdash_{\mathrm{SP}} \gamma:\overline{E}, \Delta}\,(\rightarrow\mathcal{R})$$

where $A{\geq}B$ and $C{\rightarrow}A \lhd_{\langle\Gamma,x:A;\beta:C,\Delta\rangle} \overline{E}$ and also $A{\rightarrow}D \lhd_{\langle\Gamma;\gamma:\overline{E},\Delta\rangle} \overline{E}$. Since A and B are Curry types, $A{\geq}B$ means that A and B are the same. Since $C{\rightarrow}A$ is "closed to $\overline{E}$" in a context in which A occurs, none of the atomic types in A may be closed, and so $\overline{E}$ must be of the form $\forall\overrightarrow{X_i}.(C'{\rightarrow}A)$ for some C'. However, since $A{\rightarrow}D$ is "closed" in a context in which $\overline{E}$ occurs, none of the types in A may be closed at this point either, and so $\overline{E}$ must (also) be of the form $\forall\overrightarrow{Y_j}.(A{\rightarrow}D')$ for some D'. These constraints can only all be fulfilled if there are no quantifiers, and $\overline{E} = A{\rightarrow}A$. Without a polymorphic type for γ, the term on the right of the cut in Example 3.3 cannot be typed (the "self-application" fails).

We can show the following properties for this type system:

Proposition 3.8 (Basic properties)**.**

1. *For all substitutions S, if $P \;:\cdot\; \Gamma \vdash_{\mathrm{SP}} \Delta$ then $P \;:\cdot\; (S\ \Gamma) \vdash_{\mathrm{SP}} (S\ \Delta)$.*
2. *(Weakening) If $P \;:\cdot\; \Gamma \vdash_{\mathrm{SP}} \Delta$, and $\langle\Gamma{\cup}\Gamma';\Delta{\cup}\Delta'\rangle$ is a well-formed context, then then $P \;:\cdot\; \Gamma{\cup}\Gamma' \vdash_{\mathrm{SP}} \Delta{\cup}\Delta'$.*
3. *(Strengthening) If $P \;:\cdot\; \Gamma{\cup}\Gamma' \vdash_{\mathrm{SP}} \Delta{\cup}\Delta'$, with no sockets x occurring both in Γ' and in $fs(P)$, and similarly no plugs α occurring in both Δ' and $fp(P)$, then $P \;:\cdot\; \Gamma \vdash_{\mathrm{SP}} \Delta$.*
4. *If $P \;:\cdot\; \Gamma, x:\overline{B} \vdash_{\mathrm{SP}} \Delta$ and $\overline{A}{\geq}\overline{B}$ then $P \;:\cdot\; (\Gamma\backslash x), x:\overline{A} \vdash_{\mathrm{SP}} \Delta$.*
5. *If $P \;:\cdot\; \Gamma \vdash_{\mathrm{SP}} \Delta, \alpha:\overline{A}$ and $\overline{A}{\geq}\overline{B}$ then $P \;:\cdot\; \Gamma \vdash_{\mathrm{SP}} (\Delta\backslash\alpha), \alpha:\overline{B}$.*
6. *If $P \;:\cdot\; \Gamma \vdash_{\mathrm{SP}} \Delta$ and $\Delta{\geq}\Delta'$ then $P \;:\cdot\; \Gamma \vdash_{\mathrm{SP}} \Delta'$.*

Proof. See Proof B.2 in Appendix B. □

The new type system is a proper restriction of the old one, which can be understood by adding back the explicit quantifier rules in the naïve type system wherever the $\geq$ and $\lhd_{\langle\Gamma;\Delta\rangle}$ relations are employed in the improved type system. We omit the rather-lengthly details here, since we do not depend on this result. However, in brief, in the case of the (ax) rule one employs an (ax) rule followed by a (possibly empty) sequence of $(\forall\mathcal{L})$ rules, followed by a (possibly empty) sequence of $(\forall\mathcal{R})$ rules. In all other cases which employ $\geq$, a (possibly empty) sequence of $(\forall\mathcal{L})$

rules is added. In all cases which employ $\lhd_{\langle\Gamma;\Delta\rangle}$, a (possibly empty) sequence of ($\forall\mathcal{R}$) rules is added.

In order to deal succinctly with the more-complex inference rules of the improved type system in the following proofs, we employ the following straightforward lemma:

Lemma 3.9 (Generation Lemma)**.** 1. *$\langle x.\alpha\rangle :\cdot \Gamma \vdash_{\text{SP}} \Delta$ if and only if $\Gamma = \Gamma', x:\overline{A}$ and $\Delta = \alpha:\overline{B}, \Delta'$ with $\overline{A} \geq \overline{B}$.*

2. *$\widehat{x}P\widehat{\alpha}\cdot\beta :\cdot \Gamma \vdash_{\text{SP}} \Delta$ if and only if $x \notin \Gamma$ and $\alpha \notin \Delta$ and $\Delta = \Delta', \beta:\overline{C}$ and there exist A,B such that $A \rightarrow B \lhd_{\langle\Gamma;\Delta'\rangle} \overline{C}$ and $P :\cdot \Gamma, x:A \vdash_{\text{SP}} \alpha:B, \Delta'$.*
3. *$P\widehat{\alpha}[x]\widehat{y}Q :\cdot \Gamma \vdash_{\text{SP}} \Delta$ if and only if $\alpha \notin \Delta$ and $y \notin \Gamma$ and $\Gamma = \Gamma', x:\overline{C}$ and there exist A,B such that $\overline{C} \geq A \rightarrow B$ and $P :\cdot \Gamma' \vdash_{\text{SP}} \alpha:A, \Delta$ and $Q :\cdot \Gamma', y:B \vdash_{\text{SP}} \Delta$.*
4. *$P\widehat{\alpha} \dagger \widehat{x}Q :\cdot \Gamma \vdash_{\text{SP}} \Delta$ if and only if $\alpha \notin \Delta$ and $x \notin \Gamma$ and there exists $\overline{A}$ such that $P :\cdot \Gamma \vdash_{\text{SP}} \alpha:\overline{A}, \Delta$ and $Q :\cdot \Gamma, x:\overline{A} \vdash_{\text{SP}} \Delta$.*

Proof. Each case follows from the fact that each syntactic construct can be typed by a unique typing rule, imposing exactly the conditions described. □

We can now show that this new type system amends the unsoundness of the previous one.

Theorem 3.10 (Witness Reduction for Improved Type Assignment)**.** 1. *If both of the following hold:*

$$P :\cdot \Gamma \vdash_{\text{SP}} \alpha:\overline{A}, \Delta \tag{1}$$
$$Q :\cdot \Gamma, x:\overline{A} \vdash_{\text{SP}} \Delta \tag{2}$$

then we have:

(a) $Q\{P\widehat{\alpha} \leftrightarrow x\} :\cdot \Gamma \vdash_{\text{SP}} \Delta$

(b) $P\{\alpha \leftrightarrow \widehat{x}Q\} :\cdot \Gamma \vdash_{\text{SP}} \Delta$

2. *If $P :\cdot \Gamma \vdash_{\text{SP}} \Delta$ and $P \rightarrow Q$ then $Q :\cdot \Gamma \vdash_{\text{SP}} \Delta$.*

Proof. See Proof B.3 in Section B for full details. We duplicate here a few representative cases:

1. (a) By induction on the structure of the term Q.

   $Q = \langle x.\beta\rangle$**:** Then $Q\{P\widehat{\alpha} \leftrightarrow x\} = P\widehat{\alpha} \dagger \widehat{x}Q$ and the result follows by application of the (*cut*) rule.

$Q = \langle y.\beta\rangle$, $y \neq x$: Then $Q\{P\widehat{\alpha} \leftrightarrow x\} = Q$. Since $x \notin fs(Q)$, By (*Eq.* 2) and Proposition 3.8 3 we obtain $Q :\cdot\ \Gamma \vdash_{\text{SP}} \Delta$ as required.

$Q = Q_1\widehat{\beta}\,[x]\,\widehat{z}Q_2$:
$Q\{P\widehat{\alpha} \leftrightarrow x\} = P\widehat{\alpha} \dagger \widehat{y}((Q_1\{P\widehat{\alpha} \leftrightarrow x\})\widehat{\beta}\,[y]\,\widehat{z}(Q_2\{P\widehat{\alpha} \leftrightarrow x\}))$ in which y is fresh. By (*Eq.* 2) and Lemma 3.9 3, there exist $B,C,\overline{D},\Gamma'$ such that $\overline{D} \geq (B \rightarrow C)$ and $\Gamma = \Gamma', y : \overline{D}$ and (by weakening, by applying Proposition 3.8 2 where necessary) both $Q_1 :\cdot\ \Gamma, x : \overline{A} \vdash_{\text{SP}} \beta : B, \Delta$ and $Q_2 :\cdot\ \Gamma, x : \overline{A}, z : C \vdash_{\text{SP}} \Delta$. By induction, twice, we obtain that both $Q_1\{P\widehat{\alpha} \leftrightarrow x\} :\cdot\ \Gamma \vdash_{\text{SP}} \beta : B, \Delta$ and also $Q_2\{P\widehat{\alpha} \leftrightarrow x\} :\cdot\ \Gamma, z : C \vdash_{\text{SP}} \Delta$. Since $\overline{D} \geq (B \rightarrow C)$, we can apply the $(\rightarrow\mathcal{L})$ rule to obtain the judgement $(Q_1\{P\widehat{\alpha} \leftrightarrow x\})\widehat{\beta}\,[y]\,\widehat{z}(Q_2\{P\widehat{\alpha} \leftrightarrow x\}) :\cdot\ \Gamma, y : \overline{D} \vdash_{\text{SP}} \Delta$. Finally, we apply the (*cut*) rule to obtain the required result.

(b) By induction on the structure of the term P. The argument is similar to the previous part, and we show only the most-interesting case, where $P = \widehat{y}P_1\widehat{\beta}\cdot\alpha$. Then $P\{\alpha \leftrightarrow \widehat{x}Q\} = (\widehat{y}(P_1\{\alpha \leftrightarrow \widehat{x}Q\})\widehat{\beta}\cdot\gamma)\widehat{\gamma} \dagger \widehat{x}Q$, in which γ is fresh. By (*Eq.* 1) and Lemma 3.9 2, there exist B,C with $P_1 :\cdot\ \Gamma, y : B \vdash_{\text{SP}} \beta : C, \Delta$ and $B \rightarrow C \lhd_{\langle\Gamma;\Delta\rangle} \overline{A}$. By applying Proposition 3.8 2 as necessary, we obtain $P_1 :\cdot\ \Gamma, x : \overline{A}, y : B \vdash_{\text{SP}} \beta : C, \Delta$ and $Q :\cdot\ \Gamma, x : \overline{A}, y : B \vdash_{\text{SP}} \beta : C, \Delta$. By induction, $P_1\{\alpha \leftrightarrow \widehat{x}Q\} :\cdot\ \Gamma, y : B \vdash_{\text{SP}} \beta : C, \Delta$. By the rule $(\rightarrow\mathcal{R})$ we obtain $\widehat{y}(P_1\{\alpha \leftrightarrow \widehat{x}Q\})\widehat{\beta}\cdot\gamma :\cdot\ \Gamma \vdash_{\text{SP}} \gamma : \overline{A}, \Delta$. Finally, by the rule (*cut*) we obtain that $(\widehat{y}(P_1\{\alpha \leftrightarrow \widehat{x}Q\})\widehat{\beta}\cdot\gamma)\widehat{\gamma} \dagger \widehat{x}Q :\cdot\ \Gamma \vdash_{\text{SP}} \Delta$ as required.

2. By inductions on the number of reduction steps, and the structure of the term P, we need only consider the case where P is the redex itself, and is reduced in one step to Q. Therefore, we show the witness reduction result for each of the reduction rules in turn:

(*cap*) : $\langle x.\alpha\rangle\widehat{\alpha} \dagger \widehat{y}\langle y.\beta\rangle \rightarrow \langle x.\beta\rangle$
Suppose $\langle x.\alpha\rangle\widehat{\alpha} \dagger \widehat{y}\langle y.\beta\rangle :\cdot\ \Gamma \vdash_{\text{SP}} \Delta$. By Lemma 3.9 4, $\alpha \notin \Delta$ and $x \notin \Gamma$ and there exists $\overline{B}$ such that $\langle x.\alpha\rangle :\cdot\ \Gamma \vdash_{\text{SP}} \alpha : \overline{B}, \Delta$ and $\langle y.\beta\rangle :\cdot\ \Gamma, y : \overline{B} \vdash_{\text{SP}} \Delta$. By applying Lemma 3.9 1 twice, there exists $\overline{A},\overline{C},\Gamma',\Delta'$ such that $\Gamma = \Gamma', x : \overline{A}$ and $\Delta = \beta : \overline{C}, \Delta'$ with $\overline{A} \geq \overline{B}$ and $\overline{B} \geq \overline{C}$. By Proposition 3.6 1, $\overline{A} \geq \overline{C}$. Therefore, by the rule (*ax*), we obtain $\langle x.\beta\rangle :\cdot\ \Gamma', x : \overline{A} \vdash_{\text{SP}} \beta : \overline{C}, \Delta'$ as required.

(*impR*) : $(\widehat{x}P\widehat{\alpha}\cdot\beta)\widehat{\beta} \dagger \widehat{y}\langle y.\gamma\rangle \rightarrow \widehat{x}P\widehat{\alpha}\cdot\gamma$ (*if* $\beta \notin fp(P)$)
Suppose $(\widehat{x}P\widehat{\alpha}\cdot\beta)\widehat{\beta} \dagger \widehat{y}\langle y.\gamma\rangle :\cdot\ \Gamma \vdash_{\text{SP}} \Delta$. By Lemma 3.9 4, $\beta \notin \Delta$ and $y \notin \Gamma$ and there exists $\overline{C}$ such that $\widehat{x}P\widehat{\alpha}\cdot\beta :\cdot\ \Gamma \vdash_{\text{SP}} \beta : \overline{C}, \Delta$ and $\langle y.\gamma\rangle :\cdot\ \Gamma, y : \overline{C} \vdash_{\text{SP}} \Delta$. By Lemma 3.9 2, there exist A,B,Δ'' such that $(A \rightarrow B) \lhd_{\langle\Gamma;\Delta\rangle} \overline{C}$ and

$P :\cdot \Gamma, x : A \vdash_{\text{SP}} \alpha : B, \Delta''$ and $(\beta : \overline{C}, \Delta) = (\beta : \overline{C}, \Delta'')$. Since $\beta \notin P$, by Proposition 3.8 3, we may assume without loss of generality that we have $\beta \notin \Delta''$, and therefore that $\Delta'' = \Delta$. By Lemma 3.9 1, there exist $\overline{D}, \Delta'$ such that $\Delta = \gamma : \overline{D}, \Delta'$ and $\overline{C} \geq \overline{D}$. By Proposition 3.6 6, there exists a substitution S such that $(S\ \langle \Gamma; \Delta \rangle) = \langle \Gamma; \Delta \rangle$ and $(S\ A \to B) \lhd_{\langle \Gamma; \Delta \rangle} \overline{D}$. By Proposition 3.8 1, we have $P :\cdot \Gamma, x : (S\ A) \vdash_{\text{SP}} \alpha : (S\ B), \gamma : \overline{D}, \Delta'$. By the rule $(\to\mathcal{R})$, we deduce that $\widehat{x} P \widehat{\alpha} \cdot \gamma :\cdot \Gamma \vdash_{\text{SP}} \gamma : \overline{D}, \Delta$ as required.

(*prop*-R) : $P\widehat{\alpha} \dagger \widehat{x} Q \to Q\{P\widehat{\alpha} \leftrightarrow x\}$(*if Q does not introduce x*)
By Lemma 3.9 4 and part 1a.

(*prop*-L) : $P\widehat{\alpha} \dagger \widehat{x} Q \to P\{\alpha \leftrightarrow \widehat{x} Q\}$(*if P does not introduce* α)
By Lemma 3.9 4 and part 1b.

□

3.3. *Encoding ML in $\mathcal{X}^i$*

Using our previous observation concerning the fact that *let* and a cut can both explicitly represent a substitution, we define an encoding of the language of ML into $\mathcal{X}^i$.

Definition 3.11 (Encoding ML in $\mathcal{X}^i$)**.**

$$
\begin{aligned}
\llbracket x \rrbracket_\alpha^{\text{ML}} &= \langle x.\alpha \rangle \\
\llbracket \lambda x.M \rrbracket_\alpha^{\text{ML}} &= \widehat{x} \llbracket M \rrbracket_\beta^{\text{ML}} \widehat{\beta} \cdot \alpha \\
\llbracket MN \rrbracket_\alpha^{\text{ML}} &= \llbracket M \rrbracket_\beta^{\text{ML}} \widehat{\beta} \dagger \widehat{y} (\llbracket N \rrbracket_\gamma^{\text{ML}} \widehat{\gamma}\, [y]\, \widehat{z} \langle z.\alpha \rangle) \\
\llbracket \mathit{let}\ x = M\ \texttt{in}\ N \rrbracket_\alpha^{\text{ML}} &= \llbracket M \rrbracket_\beta^{\text{ML}} \widehat{\beta} \dagger \widehat{x} \llbracket N \rrbracket_\alpha^{\text{ML}}
\end{aligned}
$$

where y, z, β, γ are fresh connectors.

This is an extension of the encoding of λ-calculus given for the $\mathcal{X}$-calculus [25]. In all cases there is exactly one occurrence of the plug α in the resulting $\mathcal{X}^i$-term, and this is the only free plug[9].

Lemma 3.12 (Cuts simulate substitutions)**.**

1. *For all ML-terms M, N, $\llbracket N \rrbracket_\alpha^{\text{ML}} \{ \llbracket M \rrbracket_\beta^{\text{ML}} \widehat{\beta} \leftrightarrow x \} \to \llbracket (N[M/x]) \rrbracket_\alpha^{\text{ML}}$.*

[9]Note that this is consistent with the logical perspective on these calculi also: the restriction of classical sequent calculus to precisely one conclusion on the right of the sequent gives Gentzen's sequent calculus for minimal logic, which underpins typical functional programming languages via the original Curry-Howard Correspondence.

2. *For all ML-terms* M, N, $\llbracket M \rrbracket^{\mathrm{ML}}_{\beta} \widehat{\beta} \dagger \widehat{x} \llbracket N \rrbracket^{\mathrm{ML}}_{\alpha} \to \llbracket (N[M/x]) \rrbracket^{\mathrm{ML}}_{\alpha}$.

Proof. 1. By straightforward induction on the structure of the term N.

2. From the previous part.

□

The fact that such a cut behaves like the substitution of the original system relies on the fact that β occurs only once in the left-hand subterm of the cut. If an arbitrary $\mathcal{X}^i$-term were to appear here in which β occurred many times, the cut might be activated to the left (via the rule (*act*-L)), and copies of the right-hand term made during propagation; then the behaviour might be quite different.

Returning to our encoding of ML, we have the following result.

Theorem 3.13 (Simulation of ML)**.** *For all* ML*-terms* M, N*, if* $M \to_{ML} N$ *then* $\llbracket M \rrbracket^{\mathrm{ML}}_{\alpha} \to \llbracket N \rrbracket^{\mathrm{ML}}_{\alpha}$.

Proof. Examining definition 2.2, there are two cases to consider.

($M \equiv (\lambda x.M_1\ M_2)$) Then $N \equiv M_1[M_2/x]$. Applying Definition 3.11, we can see that:

$$\begin{aligned}
\llbracket (\lambda x.M_1)\ M_2 \rrbracket^{\mathrm{ML}}_{\beta} &= \llbracket \lambda x.M_1 \rrbracket^{\mathrm{ML}}_{\delta} \widehat{\delta} \dagger \widehat{y}(\llbracket M_2 \rrbracket^{\mathrm{ML}}_{\epsilon} \widehat{\epsilon}\,[y]\,\widehat{z}\langle z.\beta\rangle) \\
&= (\widehat{x} \llbracket M_1 \rrbracket^{\mathrm{ML}}_{\phi} \widehat{\phi} \cdot \delta) \widehat{\delta} \dagger \widehat{y}(\llbracket M_2 \rrbracket^{\mathrm{ML}}_{\epsilon} \widehat{\epsilon}\,[y]\,\widehat{z}\langle z.\beta\rangle) \\
&\to \llbracket M_2 \rrbracket^{\mathrm{ML}}_{\epsilon} \widehat{\epsilon} \dagger \widehat{x}(\llbracket M_1 \rrbracket^{\mathrm{ML}}_{\phi} \widehat{\phi} \dagger \widehat{z}\langle z.\beta\rangle) \\
&\to \llbracket M_2 \rrbracket^{\mathrm{ML}}_{\epsilon} \widehat{\epsilon} \dagger \widehat{x}(\llbracket M_1 \rrbracket^{\mathrm{ML}}_{\phi} [\beta/\phi]) \\
&= \llbracket M_2 \rrbracket^{\mathrm{ML}}_{\epsilon} \widehat{\epsilon} \dagger \widehat{x} \llbracket M_1 \rrbracket^{\mathrm{ML}}_{\beta}
\end{aligned}$$

This case is completed by Lemma 3.12.

($M \equiv \mathit{let}\ x = M_2\ \mathtt{in}\ M_1$) Again, $N \equiv M_1[M_2/x]$. The result follows immediately from Lemma 3.12, noting that

$$\llbracket \mathit{let}\ x = M_2\ \mathtt{in}\ M_1 \rrbracket^{\mathrm{ML}}_{\beta} = \llbracket M_2 \rrbracket^{\mathrm{ML}}_{\epsilon} \widehat{\epsilon} \dagger \widehat{x} \llbracket M_1 \rrbracket^{\mathrm{ML}}_{\beta}$$

□

We can show that our type system is at least as flexible as the restricted type system for ML which we have generalised:

Proposition 3.14 (Preservation of Typings)**.** *For all* ML*-terms M, if $\Gamma \vdash_{\mathrm{ML}} M : \overline{A}$ in the type system in which polymorphism is restricted to let-terms which bind values [30], then $[\![M]\!]^{\mathrm{ML}}_{\beta} :\cdot\ \Gamma \vdash_{\mathrm{SP}} \beta : \overline{A}$.*

It is natural to ask whether our generalisation is useful in the original context of ML. For example, can we define a type system for ML based on the observations of this chapter which allows more typeable terms than [30], and is still sound? We can answer this question in the affirmative; if we define a type system via our encoding into the $\mathcal{X}^i$-calculus (i.e., we encode ML-terms and then type their encodings), we obtain a more permissive system. A simple example of this extra flexibility can be seen as follows:

Example 3.15 (Enhanced type assignment for ML)**.** *Consider the following ML-term: let $x = (\mathit{let}\ y = \lambda z.z\ \mathtt{in}\ y)\ \mathtt{in}\ x$. This term reduces to the identity $\lambda z.z$, and it would be nice if this could be reflected by its assignable types. Furthermore, since there are no free variables in the term (and no imperative features, of course), it seems as though it must be safe to do so. However, the value restriction [30] does not permit the outermost let-construct to employ a polymorphic type, since let $y = \lambda z.z\ \mathtt{in}\ y$ is not a value. Considering the encoding into $\mathcal{X}^i$; we obtain (using the plug α as output of the whole term) $((\widehat{z}\langle z.\beta\rangle\widehat{\beta}\cdot\gamma)\widehat{\gamma} \dagger \widehat{y}\langle y.\delta\rangle)\widehat{\delta} \dagger \widehat{x}\langle x.\alpha\rangle$. The polymorphic type derivable for the term $\widehat{z}\langle z.\beta\rangle\widehat{\beta}\cdot\gamma$ can be 'carried through' each of the cuts, and in particular, when the subterm $\langle y.\delta\rangle$ is typed, the polymorphic type can be assigned to δ immediately, as is required by the system. The first part of such a typing derivation follows (the outermost cut is typed analogously to the one shown):*

$$\dfrac{\dfrac{\dfrac{}{\langle z.\beta\rangle \ \vdots\cdot\ z:\varphi \vdash_{\mathrm{SP}} \beta:\varphi}\,(ax)}{\widehat{z}\langle z.\beta\rangle\widehat{\beta}\cdot\gamma \ \vdots\cdot\ \emptyset \vdash_{\mathrm{SP}} \gamma:\forall X.(X\to X)}\,(\to\mathcal{R}) \qquad \dfrac{}{\langle y.\delta\rangle \ \vdots\cdot\ y:\forall X.(X\to X) \vdash_{\mathrm{SP}} \delta:\forall X.(X\to X)}\,(ax)}{(\widehat{z}\langle z.\beta\rangle\widehat{\beta}\cdot\gamma)\widehat{\gamma} \dagger \widehat{y}\langle y.\delta\rangle \ \vdots\cdot\ \emptyset \vdash_{\mathrm{SP}} \delta:\forall X.(X\to X)}\,(cut)$$

Therefore the application of our ideas to ML yields a more-permissive type system than that proposed by Wright [30]. Our type system will not allow a polymorphic type to be given to the result of an application, but allows greater flexibility when typing nested let-bindings (cuts). A more-detailed exploration of the precise class of programs which are permitted by our approach, and whether this can be safely extended, remains future work.

3.4. Principal Contexts

It is well known that a notion of principal types for ML terms exists (as presented by Milner), with respect to an initial basis Γ. This result is shown through

the definition of the algorithm $\mathcal{W}$, which takes as input an ML-term and initial basis Γ, and can be used to compute the most general pair of substitution S and generic type $\overline{A}$ such that $(S\ \Gamma) \vdash_{\text{ML}} M : \overline{A}$.

In the case of Milner's algorithm $\mathcal{W}$, the types returned are not quantified, but in showing the completeness of the algorithm the $\forall$-closure of the type (see Definition 3.19 below) is taken. The closure can be seen to convert a type into its most general form, and so it can be argued that the principal type should be defined after this closure is taken. This is the idea we follow here; we will generalise the types of our outputs as much as possible, in our definition of a principal context.

In order to formalise our results, we require the usual notion of Curry substitution, along with some auxiliary definitions:

Definition 3.16 (Substitutions)**.**

1. *A* substitution *S is a (possibly empty) set of pairs (φ, A) where each φ is a distinct atomic type, and each A a type. The pair is meant to denote the replacement of occurrences of φ with A. Hence, as notational sugar, we write such pairs $(\varphi \mapsto A)$.*
2. *For any substitution S and type A, the* action *of S on A, written $(S\ A)$ is defined recursively as follows:*

$$(S\ \varphi) \triangleq \left\{ \begin{array}{ll} A & \textit{if } \exists(\varphi \mapsto A) \in S \\ \varphi & \textit{otherwise} \end{array} \right\}$$
$$(S\ A_1 \to A_2) \triangleq (S\ A_1) \to (S\ A_2)$$

3. *In the special case where the set of pairs is empty, we use a special symbol id, and call this the* identity substitution.
4. *We extend this definition to allow substitutions to act on contexts in the obvious way (i.e. the substitution is performed on all the types in the context).*
5. *For a substitution S we define $dom(S) = \{\varphi \mid (S\ \varphi) \neq \varphi\}$.*
6. *For a substitution S we define $range(S) = \bigcup_{\varphi \in dom(S)} atoms(S\ \varphi)$.*
7. *A subsitution S is* idempotent *if, for all atomic types φ, $((S\ \varphi)\ \varphi) = (S\ \varphi)$.*

Principal contexts will be defined using Robinson's unification algorithm [16]. Unification is also extended to contexts of sockets (and identically for plugs) in the following definition:

Definition 3.17 (Unification)**.** 1. *The algorithm unify [16] takes two types as arguments and returns a substitution (or fails: we do not model failure cases*

explicitly, but assume that if none of the definitions below apply then the algorithm terminates immediately with failure). It is defined as follows:

$$
\begin{array}{rl}
\mathit{unify}\ \varphi\ \varphi & = \mathit{id} \\
\mathit{unify}\ \varphi\ A & = (\varphi \mapsto A)\ \ \mathit{if}\ \varphi \notin A \\
\mathit{unify}\ A\ \varphi & = \mathit{unify}\ \varphi\ A \\
\mathit{unify}\ A_1 \to A_2\ B_1 \to B_2 & = S_2 \circ S_1 \\
 & \quad \mathit{where} \\
 & \qquad S_1 = \mathit{unify}\ A_1\ B_1 \\
 & \qquad S_2 = \mathit{unify}\ (S_1\ A_2)\ (S_1\ B_2)
\end{array}
$$

2. *Unification is extended to contexts as follows (where $\emptyset$ denotes an empty context):*

$$
\mathit{unifyContexts}\ \emptyset\ \Gamma_2 = \mathit{id}
$$

$$
\mathit{unifyContexts}\ (x : A, \Gamma_1)\ \Gamma_2 = \left\{ \begin{array}{ll}
\mathit{unifyContexts}\ \Gamma_1\ \Gamma_2 & \mathit{if}\ x \notin \Gamma_2 \\
S_2 \circ S_1 & \mathit{if}\ x : B \in \Gamma_2 \\
\quad \mathit{where} & \\
S_1 = \mathit{unify}\ A\ B & \\
S_2 = \mathit{unifyContexts}\ (S_1\ (\Gamma_1 \backslash x))\ (S_1\ (\Gamma_2 \backslash x)) &
\end{array} \right\}
$$

Recall that for a well-formed context $(x : A, \Gamma)$, it does not automatically follow that x is not mentioned in Γ (so long as it is with the type A), which is the reason for explicitly removing x in the recursive call above.

We assume the classical soundness and completeness results for unification, along with their extension to contexts:

Lemma 3.18 (Soundness and Completeness of Unification [16])**.**

1. *If unify A B succeeds, then it yields an idempotent substitution S_u satisfying $(S_u\ A) = (S_u\ B)$.*
2. *If there exists a substitution S such that $(S\ A) = (S\ B)$ then unify A B succeeds, yielding an idempotent substitution S_u, and there exists an idempotent substitution S' such that $S = S' \circ S_u$.*
3. *If unifyContexts Γ_1 Γ_2 succeeds, then it yields an idempotent substitution S_u satisfying $(S_u\ \Gamma_1) = (S_u\ \Gamma_2)$.*
4. *If there exists S such that $(S\ \Gamma_1) = (S\ \Gamma_2)$ then unifyContexts Γ_1 Γ_2 succeeds, yielding an idempotent substitution S_u, and there exists an idempotent substitution S' such that $S = S' \circ S_u$.*

We can define principal contexts in our shallow polymorphic version of $\mathcal{X}^i$, with respect to a given initial left-context Γ which gives types to the free sockets in a term. We define an algorithm, based loosely on the $\mathcal{W}$ algorithm of [21], which takes as input an $\mathcal{X}^i$-term P and a left-context Γ, and either fails (in which case P is not typeable) or else produces a pair of substitution S and right-context Δ, representing the least substitution and strongest right-context possible such that $P :\cdot (S\ \Gamma) \vdash_{\mathrm{SP}} \Delta$. Before we are able to define this algorithm, we need to define a number of 'helper' operations.

Firstly, we require an operation to take the 'strongest' closure of a generic type $\overline{A}$ in a context $\langle\Gamma;\Delta\rangle$; essentially this implicitly applies the $(\forall\mathcal{R})$ to the appropriate statement as many times as possible. Viewed otherwise, the operation computes the 'largest' (in the $\geq$ relation) generic type $\overline{B}$, such that $\overline{A} \lhd_{\langle\Gamma;\Delta\rangle} \overline{B}$.

Definition 3.19 ($\forall$-closure)**.** *The* $\forall$-closure *of type* $\overline{A}$ *with respect to a context* $\langle\Gamma;\Delta\rangle$*, is defined by:* $\forall$*-closure* $\overline{A}\,\langle\Gamma;\Delta\rangle = \forall X_1 \ldots \forall X_n.(\overline{A}[\overline{X_i/\varphi_i}])$ *where* $\varphi_1, \ldots, \varphi_n$ *are exactly the atomic types occurring in* $\overline{A}$ *but not in* $\langle\Gamma;\Delta\rangle$*.*

The process of $\forall$-closure may be seen as taking the 'largest' possible form of a type, in terms of the ordering imposed by $\preceq$. We can show that this operation does indeed compute the 'largest' possible type, by the following result:

Proposition 3.20 ($\forall$-closure is the most general closure)**.** *If* $\overline{B} = \forall$*-closure* $\overline{A}\,\langle\Gamma;\Delta\rangle$ *then:*

1. $\overline{A} \lhd_{\langle\Gamma;\Delta\rangle} \overline{B}$.
2. *If* $\overline{A} \lhd_{\langle\Gamma;\Delta\rangle} \overline{C}$ *then* $\overline{B} \geq \overline{C}$.
3. *For all substitutions* S, $(S\ \overline{B}) \geq \forall$*-closure* $(S\ \overline{A})\ \langle(S\ \Gamma);(S\ \Delta)\rangle$.
4. *If* $\Delta \geq \Delta'$ *then* $\overline{B} \geq \forall$*-closure* $\overline{A}\,\langle\Gamma;\Delta'\rangle$.

Proof. See Proof B.4 in Section B. □

In our amended type system, whenever a statement is introduced into a right-context it may be 'closed' to a stronger type (with more $\forall$ quantification). Furthermore, this is the only point in the derivation at which these kinds of generalisations may be applied to the statement. For our type-inference algorithm to compute the most general right-context possible, it will use the operation of $\forall$-*closure* whenever such closures are permitted by the rules, in order to obtain the strongest possible type so far. For example, if we were to run our algorithm on the term $\widehat{x}\langle x.\alpha\rangle\widehat{\alpha}\cdot\beta$, we would expect it to generate a type such as $(\varphi\rightarrow\varphi)$ for β but then to also quantify (close) it to the most-general possible type $\forall X.(X\rightarrow X)$.

This approach seems in line with the presentation of our type inference rules; we are employing them in the most-general way possible. However, it leads to a new problem when the contraction of multiple occurrences of a plug β in a term takes place. In general, different quantified types get computed by the algorithm for the different occurrences of a plug β, and at some stage these need to be 'merged' into just one type that works in all positions. In a simple type system, without quantified types, one usually applies Robinson's unification algorithm to perform this 'merging'. However, we need to deal with the fact that quantifiers will, in general, occur in the types. Furthermore, we wish the resulting type to itself be quantified as much as possible.

This leads to a desire for an operation which, given two generic types $\overline{A}$ and $\overline{B}$, computes a third generic type $\overline{C}$ which is the 'most general' type which can be used in place of both $\overline{A}$ and $\overline{B}$. This has parallels with unification; indeed we would expect that if both $\overline{A}$ and $\overline{B}$ contain no quantifiers, then it would perform exactly the operation of unification. On the other hand, if $\overline{A}$ and $\overline{B}$ contained no atomic types, it would seem reasonable that the operation should compute the 'biggest' (in the $\geq$ sense) generic type which is a generic instance of both $\overline{A}$ and $\overline{B}$. In general, we seek the 'biggest' generic type $\overline{C}$ and minimal substitution S such that both $(S\ \overline{A})\geq\overline{C}$ and $(S\ \overline{B})\geq\overline{C}$. Informally, we seek a most general solution in S and $\overline{C}$ to the problem:

$$(S\ \overline{A})\geq\overline{C}\ \&\ (S\ \overline{B})\geq\overline{C}$$

We define an algorithm, which we call 'generic unification', in order to compute this 'most general solution'. In order to do so, we need to introduce operations to modify the domains of substitutions. This is because, during the algorithm, fresh instances of the generic types will be taken, and the substitutions subsequently defined will (in general) act on the fresh atomic types introduced. However, these types were not present in the original generic types, and so the resulting substitution would not be the most general one; it might perform the minimal operations on $\overline{A}$ and $\overline{B}$ but *also* perform other operations which are redundant from the point of view of the initial problem.

In order to overcome these difficulties, we define two new operations on substitutions. Firstly, we define the *restriction* of a substitution S to a set of atomic types Φ, which is written $(S\cap\Phi)$ and is itself a substitution which acts on elements of Φ exactly as S does, and on all other atomic types as the identity substitution.

As a shorthand, we also define a complementary operation $(S\ \cap\ (dom(S)\backslash\Phi))$ (i.e. restricting a substitution to everything *but* the set Φ), which we write as

$(S \backslash \Phi)$ and read as "S without Φ".

We give formal definitions as follows:

Definition 3.21 (Restriction of a substitution)**.** *For any substitution S and set of atomic types Φ, the* restriction of S to Φ, *written $(S \cap \Phi)$ is defined by:*

$$(S \cap \Phi) = \{(\varphi \mapsto A) | (\varphi \mapsto A) \in S \ \& \ \varphi \in \Phi\}$$

We also define the shorthand:

$$(S \backslash \Phi) = (S \cap (dom(S) \backslash \Phi)) = \{(\varphi \mapsto A) | (\varphi \mapsto A) \in S \ \& \ \varphi \notin \Phi\}$$

In order to reason formally about the effect of these operations later on, we will require a number of properties about their definitions.

Lemma 3.22 (Range and domain)**.**

1. *For any substitutions S_1, S_2 if $dom(S_1) \cap range(S_2) = \emptyset$ and $dom(S_2) \cap range(S_1) = \emptyset$ then $(S_2 \circ S_1) = (S_1 \circ S_2)$.*
2. *If $S_2 \circ S_1 = S_4 \circ S_3$ and $dom(S_2) \cap range(S_1) = \emptyset$ and $dom(S_2) \cap dom(S_3) = \emptyset$ and $dom(S_2) \cap range(S_3) = \emptyset$, then there exists a substitution S_5 such that $S_1 = S_5 \circ S_3$.*
3. *For any substitution S, generic type $\overline{A}$ and atomic type φ, if $\varphi \in (S\ \overline{A})$ then either:*
   (a) *$\varphi \in atoms(\overline{A})$ and $\varphi \notin dom(S)$*, or,
   (b) *$\varphi \notin atoms(\overline{A})$ and there exists $\varphi' \in atoms(\overline{A})$ with $\varphi \in atoms(S\ \varphi')$.*
4. *For any binding renaming $[\overrightarrow{X_i/\varphi_i}]$, and any generic type $\overline{A}$ and atomic type φ, if $\varphi \in atoms(\overline{A}[\overrightarrow{X_i/\varphi_i}])$ then $\varphi \in atoms(\overline{A})$ and $\varphi \notin \{\overrightarrow{\varphi_i}\}$.*

Lemma 3.23 (Restrictions)**.** 1. *If $atoms(\overline{A}) \subseteq \{\overrightarrow{\varphi_i}\}$ then $(S \cap \{\overrightarrow{\varphi_i}\}\ \overline{A}) = (S\ \overline{A})$.*

2. *For any two sets of atomic types $\{\overrightarrow{\varphi_i}\}$ and $\{\overrightarrow{\varphi_j}\}$, and for any generic type $\overline{A}$, we have $(A \backslash \{\overrightarrow{\varphi_i}\}) \backslash \{\overrightarrow{\varphi_j}\} = (A \backslash \{\overrightarrow{\varphi_j}\}) \backslash \{\overrightarrow{\varphi_i}\} = A \backslash (\{\overrightarrow{\varphi_i}\} \cup \{\overrightarrow{\varphi_j}\})$.*
3. *For any two sets of atomic types $\{\overrightarrow{\varphi_i}\}$ and $\{\overrightarrow{\varphi_j}\}$, and for any generic type $\overline{A}$, if $\{\overrightarrow{\varphi_i}\}$ and $\{\overrightarrow{\varphi_j}\}$ are disjoint sets then we have $(A \backslash \{\overrightarrow{\varphi_i}\}) \cap \{\overrightarrow{\varphi_j}\} = A \cap \{\overrightarrow{\varphi_j}\}$.*
4. *For any substitution S, generic type $\overline{A}$ and set of atomic types $\{\vec{\varphi}\}$, if it is the case that $atoms(\overline{A}) \cap \{\vec{\varphi}\} = \emptyset$ then $(S\ \overline{A}) = ((S \backslash \{\vec{\varphi}\})\ \overline{A})$.*
5. *For any substitution S and set of atomic types $\{\vec{\varphi}\}$, if $dom(S) \subseteq \{\vec{\varphi}\}$ then $S \backslash \{\vec{\varphi}\} = id$.*
6. *For any substitutions S_1 and S_2 and set of atomic types $\{\vec{\varphi}\}$, if $\{\vec{\varphi}\} \cap dom(S_2) \cap range(S_1) = \emptyset$ then $(S_2 \circ S_1) \backslash \{\vec{\varphi}\} = (S_2 \backslash \{\vec{\varphi}\}) \circ (S_1 \backslash \{\vec{\varphi}\})$.*

7. *For any substitutions S_1 and S_2 and set of atomic types $\{\vec{\varphi}\}$, if $\{\vec{\varphi}\} \cap dom(S_1) = \emptyset$ and $\{\vec{\varphi}\} \cap dom(S_2) \cap range(S_1) = \emptyset$ then $(S_2 \circ S_1) \backslash \{\vec{\varphi}\} = (S_2 \backslash \{\vec{\varphi}\}) \circ S_1$.*
8. *For any substitution S, generic type $\overline{A}$ and set of atomic types $\{\vec{\varphi}\}$, if $(S\ \overline{A}) = \overline{A}$ then $(S \backslash \{\vec{\varphi}\}\ \overline{A}) = \overline{A}$.*
9. *For any substitutions S, S', if it is the case that for all $\varphi \in dom(S')$, we have $(S'\ \varphi) = (S\ \varphi)$, then it holds that $S' = S \cap dom(S')$.*
10. *For any idempotent substitution S and set of atomic types $\{\vec{\varphi}\}$, we have:*

$$S = ((S \cap \{\vec{\varphi}\}) \circ (S \backslash \{\vec{\varphi}\}) = (S \backslash \{\vec{\varphi}\}) \circ ((S \cap \{\vec{\varphi}\})$$

Armed with these definitions and results, we can now present the definition of generic unification.

Definition 3.24 (Generic Unification)**.**

$$\begin{array}{rcl} unifyGen\ \overline{A}\ \overline{B} & = & (S_r, \overrightarrow{\forall X_i}.C_u[\overrightarrow{X_i/\varphi_i}]) \\ & & where \\ & & A' = freshInst(\overline{A}) \\ & & B' = freshInst(\overline{B}) \\ & & S_u = unify\ A'\ B' \\ & & C_u = (S_u\ A') \\ & & \{\vec{\varphi_i}\} = atoms(C_u) \backslash (atoms(S_u\ \overline{A}) \cup atoms(S_u\ \overline{B})) \\ & & S_r = (S_u \cap (atoms(\overline{A}) \cup atoms(\overline{B}))) \end{array}$$

Note that this algorithm may fail, in the case where the call *unify* A' B' results in failure. As usual, we do not model the failure case explicitly, but speak of success or failure of the algorithm as a whole.

We can give a formal justification for the definition of the algorithm, using the following results:

Theorem 3.25 (Soundness and Completeness of Generic Unification)**.** *For any generic types $\overline{A}$ and $\overline{B}$:*

1. *(Soundness:) If unifyGen $\overline{A}$ $\overline{B}$ succeeds, resulting in a pair $(S_r, \overline{C})$ then we have $(S_r\ \overline{A}) \geq \overline{C}$ and $(S_r\ \overline{B}) \geq \overline{C}$.*
2. *(Completeness:) If S is a substitution and $\overline{D}$ a generic type such that $(S\ \overline{A}) \geq \overline{D}$ and $(S\ \overline{B}) \geq \overline{D}$, then unifyGen $\overline{A}$ $\overline{B}$ succeeds, resulting in a pair $(S_r, \overline{C})$, and there exists a further substitution S' such that $S = S' \circ S_r$ and $(S'\ \overline{C}) \geq \overline{D}$.*

Proof. See Proof B.5 in Section B. □

Just as for the simple type assignment system (Definition 2.13), we require the generalisation of unification to contexts, we require here the generalisation of generic unification to right-contexts. We choose to omit a concrete definition, but depend on the following properties, which are relatively easy to guarantee given the previous theorem:

Proposition 3.26 (Soundness and Completeness of Generic Context Unification)**.** *There exists an algorithm unifyGenContexts which takes two right-contexts Δ_1 and Δ_2 as arguments, and (if it succeeds) returns a pair of substitution S_u and right-context Δ_u, satisfying:*

1. *If unifyGenContexts $\Delta_1\ \Delta_2$ succeeds, then we have $(S_u\ \Delta_1)\geq\Delta_u$ and $(S_u\ \Delta_2)\geq\Delta_u$.*
2. *If S is a substitution and Δ a right-context such that $(S\ \Delta_1)\geq\Delta$ and $(S\ \Delta_2)\geq\Delta$, then unifyGenContexts $\Delta_1\ \Delta_2$ succeeds, and there exists a further substitution S' such that $S = S'\circ S_u$ and $(S'\ \Delta_u)\geq\Delta$.*

We are now in a position to define our type-inference algorithm.

Definition 3.27 (*sppc*)**.** *The procedure* sppc $::\ \langle\mathcal{X}^i,\Gamma\rangle\rightarrow\langle S,\Delta\rangle$ *is defined in Figure 2.*

Example 3.28. *Consider the following $\mathcal{X}^i$-terms:*

$$\begin{aligned} M &= \widehat{x}\langle x.\alpha\rangle\widehat{\alpha}\cdot\beta \\ N &= M\widehat{\epsilon}\,[i]\,\widehat{j}M \\ O &= \widehat{x}(\widehat{y}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma)\widehat{\beta}\cdot\gamma \\ R &= \langle z.\delta\rangle\widehat{\delta}\,[z]\,\widehat{w}\langle w.\epsilon\rangle \end{aligned}$$

Then we have (some examples are shown in detail in Appendix A):

$$\begin{aligned} \text{sppc}\,(M,\Gamma) &= \langle id,\{\beta:\forall X.(X\rightarrow X)\}\rangle \textit{ for any context } \Gamma \\ \text{sppc}\,(N,\{i:E\rightarrow F\}) &= \langle id,\{\beta:\forall Z.(Z\rightarrow Z)\}\rangle \textit{ for any Curry types } E,F \\ \text{sppc}\,(O,\Gamma) &= \langle id,\{\gamma:\varphi\rightarrow\varphi\}\rangle \textit{ where } \varphi \textit{ is fresh, for any context } \Gamma \\ \text{sppc}\,(R,\{z:E\rightarrow E\}) & \quad \textit{fails, for any Curry type } E \\ \text{sppc}\,(R,\{z:\forall X.(X\rightarrow X)\}) &= \langle id,\{\gamma:\varphi\rightarrow\varphi\}\rangle \textit{ where } \varphi \textit{ is fresh} \\ \text{sppc}\,(M\widehat{\beta}\dagger\widehat{z}R,\emptyset) &= \langle id,\{\gamma:\varphi\rightarrow\varphi\}\rangle \textit{ where } \varphi \textit{ is fresh} \\ \text{sppc}\,(N\widehat{\beta}\dagger\widehat{z}R,\{i:E\rightarrow F\}) &= \langle id,\{\gamma:\varphi\rightarrow\varphi\}\rangle \textit{ where } \varphi \textit{ is fresh} \\ \text{sppc}\,(O\widehat{\beta}\dagger\widehat{z}R,\Gamma) & \quad \textit{fails, for any } \Gamma \end{aligned}$$

Note that the last of these examples is the term from Example 3.3; since this term runs to untypeable terms, the algorithm is correct to reject it. We can now give our principal contexts result.

$$
\begin{aligned}
&\mathit{sppc}\,(\langle x.\alpha\rangle, \Gamma) = \langle \mathit{id}, \{\alpha : \overline{A}\}\rangle \\
&\quad \text{where} \\
&\qquad \overline{A} = \mathit{typeof}\ x\ \Gamma \\
&\mathit{sppc}\,(\widehat{x}P\widehat{\alpha}\cdot\beta, \Gamma) = \langle S_r, (S_u\ \Delta_P \backslash \alpha \backslash \beta) \cup \{\beta : \overline{D}\}\rangle \\
&\quad \text{where} \\
&\qquad \varphi = \mathit{fresh} \\
&\qquad \langle S_P, \Delta_P\rangle = \mathit{sppc}\,(P, \Gamma \cup \{x : \varphi\}) \\
&\qquad A = (S_P\ \varphi) \\
&\qquad B = \mathit{freshInstance}\ \mathit{typeof}\ \alpha\ \Delta_P \\
&\qquad \overline{C} = \forall\text{-}\mathit{closure}\ A \rightarrow B\ \langle (S_P\ \Gamma); \Delta_P \backslash \alpha\rangle \\
&\qquad \langle S_u, \overline{D}\rangle = \begin{cases} \mathit{unifyGen}\ \overline{C}\ \mathit{typeof}\ \beta\ \Delta_P & \mathit{if}\ \beta \in \Delta_P \\ \langle \mathit{id}, \overline{C}\rangle & \mathit{otherwise} \end{cases} \\
&\qquad S_r = (S_u \circ S_P \cap \mathit{atoms}(\Gamma)) \\
&\mathit{sppc}\,(P\widehat{\alpha}\,[y]\,\widehat{x}Q, \Gamma) = \langle S_r, \Delta_c\rangle \\
&\quad \text{where} \\
&\qquad \langle S_P, \Delta_P\rangle = \mathit{sppc}\,(P, \Gamma) \\
&\qquad \varphi = \mathit{fresh} \\
&\qquad \langle S_Q, \Delta_Q\rangle = \mathit{sppc}\,(Q, (S_P\ \Gamma) \cup \{y : \varphi\}) \\
&\qquad A = \mathit{freshInstance}\ \mathit{typeof}\ \alpha\ (S_Q\ \Delta_P) \\
&\qquad B = (S_Q\ \varphi) \\
&\qquad C = \mathit{freshInstance}\ \mathit{typeof}\ x\ (S_Q \circ S_P\ \Gamma) \\
&\qquad S_u = \mathit{unify}\ C\ A \rightarrow B \\
&\qquad \langle S_c, \Delta_c\rangle = \mathit{unifyGenContexts}\ (S_u \circ S_Q\ \Delta_P \backslash \alpha)\ (S_u\ \Delta_Q) \\
&\qquad S_r = (S_c \circ S_u \circ S_Q \circ S_P \cap \mathit{atoms}(\Gamma)) \\
&\mathit{sppc}\,(P\widehat{\alpha} \dagger \widehat{x}Q, \Gamma) = \langle S_r, \Delta_c\rangle \\
&\quad \text{where} \\
&\qquad \langle S_P, \Delta_P\rangle = \mathit{sppc}\,(P, \Gamma) \\
&\qquad \overline{A} = \mathit{typeof}\ \alpha\ \Delta_P \\
&\qquad \langle S_Q, \Delta_Q\rangle = \mathit{sppc}\,(Q, (S_P\ \Gamma) \cup \{x : \overline{A}\}) \\
&\qquad \langle S_c, \Delta_c\rangle = \mathit{unifyGenContexts}\ (S_Q\ \Delta_P \backslash \alpha)\ \Delta_Q \\
&\qquad S_r = (S_c \circ S_Q \circ S_P \cap \mathit{atoms}(\Gamma))
\end{aligned}
$$

Figure 2: Principal Contexts for Shallow Polymorphic system

Theorem 3.29 (Soundness and Completeness of *sppc*)**.** *Given an $\mathcal{X}^i$-term R and an initial left-context Γ such that*

$$fs(R) \subseteq dom(\Gamma) \tag{3}$$

we have:

1. *If* $\mathrm{sppc}\,(R, \Gamma)$ *succeeds and* $\mathrm{sppc}\,(R, \Gamma) = \langle S_R, \Delta_R \rangle$ *then* $R :\cdot (S_R\ \Gamma) \vdash_{\mathrm{SP}} \Delta_R$.
2. *If there exist* $\langle S, \Delta \rangle$ *such that* $R :\cdot (S\ \Gamma) \vdash_{\mathrm{SP}} \Delta$*, then a call* $\mathrm{sppc}\,(R, \Gamma)$ *succeeds, and if* $\mathrm{sppc}\,(R, \Gamma) = \langle S_R, \Delta_R \rangle$ *then there exists a further substitution* S' *such that* $S = S' \circ S_R$ *and* $(S'\ \Delta_R) \geq \Delta$.

Proof. See Proof B.6 in Section B. □

4. Extensions and Future Work

Since classical sequent calculus exhibits a natural symmetry between left and right contexts (inputs and outputs, in a computational sense), it is natural to consider the asymmetric notion of (universal) polymorphism presented so far as an incomplete picture. Universal polymorphism allows an output (plug) type to be generalised with quantified variables, and then to be connected to multiple input types, each taking a different instantiation of the variables. What then, if we allow this the opposite way around? It seems natural to consider the generalisation of an *input* type, to be instantiated many times for the multiple *outputs* it is connected with.

All of the work presented in this paper can be adapted analogously for this alternative quantifier. The resulting type system is sound, although the analogous naïve system would not be (in this case, it is *right* propagation that presents a potential for unsoundness, but this is eliminated above by an analogous restriction. We can also define a principal typings algorithm, by 'reflecting' the definitions employed in the previous section. In particular, such an algorithm would type a cut by typing first the *right*-hand subterm, and then using the (potentially existentially-quantified) type obtained to help type the left.

Existential polymorphism is traditionally understood in the context of *information hiding* [31], i.e., providing a facility to *lose* typing information from a term, rather than providing extra power in terms of typeability. However, this is a question of paradigm: in a traditional functional setting, corresponding with minimal logic (such as the λ-calculus), the addition of existential quantification does not extend the typeable terms, while the addition of universal quantification does.

This can be readily understood by moving again to the sequent calculus setting; when injecting ML (for example) into $\mathcal{X}^i$, via the translation above, one always obtains a term in which there is exactly one free plug, and exactly one occurrence of the plug. Therefore, the additional power in terms of typeability which existential polymorphism brings, is not applicable, since it caters for the situation when multiple occurrences of the same plug need to be typed in different ways.

To summarise, in the setting of classical sequent calculus, the two variants of polymorphism can be seen exactly as dual to one another; universal polymorphism allowing generalisation of outputs and instantiation at multiple inputs, and vice versa for existential. A possible and natural extension to this work which has not been investigated in depth is the possibility of allowing *both* kinds of quantification to be exploited in a shallow polymorphic type system. Since the cuts in the $\mathcal{X}^i$-calculus can simultaneously bind multiple occurrences of *both* inputs and outputs, it seems reasonable that there may be example terms which would be made typeable by such a system.

The main problem envisaged with such a system is decideable type-assignment. In particular, the presented approach to typing a cut seems not to adapt to this setting. In the case of universal polymorphism, a cut is typed by typing the left-hand subterm first, and using the information gained to help type the right. The reverse ordering of subcalls is suitable for a system with existential polymorphism. But with both quantifiers permitted, there is no obvious approach; it may be that *each* subterm provides some polymorphic behaviour which allows to overcome difficulties in typing the other subterm. We leave such issues for future work.

5. Conclusions

This paper has been concerned with the adaptation of ML-style shallow polymorphism to the context of a term calculus based on classical logic. We have shown that the problem is not straightforward, and the 'natural' approach is unsound. The exact nature of the problem is made particularly clear in the context of the sequent calculus, and we identified three contributing features of the type system which together caused it to arise. By pinpointing the problem, we were able to define a neat refinement to the type system, and prove a witness reduction result.

The question of principal typings was made more challenging by the restrictions in place in our amended type system. Because quantified types for outputs (plugs) need to be introduced "early" in our type system, it became necessary to deal with the case of many different quantified types being derived for different

occurrences of the same output. Thus created the need for an operation to calculate the most general generic type "smaller" than two generic types, which we dubbed generic unification. Although the basic idea behind generic unification is simple, the need to restrict the resulting substitutions is less obvious.

Having defined generic unification, and proved its soundness and completeness with respect to the generic instance and substitution operations employed within the type system, we were able to define a notion of principal contexts, and an algorithm to compute them. Our proof that such principal contexts are indeed principal (given an initial specification of polymorphism to be assumed for free sockets), makes our result analogous with the classical result for ML, but in the context of a calculus with a more general type system and reduction behaviour. This was the ultimate goal of the paper.

Acknowledgements

This work would not have been possible without the supervision and extensive support of Steffen van Bakel. We would also like to thank Luca Cardelli, Jayshan Raghunandan, Hugo Herbelin, Gavin Bierman and Mariangiola Dezani for their helpful discussions and comments on previous versions of this work. Thanks also to the anonymous referees - this paper has been greatly improved by their detailed suggestions.

A. Examples of *sppc* algorithm

Consider the following $\mathcal{X}^i$-terms:

$$\begin{aligned} M &= \widehat{x}\langle x.\alpha\rangle\widehat{\alpha}\cdot\beta \\ N &= M\widehat{\epsilon}\,[i]\,\widehat{j}M \\ O &= \widehat{x}(\widehat{y}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma)\widehat{\beta}\cdot\gamma \\ R &= \langle z.\delta\rangle\widehat{\delta}\,[z]\,\widehat{w}\langle w.\epsilon\rangle \end{aligned}$$

Then we have:

$$
\begin{aligned}
sppc\,(M,\Gamma) &= \langle id, \{\beta : \forall X.(X\to X)\}\rangle \textit{ for any context } \Gamma \\
sppc\,(N,\{i : E\to F\}) &= \langle id, \{\beta : \forall Z.(Z\to Z)\}\rangle \textit{ for any Curry types } E, F \\
sppc\,(O,\Gamma) &= \langle id, \{\gamma : \varphi\to\varphi\}\rangle \textit{ where } \varphi \textit{ is fresh, for any context } \Gamma \\
sppc\,(R,\{z : E\to E\}) & \quad \textit{fails, for any Curry type } E \\
sppc\,(R,\{z : \forall X.(X\to X)\}) &= \langle id, \{\gamma : \varphi\to\varphi\}\rangle \textit{ where } \varphi \textit{ is fresh} \\
sppc\,(M\widehat{\beta} \dagger \widehat{z}R, \emptyset) &= \langle id, \{\gamma : \varphi\to\varphi\}\rangle \textit{ where } \varphi \textit{ is fresh} \\
sppc\,(N\widehat{\beta} \dagger \widehat{z}R, \{i : E\to F\}) &= \langle id, \{\gamma : \varphi\to\varphi\}\rangle \textit{ where } \varphi \textit{ is fresh} \\
sppc\,(O\widehat{\beta} \dagger \widehat{z}R, \Gamma) & \quad \textit{fails, for any } \Gamma
\end{aligned}
$$

If we apply our type inference algorithm to M (with any context parameter Γ - since there are no free sockets in M it will not be used), the algorithm operates as follows:

$$
\begin{aligned}
&\textit{calculate } sppc\,(\widehat{x}\langle x.\alpha\rangle\widehat{\alpha}\cdot\beta, \Gamma)\textit{:} \\
&\varphi_1 = \textit{fresh} \\
&\langle S_P, \Delta_P\rangle = sppc\,(\langle x.\alpha\rangle, \{x : \varphi_1\}) = \langle id, \{\alpha : \varphi_1\}\rangle \\
&A = (S_P\ \varphi_1) = \varphi_1 \\
&B = \textit{freshInstance}\ (\textit{typeof}\ \alpha\ \Delta_P) = \varphi_1 \\
&\overline{C} = \forall\text{-}\textit{closure}\ A\to B\ \langle (S_P\ \Gamma); \Delta_P\backslash\alpha\rangle = \forall\text{-}\textit{closure}\ \varphi_1\to\varphi_1\ \langle \Gamma; \emptyset\rangle = \forall X.(X\to X) \\
&\langle S_u, \overline{D}\rangle = \langle id, \overline{C}\rangle = \langle id, \forall X.(X\to X)\rangle \\
&S_r = (S_u \circ S_P \cap atoms(\Gamma)) = (id \cap atoms(\Gamma)) = id \\
&sppc\,(M,\Gamma) = \langle S_r, (S_u\ \Delta_P\backslash\alpha\backslash\beta)\cup\{\beta : \overline{D}\}\rangle = \langle id, \{\beta : \forall X.(X\to X)\}\rangle
\end{aligned}
$$

Now let us consider applying the algorithm to N, using a context $\{i : E\to F\}$ for some types E and F (their choice is unimportant since we are not interested in the

socket i for this example):

$$
\begin{array}{l}
\textit{calculate sppc}\,(M\widehat{\epsilon}\,[i]\,\widehat{j}M, \{i : E \rightarrow F\})\textit{:} \\
\langle S_P, \Delta_P \rangle = \textit{sppc}\,(M, \{i : E\}) = \langle \textit{id}, \{\beta : \forall X.(X \rightarrow X)\}\rangle \ \textit{(by above)} \\
\varphi_2 = \textit{fresh} \\
\langle S_Q, \Delta_Q \rangle = \textit{sppc}\,(M, (S_P\,\{i : E\}) \cup \{i : \varphi_2\}) = \langle \textit{id}, \{\beta : \forall Y.(Y \rightarrow Y)\}\rangle \ \textit{(by above)} \\
A = \textit{freshInstance}\ (\textit{typeof}\ \epsilon\ S_Q\ \Delta_P) = \varphi_3\ \textit{(fresh)} \\
B = (S_Q\ \varphi_2) = \varphi_2 \\
C = \textit{freshInstance}\ (\textit{typeof}\ i\ S_Q \circ S_P\ \{i : E \rightarrow F\}) = E \rightarrow F \\
S_u = \textit{unify}\ C\ A \rightarrow B = \textit{unify}\ E \rightarrow F\ \varphi_3 \rightarrow \varphi_2 = \{(\varphi_3 \mapsto E), (\varphi_2 \mapsto F)\} \\
\langle S_c, \Delta_c \rangle = \textit{unifyGenContexts}\ (S_u \circ S_Q\ \Delta_P \backslash \epsilon)\ (S_u\ \Delta_Q) \\
= \textit{unifyGenContexts}\ \{\beta : \forall X.(X \rightarrow X)\}\ \{\beta : \forall Y.(Y \rightarrow Y)\} \\
\quad \textit{calculate unifyGen}\ \forall X.(X \rightarrow X)\ \forall Y.(Y \rightarrow Y)\textit{:} \\
\quad A' = \textit{freshInstance}\ \forall X.(X \rightarrow X) = \varphi_4 \rightarrow \varphi_4 \\
\quad B' = \textit{freshInstance}\ \forall Y.(Y \rightarrow Y) = \varphi_5 \rightarrow \varphi_5 \\
\quad S'_u = \textit{unify}\ A'\ B' = \{(\varphi_4 \mapsto \varphi_5)\} \\
\quad C_u = (S'_u\ A') = \varphi_5 \rightarrow \varphi_5 \\
\quad \overrightarrow{\varphi_i} = \textit{atoms}(C_u) \backslash (\textit{atoms}(S'_u\ \forall X.(X \rightarrow X)) \cup \textit{atoms}(S'_u\ \forall Y.(Y \rightarrow Y))) = \{\varphi_5\} \\
\quad S'_r = (S'_u \cap (\textit{atoms}(\forall X.(X \rightarrow X)) \cup \textit{atoms}(\forall Y.(Y \rightarrow Y)))) = (S'_u \cap \emptyset) = \textit{id} \\
\quad \textit{unifyGen}\ \forall X.(X \rightarrow X)\ \forall Y.(Y \rightarrow Y) = \langle \textit{id}, \forall \overrightarrow{Z_i}.C_u[\overrightarrow{Z_i/\varphi_i}]\rangle = \langle \textit{id}, \forall Z.(Z \rightarrow Z)\rangle \\
\langle S_c, \Delta_c \rangle = \langle \textit{id}, \{\beta : \forall Z.(Z \rightarrow Z)\}\rangle \\
S_r = (S_c \circ S_u \circ S_Q \circ S_P \cap \textit{atoms}(\{i : E \rightarrow F\})) \\
= (\{(\varphi_3 \mapsto E), (\varphi_2 \mapsto F)\} \cap \textit{atoms}((E \rightarrow F))) = \textit{id}\ \textit{(freshness of}\ \varphi_2, \varphi_3\textit{)} \\
\textit{sppc}\,(M\widehat{\epsilon}\,[i]\,\widehat{j}M, \{i : E \rightarrow F\}) = \langle S_r, \Delta_c \rangle = \langle \textit{id}, \{\beta : \forall Z.(Z \rightarrow Z)\}\rangle
\end{array}
$$

B. Proofs

Proof B.1 (of Proposition 3.6)**.**

1. *Reflexivity is immediate. For transitivity, suppose that* $\forall \overrightarrow{X_i}.A \leq \forall \overrightarrow{Y_j}.B$ *and* $\forall \overrightarrow{Y_j}.B \leq \forall \overrightarrow{Z_k}.C$*. By definition, we have for some types* $\overrightarrow{D_i}, \overrightarrow{E_j}$ *and atomic types* $\overrightarrow{\varphi_j} \notin A$ *and* $\overrightarrow{\varphi'_k} \notin B$ *that* $B = A[\overrightarrow{D_i/X_i}]\,[\overrightarrow{Y_j/\varphi_j}]$ *and* $C = B[\overrightarrow{E_j/Y_j}]\,[\overrightarrow{Z_k/\varphi'_k}]$*. Composing these two facts, we have that* $C = A[\overrightarrow{D_i/X_i}]\,[\overrightarrow{Y_j/\varphi_j}]\,[\overrightarrow{E_j/Y_j}]\,[\overrightarrow{Z_k/\varphi'_k}]$*. Let* S *be the substitution* $\{(\overrightarrow{\varphi_j \mapsto E_j})\}$*. Then note that since* $\varphi_j \notin A$*, we know*

that $(S\ A[\overrightarrow{D_i/X_i}]) = A[\overrightarrow{(S\ D_i)/X_i}]$ *. Then we have:*

$$\begin{aligned} C &= A[\overrightarrow{D_i/X_i}]\,[\overrightarrow{Y_j/\varphi_j}]\,[\overrightarrow{E_j/Y_j}]\,[\overrightarrow{Z_k/\varphi'_k}] \\ &= A[\overrightarrow{D_i/X_i}]\,[\overrightarrow{E_j/\varphi_j}]\,[\overrightarrow{Z_k/\varphi'_k}] \\ &= (S\ A[\overrightarrow{D_i/X_i}])[\overrightarrow{Z_k/\varphi'_k}] \\ &= A[\overrightarrow{(S\ D_i)/X_i}]\,[\overrightarrow{Z_k/\varphi'_k}] \end{aligned}$$

Finally, we can see that $\overrightarrow{\varphi'_k} \notin A$ *since if it were the case that* $\varphi'_k \in A$ *then since* $\overrightarrow{\varphi_j} \notin A$ *we would have* $\varphi'_k \in A[\overrightarrow{D_i/X_i}]\,[\overrightarrow{Y_j/\varphi_j}] = B$*; a contradiction.*

2. *Reflexivity and transivity are straightforward. Anti-symmetry follows from the fact that if* $\overline{A} \lhd_{\langle\Gamma;\Delta\rangle} \overline{B}$ *and* $\overline{A} \neq \overline{B}$ *then* $\overline{B}$ *contains strictly more* $\forall$ *symbols than* $\overline{A}$*.*
3. *Let* $\overline{A} = \forall\overrightarrow{X_i}.A$*. Then, since* $\overline{A} \geq \overline{B}$*, we know that for some* $\overrightarrow{D_i}, \overrightarrow{Y_j}, \overrightarrow{\varphi_j}$ *we must have* $\overline{B} = \overrightarrow{\forall Y_j}.(A[\overrightarrow{D_i/X_i}]\,[\overrightarrow{Y_j/\varphi_j}])$*, with* $\overrightarrow{\varphi_j} \notin A$*. In addition, since* $\overline{B} \lhd_{\langle\Gamma;\Delta\rangle} \overline{C}$*, we must have for some* $\overrightarrow{Z_k}$ *and* $\overrightarrow{\varphi'_k} \notin \langle\Gamma;\Delta\rangle$ *that* $\overline{C} = \overrightarrow{\forall Z_k}.\overline{B}[\overrightarrow{Z_k/\varphi'_k}]$*. Since* $\overrightarrow{\varphi'_k} \notin \langle\Gamma;\Delta\rangle$ *and* $\overline{A} \in \langle\Gamma;\Delta\rangle$*, we have* $\overrightarrow{\varphi'_k} \notin \overline{A}$*. From these facts, we can conclude that* $\overline{C} = \overrightarrow{\forall Z_k}.\overrightarrow{\forall Y_j}.(A[\overrightarrow{D_i/X_i}]\,[\overrightarrow{Y_j/\varphi_j}]\,[\overrightarrow{Z_k/\varphi'_k}])$ *with* $\overrightarrow{\varphi_j}, \overrightarrow{\varphi'_k} \notin \overline{A}$*, i.e.* $\overline{C} \geq \overline{A}$ *as required.*
4. *Without loss of generality, say* $\overline{A} = \overrightarrow{\forall X_i}.A$ *and* $\overline{B} = \overrightarrow{\forall Y_j}.(A[\overrightarrow{C_i/X_i}]\,[\overrightarrow{Y_j/\varphi_j}])$ *(with* $\varphi_j \notin \overline{A}$*). Note that* $\varphi_j \notin \overline{B}$ *also, due to the renaming* $[\overrightarrow{Y_j/\varphi_j}]$*. Let* $S' = (S \cap \{\overrightarrow{\varphi_j}\})$*. Then we have* $(S'\ \overline{A}) = (S\ \overline{A})$ *and* $(S'\ \overline{B}) = (S\ \overline{B})$*. Therefore, it suffices to prove that* $(S'\ \overline{A}) \geq (S'\ \overline{B})$*. This can be seen from the fact that* $(S\ \overline{A}) = \overrightarrow{\forall X_i}.(S\ A)$ *and the following working:*

$$\begin{aligned} (S'\ \overline{B}) &= (S'\ \overrightarrow{\forall Y_j}.(A[\overrightarrow{C_i/X_i}]\,[\overrightarrow{Y_j/\varphi_j}])) && \textit{defn of } \overline{B} \\ &= \overrightarrow{\forall Y_j}.(S'\ (A[\overrightarrow{C_i/X_i}]\,[\overrightarrow{Y_j/\varphi_j}])) && \textit{defn of substitution} \\ &= \overrightarrow{\forall Y_j}.((S'\ A[\overrightarrow{C_i/X_i}])[\overrightarrow{Y_j/\varphi_j}]) && \textit{defn of } S' \\ &= \overrightarrow{\forall Y_j}.((S'\ A)[\overrightarrow{(S'\ C_i)/X_i}]\,[\overrightarrow{Y_j/\varphi_j}]) \\ &= \overrightarrow{\forall Y_j}.((S'\ A)[\overrightarrow{D_i/X_i}]\,[\overrightarrow{Y_j/\varphi_j}]) && \textit{setting } \overrightarrow{D_i = (S'\ C_i)} \end{aligned}$$

5. *By definition, there exist* $\overrightarrow{\varphi_i}$ *and* $\overrightarrow{X_i}$ *such that* $\overline{B} = \overrightarrow{\forall X_i}.\overline{A}[\overrightarrow{X_i/\varphi_i}]$ *and* $\overrightarrow{\varphi_i \notin \langle\Gamma;\Delta\rangle}$ *Define* $S' = \{\overrightarrow{(\varphi_i \mapsto \varphi'_i)}\}$*, where* $\overrightarrow{\varphi'_i}$ *are fresh atomic types. Then we know that* $\overline{B} = \overrightarrow{\forall X_i}.(S'\ \overline{A})[\overrightarrow{X_i/\varphi'_i}]$*. Therefore,* $(S\ \overline{B}) = (S\ (\overrightarrow{\forall X_i}.(S'\ \overline{A})[\overrightarrow{X_i/\varphi'_i}])) = \overrightarrow{\forall X_i}.(S \circ S'\ \overline{A})[\overrightarrow{X_i/\varphi'_i}]$*. By construction,* $\overrightarrow{\varphi'_i \notin \langle(S\ \Gamma);(S\ \Delta)\rangle}$ *, and so we conclude* $(S \circ S'\ \overline{A}) \lhd_{\langle(S\ \Gamma);(S\ \Delta)\rangle} (S\ \overline{B})$ *as required.*
6. *By definition, for some* $\overrightarrow{X_i}$ *and* $\overrightarrow{\varphi_i} \notin \langle\Gamma;\Delta\rangle$*, we must have* $\overline{A} = \overrightarrow{\forall X_i}.A[\overrightarrow{X_i/\varphi_i}]$*. Additionally, since* $\overline{A} \geq \overline{B}$*, there must exist* $\overrightarrow{C_i}$ *and* $\overrightarrow{\varphi_j}$ *and* $\overrightarrow{Y_j}$ *such that we can obtain* $\overline{B} = \overrightarrow{\forall Y_j}.(A[\overrightarrow{X_i/\varphi_i}])[\overrightarrow{C_i/X_i}]\,[\overrightarrow{Y_j/\varphi_j}]$ *and* $\varphi_j \notin \overline{A}$*. Let* S *be the*

substitution $\{(\overrightarrow{\varphi_i \mapsto C_i})\}$. *Then* $\overline{B} = \overrightarrow{\forall Y_j}.(S\ A)[\overrightarrow{Y_j/\varphi_j}]$. *Let* $S' = \{(\overrightarrow{\varphi_j \mapsto \varphi'_j})\}$ *where* $\overrightarrow{\varphi'_j}$ *are fresh. Then* $\overline{B} = \overrightarrow{\forall Y_j}.(S' \circ S\ A)[\overrightarrow{Y_j/\varphi'_j}]$ *and* $\overrightarrow{\varphi' \notin \langle\Gamma;\Delta\rangle}$, *i.e.,* $(S' \circ S\ A) \lhd_{\langle\Gamma;\Delta\rangle} \overline{B}$. *Finally, since* $\overrightarrow{\varphi_i} \notin \langle\Gamma;\Delta\rangle$, *we have* $S\ \langle\Gamma;\Delta\rangle = \langle\Gamma;\Delta\rangle$ *as required.*

7. *Since* $\overline{A} \geq B$, *there must exist* $\overrightarrow{C_i}$ *such that* $B = A[\overrightarrow{C_i/X_i}]$. *Let* $S = \{(\overrightarrow{\varphi_i \mapsto C_i})\}$. *Then the result immediately follows.*

Proof B.2 (of Proposition 3.8). 1. *By induction on the structure of the term P. We give two representative cases (all others are simpler):*

$\langle x.\alpha\rangle$**:** *Then by Lemma 3.9 1,* $\Gamma = \Gamma', x:\overline{A}$ *and* $\Delta = \alpha:\overline{B}, \Delta'$ *with* $\overline{A} \geq \overline{B}$. *By Proposition 3.6 4,* $(S\ \overline{A}) \geq (S\ \overline{B})$. *Therefore, by applying the rule* (*ax*), *we obtain* $\langle x.\alpha\rangle :\cdot (S\ \Gamma'), x:(S\ \overline{A}) \vdash_{\mathrm{SP}} \alpha:(S\ \overline{B}), \Delta'$ *as required.*

$\widehat{xP\alpha}\cdot\beta$**:** *Then by Lemma 3.9 2,* $\Delta = \beta:\overline{C}, \Delta'$ *and there exist A,B such that*

$$P :\cdot \Gamma, x:A \vdash_{\mathrm{SP}} \alpha:B, \Delta' \tag{4}$$

and $(A \rightarrow B) \lhd_{\langle\Gamma;\Delta'\rangle} \overline{C}$. *By Proposition 3.6 5, there exists a substitution* S' *such that:*

$$(S \circ S'\ (A \rightarrow B)) \lhd_{\langle(S\ \Gamma);(S\ \Delta')\rangle} (S\ \overline{C}) \tag{5}$$
$$(S'\ \langle\Gamma;\Delta\rangle) = \langle\Gamma;\Delta\rangle \tag{6}$$
$$(S'\ \overline{C}) = \overline{C} \tag{7}$$

By induction, using (*Eq.* 4) *with the substitution* $(S \circ S')$, *we obtain* $P :\cdot (S \circ S'\ \Gamma), x:(S \circ S'\ A) \vdash_{\mathrm{SP}} \alpha:(S \circ S'\ B), (S \circ S'\ \Delta')$. *Using* (*Eq.* 6) *and* (*Eq.* 7), *this becomes* $P :\cdot (S\ \Gamma), x:(S \circ S'\ A) \vdash_{\mathrm{SP}} \alpha:(S \circ S'\ B), (S\ \Delta')$. *Furthermore, noting that* $(S \circ S'\ (A \rightarrow B)) = (S \circ S'\ A) \rightarrow (S \circ S'\ B)$, *we can apply* $(\rightarrow\mathcal{R})$ *with* (*Eq.* 5) *to obtain* $\widehat{xP\alpha}\cdot\beta :\cdot (S\ \Gamma) \vdash_{\mathrm{SP}} \beta:(S\ \overline{C}), (S\ \Delta')$ *as required.*

2. *By induction on the structure of the term P. The only cases which are not straightforward are when 'closures' are taken, since we must be careful that the appropriate conditions can still be fulfilled within the larger context. This situation is exemplified by the case of a term* $\widehat{xP\alpha}\cdot\beta$, *and this is the only case we show here. As usual, by Lemma 3.9 2, we obtain the statements* $P :\cdot \Gamma, x:A \vdash_{\mathrm{SP}} \alpha:B, \Delta''$ *with* $\Delta = \beta:\overline{C}, \Delta''$ *and* $(A \rightarrow B) \lhd_{\langle\Gamma;\Delta''\rangle} \overline{C}$. *By unravelling the definition, we know that* $\overline{C} = \overrightarrow{\forall X_i}.(A \rightarrow B)[\overrightarrow{X_i/\varphi_i}]$, *for some* $\overrightarrow{X_i}$ *and some* $\overrightarrow{\varphi_i \notin \langle\Gamma;\Delta''\rangle}$. *In order to ensure that we can still 'close' the*

type in the larger context, we rename these atomic types: define the substitution $S = \{(\overline{\varphi_i \mapsto \varphi_i'})\}$ *for fresh* φ_i'*. Note that* $(S\ \Gamma) = \Gamma$ *and* $(S\ \Delta'') = \Delta''$*. Then, by part 1, we obtain* $P :\cdot\ \Gamma, x:(S\ A) \vdash_{\mathrm{SP}} \alpha:(S\ B), \Delta''$*. Since* x *and* α *are bound in the original term, we may assume that* $x \notin \Gamma'$ *and* $\alpha \notin \Delta'$*. Therefore,* $\langle \Gamma, x:(S\ A), \Gamma'; \alpha:(S\ B), \Delta'', \Delta' \rangle$ *is a well-formed context. By induction,* $P :\cdot\ \Gamma, x:(S\ A), \Gamma' \vdash_{\mathrm{SP}} \alpha:(S\ B), \Delta'', \Delta'$*. In order to be able to apply the* $(\rightarrow\mathcal{R})$ *and conclude, it would suffice to show that* $(S\ A{\rightarrow}B) \lhd_{\langle \Gamma\cup\Gamma';\Delta\cup\Delta'\rangle} \overline{C}$*. But this follows by construction of* S*.*

3. *By straightforward induction on the structure of the derivation.*
4. *By induction on the structure of the term. We present two representative cases.*

   $P = \langle x.\alpha \rangle$**:** *By Lemma 3.9 1, there exist* X, Δ' *such that* $\Delta = \alpha:\overline{C}, \Delta'$ *and* $\overline{B} \geq \overline{C}$*. By Proposition 3.6 1 we have* $\overline{A} \geq \overline{C}$*. Therefore, by applying the rule* (ax)*, we obtain* $\langle x.\alpha \rangle :\cdot\ (\Gamma\backslash x), x:\overline{A} \vdash_{\mathrm{SP}} \alpha:\overline{C}, \Delta$ *as required.*

   $P = \langle y.\alpha \rangle$**,** $y \neq x$**:** *This case is immediate from Lemma 3.9 1.*

   $P = \widehat{y}P_1\widehat{\alpha}{\cdot}\beta$**:** *By straightforward induction, using Lemma 3.9 2.*

   $P = Q\widehat{\alpha}[x]\widehat{y}R$**:** *By Lemma 3.9 3 and Proposition 3.8 2, there exist C,D such that* $Q :\cdot\ \Gamma, x:\overline{B} \vdash_{\mathrm{SP}} \alpha:\overline{C}, \Delta$ *and* $R :\cdot\ \Gamma, x:\overline{B}, y:D \vdash_{\mathrm{SP}} \Delta$ *and* $\overline{B} \geq (C{\rightarrow}D)$*. By induction,* $Q :\cdot\ (\Gamma\backslash x), x:\overline{A} \vdash_{\mathrm{SP}} \alpha:\overline{C}, \Delta$ *and* $R :\cdot\ (\Gamma\backslash x), x:\overline{A}, y:D \vdash_{\mathrm{SP}} \Delta$*. By Proposition 3.6 1, we have* $\overline{A} \geq (C{\rightarrow}D)$*. By the rule* $(\rightarrow\mathcal{L})$*, we obtain* $P :\cdot\ (\Gamma\backslash x), x:\overline{A} \vdash_{\mathrm{SP}} \Delta$ *as required.*

5. *By induction on the structure of the term P. We present two representative cases.*

   $P = \langle x.\alpha \rangle$**:** *By Lemma 3.9 1, there exist* $\overline{C}, \Gamma'$ *such that* $\Gamma = \Gamma', x:\overline{C}$ *and* $\overline{C} \geq \overline{B}$*. By Proposition 3.6 1,* $\overline{C} \geq \overline{A}$*. Therefore, by the rule* (ax)*, we can deduce that* $\langle x.\alpha \rangle :\cdot\ \Gamma', x:\overline{C} \vdash_{\mathrm{SP}} \alpha:\overline{A}, (\Delta\backslash\alpha)$ *as required.*

   $P = \widehat{x}Q\widehat{\beta}{\cdot}\alpha$**:** *By Lemma 3.9 2, there exist C,D such that* $(C{\rightarrow}D) \lhd_{\langle\Gamma;\Delta\rangle} \overline{A}$ *and* $Q :\cdot\ \Gamma, x:C \vdash_{\mathrm{SP}} \beta:D, \Delta$*. By Proposition 3.6 6, there exists a substitution* S *such that* $(S\ \langle\Gamma;\Delta\rangle) = \langle\Gamma;\Delta\rangle$ *and* $(S\ C{\rightarrow}D) \lhd_{\langle\Gamma;\Delta\rangle} \overline{B}$*. By Proposition 3.8 1 we have* $Q :\cdot\ \Gamma, x:(S\ C) \vdash_{\mathrm{SP}} \beta:(S\ D), \Delta$*. We consider two cases:*

   $\alpha:\overline{B} \in \Delta$**:** *Then, by induction,* $Q :\cdot\ \Gamma, x:(S\ C) \vdash_{\mathrm{SP}} \beta:(S\ D), \alpha:\overline{B}, (\Delta\backslash\alpha)$*. By the rule* $(\rightarrow\mathcal{R})$*, we obtain* $\widehat{x}Q\widehat{\beta}{\cdot}\alpha :\cdot\ \Gamma \vdash_{\mathrm{SP}} \alpha:\overline{B}, (\Delta\backslash\alpha)$ *as required.*

   $\alpha \notin \Delta$**:** *Then, by the* $(\rightarrow\mathcal{R})$ *rule, we obtain* $\widehat{x}Q\widehat{\beta}{\cdot}\alpha :\cdot\ \Gamma \vdash_{\mathrm{SP}} \alpha:\overline{B}, \Delta$ *as required.*

6. *By induction on the structure of the term P, similar to the previous part.*

Proof B.3 (of Theorem 3.10). 1. (a) *By induction on the structure of the term Q.*

$Q = \langle x.\beta\rangle$**:** *Then* $Q\{P\widehat{\alpha}\leftrightarrow x\} = P\widehat{\alpha} \dagger \widehat{x}Q$ *and the result follows by application of the* (*cut*) *rule.*

$Q = \langle y.\beta\rangle, y \neq x$**:** *Then* $Q\{P\widehat{\alpha}\leftrightarrow x\} = Q$. *Since* $x \notin fs(Q)$, *By* (*Eq.* 2) *and Proposition 3.8 3 we obtain* $Q :\cdot \Gamma \vdash_{\text{SP}} \Delta$ *as required.*

$Q = \widehat{y}Q_1\widehat{\beta}\cdot\gamma$**:** *Then* $Q\{P\widehat{\alpha}\leftrightarrow x\} = \widehat{y}(Q_1\{P\widehat{\alpha}\leftrightarrow x\})\widehat{\beta}\cdot\gamma$. *By* (*Eq.* 2) *and Lemma 3.9 2, there exist* $B,C,\overline{D}$ *and* Δ' *such that* $\Delta = \Delta', \gamma : \overline{D}$ *and* $Q_1 :\cdot \Gamma, x : \overline{A}, y : B \vdash_{\text{SP}} \beta : C, \Delta$ *and* $(B{\rightarrow}C) \lhd_{\langle\Gamma,x:\overline{A};\Delta'\rangle} \overline{D}$. *From the induction hypothesis,* $Q_1\{P\widehat{\alpha}\leftrightarrow x\} :\cdot \Gamma, y : B \vdash_{\text{SP}} \beta : C, \Delta$. *Now we apply* $(\rightarrow\mathcal{R})$ *rule to obtain* $\widehat{y}(Q_1\{P\widehat{\alpha}\leftrightarrow x\})\widehat{\beta}\cdot\gamma :\cdot \Gamma \vdash_{\text{SP}} \gamma : \overline{D}, \Delta'$ *as required.*

$Q = Q_1\widehat{\beta}\,[x]\,\widehat{z}Q_2$**:**
$Q\{P\widehat{\alpha}\leftrightarrow x\} = P\widehat{\alpha} \dagger \widehat{y}((Q_1\{P\widehat{\alpha}\leftrightarrow x\})\widehat{\beta}\,[y]\,\widehat{z}(Q_2\{P\widehat{\alpha}\leftrightarrow x\}))$ *in which* y *is fresh. By* (*Eq.* 2) *and Lemma 3.9 3, there exist* $B,C,\overline{D},\Gamma'$ *such that* $\overline{D}{\geq}(B{\rightarrow}C)$ *and* $\Gamma = \Gamma', y : \overline{D}$ *and (by weakening, by applying Proposition 3.8 2 where necessary) both* $Q_1 :\cdot \Gamma, x : \overline{A} \vdash_{\text{SP}} \beta : B, \Delta$ *and* $Q_2 :\cdot \Gamma, x : \overline{A}, z : C \vdash_{\text{SP}} \Delta$. *By induction, twice, we obtain that both* $Q_1\{P\widehat{\alpha}\leftrightarrow x\} :\cdot \Gamma \vdash_{\text{SP}} \beta : B, \Delta$ *and also* $Q_2\{P\widehat{\alpha}\leftrightarrow x\} :\cdot \Gamma, z : C \vdash_{\text{SP}} \Delta$. *Since* $\overline{D}{\geq}(B{\rightarrow}C)$, *we can apply the* $(\rightarrow\mathcal{L})$ *rule to obtain the judgement* $(Q_1\{P\widehat{\alpha}\leftrightarrow x\})\widehat{\beta}\,[y]\,\widehat{z}(Q_2\{P\widehat{\alpha}\leftrightarrow x\}) :\cdot \Gamma, y : \overline{D} \vdash_{\text{SP}} \Delta$. *Finally, we apply the* (*cut*) *rule to obtain the required result.*

$Q = Q_1\widehat{\beta}\,[y]\,\widehat{z}Q_2, y \neq x$**:** *By straightforward induction, similar to the previous case.*

$Q = \langle x.\beta\rangle\widehat{\beta} \dagger \widehat{y}Q_2$**:** *Then* $Q\{P\widehat{\alpha}\leftrightarrow x\} = P\widehat{\alpha} \dagger \widehat{y}(Q_1\{P\widehat{\alpha}\leftrightarrow x\})$. *By Lemma 3.9 4, there exists* $\overline{B}$ *such that both*

$$\langle x.\beta\rangle :\cdot \Gamma, x : \overline{A} \vdash_{\text{SP}} \beta : \overline{B}, \Delta \tag{8}$$

$$Q_1 :\cdot \Gamma, x : \overline{A}, y : \overline{B} \vdash_{\text{SP}} \Delta \tag{9}$$

By Lemma 3.9 1, we must have

$$\overline{A}{\geq}\overline{B} \tag{10}$$

By applying Proposition 3.8 2 to (*Eq.* 8), *we obtain*

$$P :\cdot \Gamma, y : \overline{B} \vdash_{\text{SP}} \alpha : \overline{A}, \Delta \tag{11}$$

By applying induction to (*Eq.* 2) *and* (*Eq.* 11), *we obtain*

$$Q_1\{P\widehat{\alpha}\leftrightarrow x\} :\cdot \Gamma, y : \overline{B} \vdash_{\text{SP}} \Delta \tag{12}$$

By applying Proposition 3.8 4 to (*Eq.* 10) *and* (*Eq.* 12), *we obtain*

$$Q_1\{P\widehat{\alpha}\leftrightarrow x\} :\cdot \Gamma, y:\overline{A} \vdash_{\text{SP}} \Delta \tag{13}$$

As a final step, by applying the rule (*cut*) *to* (*Eq.* 1) *and* (*Eq.* 13) *we obtain* $P\widehat{\alpha} \dagger \widehat{y}(Q_1\{P\widehat{\alpha}\leftrightarrow x\}) :\cdot \Gamma \vdash_{\text{SP}} \Delta$ *as required.*

$Q = Q_1\widehat{\beta} \dagger \widehat{y}Q_2$, $Q \neq \langle x.\beta\rangle$**:**
$Q\{P\widehat{\alpha}\leftrightarrow x\} = (Q_1\{P\widehat{\alpha}\leftrightarrow x\})\widehat{\beta} \dagger \widehat{y}(Q_2\{P\widehat{\alpha}\leftrightarrow x\})$.
Using (*Eq.* 2) *and applying Lemma 3.9 4, there exists a type* $\overline{B}$ *such that* $Q_1 :\cdot \Gamma, x:\alpha \vdash_{\text{SP}} \beta:\overline{B}, \Delta$ *and* $Q_2 :\cdot \Gamma, x:\overline{A}, y:\overline{B} \vdash_{\text{SP}} \Delta$. *By induction,* $Q\{P\widehat{\alpha}\leftrightarrow x\} :\cdot \Gamma \vdash_{\text{SP}} \beta:\overline{B}, \Delta$ *and* $Q_2\{P\widehat{\alpha}\leftrightarrow x\} :\cdot \Gamma \vdash_{\text{SP}} y:\overline{B}, \Delta$. *We conclude by applying the rule* (*cut*).

(b) *By induction on the structure of the term P. The argument is similar to the previous part, and we show only the most-interesting case, where* $P = \widehat{y}P_1\widehat{\beta}\cdot\alpha$. *Then* $P\{\alpha\leftrightarrow\widehat{x}Q\} = (\widehat{y}(P_1\{\alpha\leftrightarrow\widehat{x}Q\})\widehat{\beta}\cdot\gamma)\widehat{\gamma} \dagger \widehat{x}Q$, *in which* γ *is fresh. By* (*Eq.* 1) *and Lemma 3.9 2, there exist B,C with* $P_1 :\cdot \Gamma, y:B \vdash_{\text{SP}} \beta:C, \Delta$ *and* $B{\rightarrow}C \lhd_{\langle\Gamma;\Delta\rangle} \overline{A}$. *By Proposition 3.8 2, we obtain* $P_1 :\cdot \Gamma, x:\overline{A}, y:B \vdash_{\text{SP}} \beta:C, \Delta$ *and* $Q :\cdot \Gamma, x:\overline{A}, y:B \vdash_{\text{SP}} \beta:C, \Delta$. *By induction,* $P_1\{\alpha\leftrightarrow\widehat{x}Q\} :\cdot \Gamma, y:B \vdash_{\text{SP}} \beta:C, \Delta$. *By the rule* ($\rightarrow\mathcal{R}$) *we obtain* $\widehat{y}(P_1\{\alpha\leftrightarrow\widehat{x}Q\})\widehat{\beta}\cdot\gamma :\cdot \Gamma \vdash_{\text{SP}} \gamma:\overline{A}, \Delta$. *Finally, by the rule* (*cut*) *we obtain that* $(\widehat{y}(P_1\{\alpha\leftrightarrow\widehat{x}Q\})\widehat{\beta}\cdot\gamma)\widehat{\gamma} \dagger \widehat{x}Q :\cdot \Gamma \vdash_{\text{SP}} \Delta$ *as required.*

2. *By inductions on the number of reduction steps, and the structure of the term P, we need only consider the case where P is the redex itself, and is reduced in one step to Q. Therefore, we show the witness reduction result for each of the reduction rules in turn:*

(*cap*) : $\langle x.\alpha\rangle\widehat{\alpha} \dagger \widehat{y}\langle y.\beta\rangle \rightarrow \langle x.\beta\rangle$
Suppose $\langle x.\alpha\rangle\widehat{\alpha} \dagger \widehat{y}\langle y.\beta\rangle :\cdot \Gamma \vdash_{\text{SP}} \Delta$. *By Lemma 3.9 4,* $\alpha \notin \Delta$ *and* $x \notin \Gamma$ *and there exists* $\overline{B}$ *such that* $\langle x.\alpha\rangle :\cdot \Gamma \vdash_{\text{SP}} \alpha:\overline{B}, \Delta$ *and* $\langle y.\beta\rangle :\cdot \Gamma, y:\overline{B} \vdash_{\text{SP}} \Delta$. *By applying Lemma 3.9 1 twice, there exists* $\overline{A}$,$\overline{C}$,Γ',Δ' *such that* $\Gamma = \Gamma', x:\overline{A}$ *and* $\Delta = \beta:\overline{C}, \Delta'$ *with* $\overline{A}{\geq}\overline{B}$ *and* $\overline{B}{\geq}\overline{C}$. *By Proposition 3.6 1,* $\overline{A}{\geq}\overline{C}$. *Therefore, by the rule* (*ax*), *we obtain* $\langle x.\beta\rangle :\cdot \Gamma', x:\overline{A} \vdash_{\text{SP}} \beta:\overline{C}, \Delta'$ *as required.*

(*impR*) : $(\widehat{x}P\widehat{\alpha}\cdot\beta)\widehat{\beta} \dagger \widehat{y}\langle y.\gamma\rangle \rightarrow \widehat{x}P\widehat{\alpha}\cdot\gamma$ (*if* $\beta \notin fp(P)$)
Suppose $(\widehat{x}P\widehat{\alpha}\cdot\beta)\widehat{\beta} \dagger \widehat{y}\langle y.\gamma\rangle :\cdot \Gamma \vdash_{\text{SP}} \Delta$. *By Lemma 3.9 4,* $\beta \notin \Delta$ *and* $y \notin \Gamma$ *and there exists* $\overline{C}$ *with* $\widehat{x}P\widehat{\alpha}\cdot\beta :\cdot \Gamma \vdash_{\text{SP}} \beta:\overline{C}, \Delta$ *and* $\langle y.\gamma\rangle :\cdot \Gamma, y:\overline{C} \vdash_{\text{SP}} \Delta$. *By Lemma 3.9 2, there exist A,B,*Δ'' *such that* $(A{\rightarrow}B) \lhd_{\langle\Gamma;\Delta\rangle} \overline{C}$ *and* $P :\cdot \Gamma, x:A \vdash_{\text{SP}} \alpha:B, \Delta''$ *and* $(\beta:\overline{C}, \Delta) = (\beta:\overline{C}, \Delta'')$. *Since* $\beta \notin P$, *by*

Proposition 3.8 3, we may assume without loss of generality that we have $\beta \notin \Delta''$, and therefore that $\Delta'' = \Delta$. By Lemma 3.9 1, there exist $\overline{D}$,Δ' such that $\Delta = \gamma : \overline{D}, \Delta'$ and $\overline{C} \geq \overline{D}$. By Proposition 3.6 6, there exists a substitution S such that $(S\ \langle\Gamma;\Delta\rangle) = \langle\Gamma;\Delta\rangle$ and $(S\ A{\to}B) \lhd_{\langle\Gamma;\Delta\rangle} \overline{D}$. By Proposition 3.8 1, we have $P :\cdot\ \Gamma, x : (S\ A) \vdash_{\text{SP}} \alpha : (S\ B), \gamma : \overline{D}, \Delta'$. By the rule $(\to\mathcal{R})$, we deduce that $\widehat{xP\alpha}\cdot\gamma :\cdot\ \Gamma \vdash_{\text{SP}} \gamma : \overline{D}, \Delta$ as required.

$(impL)$: $\langle x.\alpha\rangle\widehat{\alpha} \dagger \widehat{y}(P\widehat{\beta}[y]\widehat{z}Q) \to P\widehat{\beta}[x]\widehat{z}Q$ *(if $y \notin fs(P, Q)$)*

Suppose $\langle x.\alpha\rangle\widehat{\alpha} \dagger \widehat{y}(P\widehat{\beta}[y]\widehat{z}Q) :\cdot\ \Gamma \vdash_{\text{SP}} \Delta$. By Lemma 3.9 4, $\alpha \notin \Delta$ and $y \notin \Gamma$ and there exists $\overline{B}$ such that we have both $\langle x.\alpha\rangle :\cdot\ \Gamma \vdash_{\text{SP}} \alpha : \overline{B}, \Delta$ and $P\widehat{\beta}[y]\widehat{z}Q :\cdot\ \Gamma, y : \overline{B} \vdash_{\text{SP}} \Delta$. By Lemma 3.9 1, there exist $\overline{A}$,Γ' such that $\Gamma = \Gamma', x : \overline{A}$ and $\overline{A} \geq \overline{B}$. By Lemma 3.9 3, there exist C,D,Γ'' such that $(\Gamma, y : \overline{B}) = (\Gamma'', y : \overline{B})$ and $\overline{B} \geq (C{\to}D)$ and $P :\cdot\ \Gamma'' \vdash_{\text{SP}} \beta : C, \Delta$ and also $Q :\cdot\ \Gamma'', z : D \vdash_{\text{SP}} \Delta$. By Proposition 3.8 3 we can assume without loss of generality that $\Gamma'' = \Gamma$. Since $\overline{A} \geq \overline{B} \geq (C{\to}D)$, by Proposition 3.6 1 we can deduce $\overline{A} \geq (C{\to}D)$. By applying the rule $(\to\mathcal{L})$ we can finally obtain $P\widehat{\beta}[x]\widehat{z}Q :\cdot\ \Gamma', x : \overline{A} \vdash_{\text{SP}} \Delta$ as required.

(imp) : $(\widehat{xP\alpha}\cdot\beta)\widehat{\beta} \dagger \widehat{y}(Q\widehat{\gamma}[y]\widehat{z}R) \to \left\{\begin{array}{l}(Q\widehat{\gamma} \dagger \widehat{xP})\widehat{\alpha} \dagger \widehat{z}R \\ Q\widehat{\gamma} \dagger \widehat{x}(P\widehat{\alpha} \dagger \widehat{z}R)\end{array}\right\}$ *(if $\beta \notin fp(P)$, $y \notin fs(Q, R)$)*

Suppose $(\widehat{xP\alpha}\cdot\beta)\widehat{\beta} \dagger \widehat{y}(Q\widehat{\gamma}[y]\widehat{z}R) :\cdot\ \Gamma \vdash_{\text{SP}} \Delta$. By Lemma 3.9 4, $\beta \notin \Delta$ and $y \notin \Gamma$ and there exists $\overline{C}$ such that we have both $\widehat{xP\alpha}\cdot\beta :\cdot\ \Gamma \vdash_{\text{SP}} \beta : \overline{C}, \Delta$ and $Q\widehat{\gamma}[y]\widehat{z}R :\cdot\ \Gamma, y : \overline{C} \vdash_{\text{SP}} \Delta$. By Lemma 3.9 2, there exist A,B,Δ' such that $A{\to}B \lhd_{\langle\Gamma;\Delta\rangle} \overline{C}$ and $P :\cdot\ \Gamma, x : A \vdash_{\text{SP}} \alpha : B, \Delta'$ and $(\beta : \overline{C}, \Delta) = (\beta : \overline{C}, \Delta')$. As in previous cases, w.l.o.g. $\Delta' = \Delta$ since $\beta \notin \Gamma$ and $\beta \notin fs(P)$. Now, by Lemma 3.9 3 and similar argument, there exist D,E such that $\overline{C} \geq (D{\to}E)$ and $Q :\cdot\ \Gamma \vdash_{\text{SP}} \gamma : D, \Delta$ and $R :\cdot\ \Gamma, z : E \vdash_{\text{SP}} \Delta$. By Proposition 3.6 6, there exists a substitution S such that $(S\ \langle\Gamma;\Delta\rangle) = \langle\Gamma;\Delta\rangle$ and $(S\ A{\to}B) \lhd_{\langle\Gamma;\Delta\rangle} (D{\to}E)$. In particular, $(S\ A) = D$ and $(S\ B) = E$. By Proposition 3.8 1, we obtain $P :\cdot\ \Gamma, x : D \vdash_{\text{SP}} \alpha : E, \Delta$. Now, by applying Proposition 3.8 2 and the rule (cut) repeatedly, we first obtain both $Q\widehat{\gamma} \dagger \widehat{xP} :\cdot\ \Gamma \vdash_{\text{SP}} \alpha : E, \Delta$ and $P\widehat{\alpha} \dagger \widehat{z}R :\cdot\ \Gamma, x : D \vdash_{\text{SP}} \Delta$, and then obtain both $(Q\widehat{\gamma} \dagger \widehat{xP})\widehat{\alpha} \dagger \widehat{z}R :\cdot\ \Gamma \vdash_{\text{SP}} \Delta$ and $Q\widehat{\gamma} \dagger \widehat{x}(P\widehat{\alpha} \dagger \widehat{z}R) :\cdot\ \Gamma \vdash_{\text{SP}} \Delta$ as required.

$(prop\text{-}\textsc{r})$: $P\widehat{\alpha} \dagger \widehat{x}Q \to Q\{P\widehat{\alpha} \leftrightarrow x\}$ *(if Q does not introduce x)*

By Lemma 3.9 4 and part 1a.

$(prop\text{-}\textsc{l})$: $P\widehat{\alpha} \dagger \widehat{x}Q \to P\{\alpha \leftrightarrow \widehat{x}Q\}$ *(if P does not introduce α)*

By Lemma 3.9 4 and part 1b.

Proof B.4 (of Proposition 3.20)**.**

1. *Immediate from the definition, since* $\overline{B}[\overrightarrow{\varphi_i/X_i}] = \overline{A}$.
2. *Let* $\overline{C} = \overrightarrow{\forall Y_i}.\overline{A}[\overrightarrow{Y_i/\varphi_i}]$. *Let* $\{\overrightarrow{\varphi_j}\}$ *be the subset of* $\{\overrightarrow{\varphi_i}\}$ *which actually occur in* $\overline{A}$. *Without loss of generality, replace all of the other atomic types in* $\{\overrightarrow{\varphi_i}\}$ *with fresh atomic types. By the definition of closure, we have* $\overrightarrow{\varphi_i \notin \langle\Gamma;\Delta\rangle}$. *Now let* $\{\overrightarrow{\varphi_k}\}$ *be the set of atomic types occurring in* $\overline{A}$ *but not in* $\langle\Gamma;\Delta\rangle$. *Then* $\{\overrightarrow{\varphi_j}\} \subseteq \{\overrightarrow{\varphi_k}\}$, *and for some* $\{\overrightarrow{X_k}\}$, $\overline{B} = \overrightarrow{\forall X_k}.\overline{A}[\overrightarrow{X_k/\varphi_k}]$. *Then* $\overline{C} = \overrightarrow{\forall Y_i}.\overline{B}[\overrightarrow{\varphi_k/X_k}]\,[\overrightarrow{Y_i/\varphi_i}]$ *as required.*
3. *Write* $\overline{B} = \overrightarrow{\forall X_i}.\overline{A}[\overrightarrow{X_i/\varphi_i}]$, *where* $\{\varphi\}$ *are the atomic types occurring in* $\overline{A}$ *but not in* $\langle\Gamma;\Delta\rangle$. *Now let* $\overline{C} = \forall$*-closure* $(S\ \overline{A})\ \langle(S\ \Gamma);(S\ \Delta)\rangle = \overrightarrow{\forall Y_j}.(S\ \overline{A})[\overrightarrow{Y_j/\varphi_j}]$, *where* $\{\varphi_j\}$ *are the atomic types occurring in* $\overline{A}$ *but not in* $\langle(S\ \Gamma);(S\ \Delta)\rangle$. *Then we aim to show* $(S\ \overline{B}) \geq \overline{C}$. *This follows because* $(S\ \overline{A}) = (S\ \overline{B})[\overrightarrow{((S\ \varphi_i)/X_i}]$ *and so we have* $\overline{C} = \overrightarrow{\forall Y_j}.(S\ \overline{B})[\overrightarrow{((S\ \varphi_i)/X_i}]\,[\overrightarrow{Y_j/\varphi_j}]$. *Finally, we must be sure that* $\overrightarrow{\varphi_j} \notin (S\ \overline{B})$. *Suppose that there is some* $\varphi_j \in (S\ \overline{B})$ *(and we will show a contradiction). Then, by Lemma 3.22 3, there are two possible cases:*

   $\varphi_j \in \overline{B}$ **and** $(S\ \varphi_j) = \varphi_j$**:** *Then, since* $\overline{B} = \forall$*-closure* $\overline{A}\ \langle\Gamma;\Delta\rangle$, *we must have* $\varphi_j \in \langle\Gamma;\Delta\rangle$. *However, then* $\varphi_j \in \langle(S\ \Gamma);(S\ \Delta)\rangle$, *contradicting the definition of* $\overline{C}$.

   $\exists\varphi \in \overline{B}$ **with** $\varphi_j \in (S\ \varphi)$**:** *Then, since* $\overline{B} = \forall$*-closure* $\overline{A}\ \langle\Gamma;\Delta\rangle$, *we must have* $\varphi \in \langle\Gamma;\Delta\rangle$. *But then* $\varphi_j \in \langle(S\ \Gamma);(S\ \Delta)\rangle$, *contradicting the definition of* $\overline{C}$.
4. *Follows easily from the observation that for any atomic type* φ *and types* $\overline{A} \geq \overline{B}$, $\varphi \in \overline{A} \Rightarrow \varphi \in \overline{B}$.

Proof B.5 (of Theorem 3.25)**.** 1. *Let* $\overline{A} = \overrightarrow{\forall Y_j}.A$ *and* $\overline{B} = \overrightarrow{\forall Z_k}.B$. *In accordance with the definition of the algorithm, let* $A' = freshInst(\overline{A}) = A[\overrightarrow{\varphi_j/Y_j}]$ *and* $B' = freshInst(\overline{B}) = B[\overrightarrow{\varphi_k/Z_k}]$. *Since the call succeeds, we must have that the call unify* $A'\ B'$ *succeeds, yielding a substitution*

$$S_u = unify\ A'\ B' \tag{14}$$

Let $C_u = (S_u\ A')$. *Note that by soundness of unification (Lemma 3.18) we have* $(S_u\ B') = (S_u\ A') = C_u$.
Define a set of atomic types $\{\overrightarrow{\varphi_i}\} = atoms(C_u)\backslash(atoms(S_u\ \overline{A}) \cup atoms(S_u\ \overline{B}))$.
We have that $\overline{C} = \overrightarrow{\forall Z_i}.C_u[\overrightarrow{Z_i/\varphi_i}]$ *while* $S = (S_u \cap (atoms(\overline{A}) \cup atoms(\overline{B})))$.
We will now show that $(S\ \overline{A}) \geq \overline{C}$. *The argument that also* $(S\ \overline{B}) \geq \overline{C}$ *is analogous and will be omitted.*

Notice firstly that (using Lemma 3.23 1), we have $(S_r\,\overline{A}) = (S_u\,\overline{A}) = \overrightarrow{\forall Y_j}.(S_u\,A)$. *Define a set of types* $\overrightarrow{D_j = (S_u\,\varphi_j)}$. *Then by construction, we have that:*

$$C_u = (S_u\,A[\overrightarrow{\varphi_j/Y_j}]) = (S_u\,A)[\overrightarrow{(S_u\,\varphi_j)/Y_j}] = (S_u\,A)[\overrightarrow{D_j/Y_j}]$$

Therefore, $C_u[\overrightarrow{Z_i/\varphi_i}] = (S_u\,A)[\overrightarrow{D_j/Y_j}]\,[\overrightarrow{Z_i/\varphi_i}]$. *Furthermore, by the definition of the set* $\{\overrightarrow{\varphi_i}\}$, *we have* $\varphi_i \notin (S_u\,\overline{A})$. *Therefore, by Definition 2.5, we have* $\overrightarrow{\forall Y_j}.(S_u\,A) \geq \overline{C}$. *Since we know* $(S_r\,\overline{A}) = \overrightarrow{\forall Y_j}.(S_u\,A)$, *we have* $(S_r\,\overline{A}) \geq \overline{C}$ *as required.*

2. *Firstly, let us define (in which all of the* $\overrightarrow{\varphi_j}, \overrightarrow{\varphi_k}, \overrightarrow{\varphi_l}$ *are fresh):*

$$\overline{A} = \overrightarrow{\forall Y_j}.A \tag{15}$$
$$\overline{B} = \overrightarrow{\forall Z_k}.B \tag{16}$$
$$\overline{D} = \overrightarrow{\forall W_l}.D \tag{17}$$
$$A' = freshInst(\overline{A}) = A[\overrightarrow{\varphi_j/Y_j}] \tag{18}$$
$$B' = freshInst(\overline{B}) = B[\overrightarrow{\varphi_k/Z_k}] \tag{19}$$
$$D' = freshInst(\overline{D}) = D[\overrightarrow{\varphi_l/W_l}] \tag{20}$$

Since $(S\ \overline{A}) \geq \overline{D}$, *we know (from Definition 2.5) that, for some* $\overrightarrow{E'_j}$ *and some* $\overrightarrow{\varphi'_l} \notin atoms(S\ \overline{A})$, *we have* $D = (S\ A)[\overrightarrow{E'_j/Y_j}]\,[\overrightarrow{W_l/\varphi'_l}]$. *Define the substitution* $S_A = \{\overrightarrow{(\varphi'_l \mapsto \varphi_l)}\}$, *and define the types* $\overrightarrow{E_j = (S_A\,E'_j)}$. *Then we obtain that* $D = (S\ A)[\overrightarrow{E_j/Y_j}]\,[\overrightarrow{W_l/\varphi_l}]$. *Notice that*

$$\overrightarrow{\varphi_l \notin atoms(S\ \overline{A})} \tag{21}$$

since the $\overrightarrow{\varphi_l}$ *were chosen to be fresh.*
In a similar fashion, from the fact that $(S\ \overline{B}) \geq \overline{D}$ *we can deduce that, for some types* $\overrightarrow{F_k}$ *we have* $D = (S\ B)[\overrightarrow{F_k/Z_k}]\,[\overrightarrow{W_l/\varphi_l}]$, *and that*

$$\overrightarrow{\varphi_l \notin atoms(S\ \overline{B})} \tag{22}$$

Since $D' = D[\overrightarrow{\varphi_l/W_l}]$, *we deduce from the above that* $(S\ A)[\overrightarrow{E_j/Y_j}] = D' = (S\ B)[\overrightarrow{F_k/Z_k}]$. *Define next the two substitutions*

$$S_E = \{\overrightarrow{(\varphi_j \mapsto E_j)}\} \tag{23}$$
$$S_F = \{\overrightarrow{(\varphi_k \mapsto E_k)}\} \tag{24}$$

By construction, we have $(S_F \circ S_E \circ S\ A') = (S\ A)[\overrightarrow{E_j/Y_j}] = D' = (S_F \circ S_E \circ S\ B')$. *Therefore, the substitution* $(S_F \circ S_E \circ S)$ *is a unifier for the types* A' *and* B'. *By*

completeness of unification (Lemma 3.18), there exist substitutions S_1 and S_u such that

$$(S_F \circ S_E \circ S) = (S_1 \circ S_u) \tag{25}$$

$$S_u = \mathit{unify}\ A'\ B' \tag{26}$$

In particular, the call unify A' B' does not fail, and so neither does the call unifyGen $\overline{A}$ $\overline{B}$ in question. Therefore, there exist $(S_r, \overline{C})$ = unifyGen $\overline{A}$ $\overline{B}$, where:

$$S_r = (S_u \cap (\mathit{atoms}(\overline{A}) \cup \mathit{atoms}(\overline{B}))) \tag{27}$$

$$\{\overrightarrow{\varphi_i}\} = \mathit{atoms}(S_u\ A') \backslash (\mathit{atoms}(S_u\ \overline{A}) \cup \mathit{atoms}(S_u\ \overline{B})) \tag{28}$$

$$\overline{C} = \overrightarrow{\forall X_i}.(S_u\ A')[\overrightarrow{X_i/\varphi_i}] \tag{29}$$

For convenience, we define $C' = (S_u\ A')$, so that $\overline{C} = \overrightarrow{\forall X_i}.C'[\overrightarrow{X_i/\varphi_i}]$.
We seek next to show that $(S_1\ \overline{C}) \geq \overline{D}$, from which we will be able to obtain the desired result without too much trouble. We would like to begin by showing that $(S_1\ \overline{C}) = \overrightarrow{\forall X_i}.(S_1 \circ S_u\ A')[\overrightarrow{X_i/\varphi_i}]$. However, this is not necessarily true, since we have no guarantee that the φ_is are not affected by the substitution S_1. We choose to work around this, by choosing a new set $\overrightarrow{\varphi'_i}$ of fresh atomic types (one for each atomic type φ_i) and employing a renaming substitution

$$S_2 = \{\overrightarrow{(\varphi_i \mapsto \varphi'_i)}\} \tag{30}$$

We can now see instead that

$$(S_1\ \overline{C}) = \overrightarrow{\forall X_i}.(S_1 \circ S_2 \circ S_u\ A')[\overrightarrow{X_i/\varphi'_i}] \tag{31}$$

To be able to deduce that $(S_1\ \overline{C}) \geq \overline{D}$, then (by Definition 2.5), we require a set of types $\overrightarrow{G_i}$ such that $D = (((S_1 \circ S_2 \circ S_u\ A')[\overrightarrow{X_i/\varphi'_i}])[\overrightarrow{G_i/X_i}][\overrightarrow{W_l/\varphi_l}]$, and to show also that $\overrightarrow{\varphi_l \notin (S_1\ \overline{C})}$.
We claim that if we define the types $\overrightarrow{G_i = (S_1\ \varphi_i)}$ then these will do the trick. Firstly, if we define the substitution $S_G = \{\overrightarrow{(\varphi'_i \mapsto S_1\ \varphi_i)}\}$ then we can show that $(((S_1 \circ S_2 \circ S_u\ A')[\overrightarrow{X_i/\varphi'_i}])[\overrightarrow{G_i/X_i}][\overrightarrow{W_l/\varphi_l}] = D$ as follows:

$$
\begin{array}{ll}
(((S_1 \circ S_2 \circ S_u)\ A')[\overrightarrow{X_i/\varphi'_i}])[\overrightarrow{(S_1\ \varphi_i)/X_i}]\ [\overrightarrow{W_l/\varphi_l}] & \\
= ((S_G \circ S_1 \circ S_2 \circ S_u)\ A')[\overrightarrow{W_l/\varphi_l}] & \textit{composing } [\overrightarrow{X_i/\varphi'_i}], [\overrightarrow{(S_1\ \varphi_i)/X_i}] \\
= ((S_1 \circ S_G \circ S_2 \circ S_u)\ A')[\overrightarrow{W_l/\varphi_l}] & \textit{Lemma 3.22 1} \\
= ((S_1 \circ (S_1 \cap \{\overrightarrow{\varphi_i}\}) \circ S_u)\ A')[\overrightarrow{W_l/\varphi_l}] & \textit{Lemma 3.23 9} \\
= (((S_1 \backslash \{\overrightarrow{\varphi_i}\}) \circ (S_1 \cap \{\overrightarrow{\varphi_i}\}) \circ S_u)\ A')[\overrightarrow{W_l/\varphi_l}] & \textit{idempotency of } S_1\textit{, Lemma 3.23 4} \\
= ((S_1 \circ S_u)\ A')[\overrightarrow{W_l/\varphi_l}] & \textit{Lemma 3.23 10} \\
= ((S_F \circ S_E \circ S\ A')[\overrightarrow{W_l/\varphi_l}] & (S_F \circ S_E \circ S\ = S_1 \circ S_u) \\
= D'[\overrightarrow{W_l/\varphi_l}] & (D' = S_F \circ S_E \circ S\ A') \\
= D &
\end{array}
$$

We need to also show that $\overrightarrow{\varphi_l \notin (S_1\ \overline{C})}$ *, which, combined with the argument above would justify that* $(S_1\ \overline{C}) \geq \overline{D}$*. We will argue by contradiction; assuming that for some* $\varphi_l \in \{\overrightarrow{\varphi_l}\}$ *we have*

$$\varphi_l \in (S_1\ \overline{C}) \tag{32}$$

we will show that a contradiction inevitably follows. By (*Eq.* 32) *and* (*Eq.* 31)*, we deduce that*

$$\varphi_l \in \overrightarrow{\forall X_i}.(S_1 \circ S_2 \circ S_u\ A')[\overrightarrow{X_i/\varphi'_i}] \tag{33}$$

Then, by Lemma 3.22 4, we must have 3.22 3, we identify two cases:

Case 1: $\varphi_l \notin$ ***dom***(S_1) **and** $\varphi_l \in (S_2 \circ S_u\ A')$ *Then since* (*Eq.* 30) $\varphi_l \notin \{\overrightarrow{\varphi'_i}\}$*, by Lemma 3.22 3 again, we must have*

$$\varphi_l \in atoms(S_u\ A') \tag{34}$$

But from the freshness of φ_l *when chosen* (*Eq.* 20) *we must have:* $\varphi_l \notin atoms(A')$ *and* $\varphi_l \notin atoms(B')$*. Furthermore, by* (*Eq.* 26)*, we can assume also that* $\varphi_l \notin atoms(S_u\ A')$*, contradicting* (*Eq.* 34)

Case 2: $\exists \varphi, H$ **with** $\varphi \in$ ***atoms***$(S_2 \circ S_u\ A')$ **and** $(\varphi \mapsto H) \in S_1$ **and** $\varphi_l \in$ ***atoms***(H) *We must have* $\varphi \notin \{\overrightarrow{\varphi'_i}\}$ *since this set of atomic types was chosen to be fresh at* (*Eq.* 30)*. Therefore, by Lemma 3.22 3, we must have* $\varphi \in (S_u\ A')$ *and* $\varphi \notin \{\overrightarrow{\varphi_i}\}$*. By* (*Eq.* 28) *it must be the case that either* $\varphi \in (S_u\ \overline{A})$ *or* $\varphi \in (S_u\ \overline{B})$*. But then, by the assumptions of this case, either* $\varphi_l \in (S_1 \circ S_u\ \overline{A})$ *or* $\varphi_l \in (S_1 \circ S_u\ \overline{B})$*. By* (*Eq.* 25)*,* (*Eq.* 23) *and* (*Eq.* 24)*, we have that either* $\varphi_l \in (S\ A)$ *or* $\varphi_l \in (S\ B)$*, contradicting* (*Eq.* 21) *and* (*Eq.* 22)*, respectively.*

This completes the argument justifying that

$$(S_1\ \overline{C}) \geq \overline{D} \tag{35}$$

We now need to work on the form of the substitutions involved. We will employ the set of atomic types $\Psi = \{\overrightarrow{\varphi_j}\} \cup \{\overrightarrow{\varphi_k}\}$*, noting that, by* (*Eq.* 23) *and* (*Eq.* 24)*, we know*

$$\Psi = dom(S_F \circ S_E) \tag{36}$$

We can then show:

$$\begin{array}{rrcll}
 & S_F \circ S_E \circ S & = & S_1 \circ S_u & (Eq.\ 25) \\
\therefore & (S_F \circ S_E \circ S) \backslash \Psi & = & (S_1 \circ S_u) \backslash \Psi & \\
\therefore & ((S_F \circ S_E) \backslash \Psi) \circ S & = & (S_1 \circ S_u) \backslash \Psi & \textit{Lemma 3.23 7} \\
\therefore & id \circ S & = & (S_1 \circ S_u) \backslash \Psi & \textit{Lemma 3.23 5} \\
\therefore & S & = & (S_1 \backslash \Psi) \circ (S_u \backslash \Psi) & \textit{Lemma 3.23 6}
\end{array}$$

This is close to what we need, but the substitution actually returned from the call is $S_r = (S_u \cap (atoms(\overline{A}) \cup atoms(\overline{B})))$ *as defined in* (*Eq.* 27)*.*
We now write $\Phi = (atoms(\overline{A}) \cup atoms(\overline{B}))$ *and deduce:*

$$\begin{array}{rcll}
\therefore S & = & (S_1 \backslash \Psi) \circ ((S_u \backslash \Psi) \backslash \Phi) \circ ((S_u \backslash \Psi) \cap \Phi) & \textit{Lemma 3.23 10} \\
\therefore S & = & (S_1 \backslash \Psi) \circ (S_u \backslash (\Psi \cup \Phi)) \circ ((S_u \backslash \Psi) \cap \Phi) & \textit{Lemma 3.23 2} \\
\therefore S & = & (S_1 \backslash \Psi) \circ (S_u \backslash (\Psi \cup \Phi)) \circ (S_u \cap \Phi) & \textit{Lemma 3.23 3}
\end{array}$$

Now define $S_3 = (S_1 \backslash \Psi) \circ (S_u \backslash (\Psi \cup \Phi))$*. We observe that:*

$$\begin{array}{rcll}
(S_3\ \overline{C}) & = & ((S_1 \backslash \Psi) \circ (S_u \backslash (\Psi \cup \Phi))\ \overline{C}) & \\
 & = & ((S_1 \backslash \Psi)\ \overline{C}) & \textit{Lemma 3.23 8} \\
 & = & (S_1\ \overline{C}) & \textit{Lemma 3.23 4} \\
 & \geq & \overline{D} & (Eq.\ 35)
\end{array}$$

Therefore, we have that $S = S_3 \circ S_r$ *and* $(S_3\ \overline{C}) \geq \overline{D}$*, as required.*

Proof B.6 (of Theorem 3.29)**.** 1. *By induction on the structure of the term R.*

$R = \langle x.\alpha \rangle$**:** *Let* $\overline{A}$ = *typeof* $x\ \Gamma$*. From the definition of the algorithm, we have* $S_R = id$ *and* $\Delta_R = \{\alpha : \overline{A}\}$*. By* (*Eq.* 3)*, we have* $\Gamma = \Gamma, x : \overline{A}$*. Then, by the rule* (*ax*)*, we have* $\langle x.\alpha \rangle :\cdot\ \Gamma, x : \overline{A} \vdash_{\mathrm{SP}} \alpha : \overline{A}$ *as required.*

$R = \widehat{xP\alpha}\cdot\beta$**:** *In accordance with the algorithm, let:*

$$\varphi = \mathit{fresh} \tag{37}$$
$$\langle S_P, \Delta_P\rangle = \mathrm{sppc}\,(P, \Gamma\cup\{x:\varphi\}) \tag{38}$$
$$A = (S_P\ \varphi) \tag{39}$$
$$B = \mathit{freshInstance}\ \mathit{typeof}\ \alpha\ \Delta_P \tag{40}$$
$$\overline{C} = \forall\text{-}\mathit{closure}\ A{\to}B\ \langle(S_P\ \Gamma);\Delta_P\backslash\alpha\rangle \tag{41}$$
$$\langle S_u, \overline{D}\rangle = \begin{cases} \mathit{unifyGen}\ \overline{C}\ \mathit{typeof}\ \beta\ \Delta_P & \mathit{if}\ \beta\in\Delta_P \\ \langle \mathit{id}, \overline{C}\rangle & \mathit{otherwise} \end{cases} \tag{42}$$
$$S_r = (S_u \circ S_P \cap \mathit{atoms}(\Gamma)) \tag{43}$$
$$\mathrm{sppc}\,(\widehat{xP\alpha}\cdot\beta, \Gamma) = \langle S_r, (S_u\ \Delta_P\backslash\alpha\backslash\beta)\cup\{\beta:\overline{D}\}\rangle \tag{44}$$

By induction, using (*Eq.* 38)*, we have*

$$P :\cdot\ (S_P\ \Gamma\cup\{x:\varphi\}) \vdash_{\mathrm{SP}} \Delta_P \tag{45}$$

By applying Propositions 3.8 2 and 5 to (*Eq.* 45) *as appropriate, and using* (*Eq.* 39) *and* (*Eq.* 40)*, we obtain*

$$P :\cdot\ (S_P\ \Gamma), x:A \vdash_{\mathrm{SP}} (\Delta_P\backslash\alpha), \alpha:B \tag{46}$$

We wish now to apply the type-assignment rule (*impR*)*. However, examining the conclusion of this rule, we need to ensure that the resulting right-context will be well-formed, i.e., deal with the possibility that* $\beta\in\Delta_P$ *already. To do this we consider two cases:*

$\beta\in\Delta_P$**:** *Then by* (*Eq.* 42) *we have*

$$\langle S_u, \overline{D}\rangle = \mathit{unifyGen}\ \overline{C}\ \mathit{typeof}\ \beta\ \Delta_P \tag{47}$$

Since the original call to $\mathrm{sppc}\,(R,\Gamma)$ *was assumed to succeed, this sub-call to unifyGen must also succeed, so such a pair exists. By the soundness of unifyGen (Theorem 3.25 1), we have that*

$$(S_u\ \overline{C}) \geq \overline{D} \tag{48}$$
$$(S_u\ \mathit{typeof}\ \beta\ \Delta_P) \geq \overline{D} \tag{49}$$

. By Proposition 3.20 1 and (*Eq.* 41)*, we have*

$$(A{\to}B) \lhd_{\langle(S_P\ \Gamma);\Delta_P\backslash\alpha\rangle} \overline{C} \tag{50}$$

By Proposition 3.6 5 there exists a substitution S' such that:

$$(S'\ \langle (S_P\ \Gamma); \Delta_P \backslash \alpha\rangle) = \langle (S_P\ \Gamma); \Delta_P \backslash \alpha\rangle \tag{51}$$

$$(S'\ \overline{C}) = \overline{C} \tag{52}$$

$$(S_u \circ S'\ A \rightarrow B) \lhd_{(S_u\ \langle (S_P\ \Gamma); \Delta_P \backslash \alpha\rangle)} (S_u\ \overline{C}) \tag{53}$$

By Proposition 3.6 6 and (*Eq.* 48)*, we obtain that there exists a substitution S'' such that:*

$$(S''\ (S_u\ \langle (S_P\ \Gamma); \Delta_P \backslash \alpha\rangle) = (S_u\ \langle (S_P\ \Gamma); \Delta_P \backslash \alpha\rangle)) \tag{54}$$

$$(S''\ (S_u \circ S'\ A \rightarrow B)) \lhd_{(S_u\ \langle (S_P\ \Gamma); \Delta_P \backslash \alpha\rangle)} \overline{D} \tag{55}$$

Now, taking (*Eq.* 46) *and applying Proposition 3.8 1 using the substitution $S''' = S'' \circ S_u \circ S'$, along with* (*Eq.* 51) *and* (*Eq.* 54)*, we obtain:*

$$P :\cdot (S_u \circ S_P\ \Gamma), x : (S'''\ A) \vdash_{\mathrm{SP}} (S_u\ (\Delta_P \backslash \alpha \backslash \beta)), \alpha : (S'''\ B) \tag{56}$$

By Proposition 3.8 4 and (*Eq.* 49)*, we obtain*

$$P :\cdot (S_u \circ S_P\ \Gamma), x : (S'''\ A) \vdash_{\mathrm{SP}} (S_u\ (\Delta_P \backslash \alpha \backslash \beta)), \beta : \overline{D}, \alpha : (S'''\ B) \tag{57}$$

Combining this with (*Eq.* 55)*, we can apply the rule* ($\rightarrow\mathcal{R}$)*, to obtain $\widehat{x}P\widehat{\alpha}\cdot\beta :\cdot (S_u \circ S_P\ \Gamma) \vdash_{\mathrm{SP}} (S_u\ \Delta_P \backslash \alpha \backslash \beta), \beta : \overline{D}$ as required.*

$\beta \notin \Delta_P$**:** *Then, by* (*Eq.* 42)*, we have $S_u = id$ and $\overline{D} = \overline{C}$. Furthermore, since $\beta \notin \Delta_P$, by applying Proposition 3.20 to* (*Eq.* 41)*, and applying the rule* (*impR*) *to* (*Eq.* 46)*, we obtain*

$$\widehat{x}P\widehat{\alpha}\cdot\beta :\cdot (S_P\ \Gamma) \vdash_{\mathrm{SP}} (\Delta_P \backslash \alpha), \beta : \overline{D} \tag{58}$$

Therefore, trivially we have $\widehat{x}P\widehat{\alpha}\cdot\beta :\cdot (S_u \circ S_p\ \Gamma) \vdash_{\mathrm{SP}} (S_u\ \Delta_P \backslash \alpha \backslash \beta), \beta : \overline{D}$.

We conclude the case, noting that $(S_r\ \Gamma) = (S_u \circ S_p\ \Gamma)$ by definition of S_r.

$R = P\widehat{\alpha}\,[x]\,\widehat{y}Q$**:** *In accordance with the algorithm, let:*

$$\langle S_P, \Delta_P\rangle = \mathrm{sppc}\,(P, \Gamma) \tag{59}$$
$$\varphi = \mathit{fresh} \tag{60}$$
$$\langle S_Q, \Delta_Q\rangle = \mathrm{sppc}\,(Q, (S_P\,\Gamma)\cup\{y:\varphi\}) \tag{61}$$
$$A = \mathit{freshInstance\ typeof}\ \alpha\ (S_Q\,\Delta_P) \tag{62}$$
$$B = (S_Q\,\varphi) \tag{63}$$
$$C = \mathit{freshInstance\ typeof}\ x\ (S_Q{\circ}S_P\,\Gamma) \tag{64}$$
$$S_u = \mathit{unify}\ C\ A{\rightarrow}B \tag{65}$$
$$\langle S_c, \Delta_c\rangle = \mathit{unifyGenContexts}\ (S_u{\circ}S_Q\,\Delta_P{\backslash}\alpha)\ (S_u\,\Delta_Q) \tag{66}$$
$$S_r = (S_c{\circ}S_u{\circ}S_Q{\circ}S_P \cap \mathit{atoms}(\Gamma)) \tag{67}$$
$$\mathrm{sppc}\,(P\widehat{\alpha}\,[y]\,\widehat{x}Q, \Gamma) = \langle S_r, \Delta_c\rangle \tag{68}$$

By induction, twice (using (*Eq.* 59) *and* (*Eq.* 61) *with* (*Eq.* 63)*), we obtain:*

$$P :\cdot (S_P\,\Gamma) \vdash_{\mathrm{SP}} \Delta_P \tag{69}$$
$$Q :\cdot (S_q{\circ}S_p\,\Gamma), y:B \vdash_{\mathrm{SP}} \Delta_Q \tag{70}$$

By Proposition 3.8 1 and (*Eq.* 69)*, we obtain*

$$P :\cdot (S_Q{\circ}S_P\,\Gamma) \vdash_{\mathrm{SP}} (S_Q\,\Delta_P) \tag{71}$$

Using Proposition 3.8 5 with (*Eq.* 62) *(and applying Proposition 3.8 2 if necessary), we obtain*

$$P :\cdot (S_Q{\circ}S_P\,\Gamma) \vdash_{\mathrm{SP}} (S_Q\,\Delta_P{\backslash}\alpha), \alpha:A \tag{72}$$

Now, let $\overline{C}$ = *typeof x typeof x* $(S_Q{\circ}S_P\,\Gamma)$ *(and so* C = *freshInst*$(\overline{C})$*, by* (*Eq.* 64)*). By definition of freshInst, we have* $\overline{C}{\geq}C$*. By Proposition 3.6 4 and using* (*Eq.* 65)*, we have*

$$(S_c{\circ}S_u\,\overline{C}){\geq}(S_c{\circ}S_u\,C) = (S_c{\circ}S_u\,A{\rightarrow}B) = ((S_c{\circ}S_u\,A){\rightarrow}S_c{\circ}S_u\,B) \tag{73}$$

By applying Proposition 3.8 1 twice, to (*Eq.* 72) *and* (*Eq.* 70)*, we obtain:*

$$P :\cdot (S_c{\circ}S_u{\circ}S_Q{\circ}S_P\,\Gamma) \vdash_{\mathrm{SP}} (S_c{\circ}S_u{\circ}S_Q\,\Delta_P{\backslash}\alpha), \alpha:(S_c{\circ}S_u\,A) \tag{74}$$
$$Q :\cdot (S_c{\circ}S_u{\circ}S_Q{\circ}S_P\,\Gamma), y:(S_c{\circ}S_u\,B) \vdash_{\mathrm{SP}} (S_c{\circ}S_u\,\Delta_Q) \tag{75}$$

By the soundness of unifyGenContexts (Proposition 3.26 1), we have that $(S_c \circ S_u \circ S_Q\ \Delta_P \backslash \alpha) \geq \Delta_C$ *and* $(S_c \circ S_u\ \Delta_Q) \geq \Delta_C$*. Therefore, by Proposition 3.8 6, we obtain that both* $P :\cdot (S_c \circ S_u \circ S_Q \circ S_P\ \Gamma) \vdash_{\text{SP}} \Delta_C, \alpha : (S_c \circ S_u\ A)$ *and* $Q :\cdot (S_c \circ S_u \circ S_Q \circ S_P\ \Gamma), y : (S_c \circ S_u\ B) \vdash_{\text{SP}} \Delta_C$*. Using (Eq. 73) and the rule* $(\rightarrow\mathcal{L})$*, we obtain* $P\widehat{\alpha}[x]\widehat{y}Q :\cdot (S_c \circ S_u \circ S_Q \circ S_P\ \Gamma) \vdash_{\text{SP}} \Delta_C$*, and we conclude by Lemma 3.23 1.*

$R = P\widehat{\alpha} \dagger \widehat{x}Q$**:** *By induction, twice, we obtain that both* $P :\cdot (S_P\ \Gamma) \vdash_{\text{SP}} \Delta_P$ *and* $Q :\cdot (S_Q \circ S_P\ \Gamma), x : (S_Q\ \overline{A}) \vdash_{\text{SP}} \Delta_Q$*. By weakening (Proposition 3.8 2) as necessary, we obtain* $P :\cdot (S_P\ \Gamma) \vdash_{\text{SP}} (\Delta_P \backslash \alpha), \alpha : \overline{A}$*. Then, by applying Proposition 3.8 1, twice,* $P :\cdot (S_c \circ S_Q \circ S_P\ \Gamma) \vdash_{\text{SP}} (S_c \circ S_Q\ \Delta_P \backslash \alpha), \alpha : (S_c \circ S_Q\ \overline{A})$ *and* $Q :\cdot (S_c \circ S_Q \circ S_P\ \Gamma), x : (S_c \circ S_Q\ \overline{A}) \vdash_{\text{SP}} (S_c\ \Delta_Q)$*.*

By the soundness of unifyGenContexts (Proposition 3.26 1), and Proposition 3.8 6, we obtain that both $P :\cdot (S_c \circ S_Q \circ S_P\ \Gamma) \vdash_{\text{SP}} \Delta_c, \alpha : (S_c \circ S_u\ \overline{A})$ *and* $Q :\cdot (S_c \circ S_Q \circ S_P\ \Gamma), x : (S_c \circ S_Q\ \overline{A}) \vdash_{\text{SP}} \Delta_c$*. By applying the rule (cut), we obtain* $P\widehat{\alpha} \dagger \widehat{x}Q :\cdot (S_c \circ S_Q \circ S_P\ \Gamma) \vdash_{\text{SP}} \Delta_c$*. We conclude by applying Lemma 3.23 1, since* $S_r = (S_c \circ S_Q \circ S_P \cap atoms(\Gamma))$*.*

2. *By induction on the structure of the term R.*

$R = \langle x.\alpha \rangle$**:** *By Lemma 3.9 1, we must have* $\Gamma = \Gamma', x : \overline{A}$ *and* $\Delta = \alpha : \overline{B}, \Delta'$ *with* $(S\ \overline{A}) \geq \overline{B}$*. Since* $\Delta_R = \{\alpha : (S\ \overline{A})\}$*, and* $S_R = id$*, can choose* $S' = S$ *and then we have we have* $(S'\ \Delta_R) \geq \Delta$ *as required.*

$R = \widehat{x}P\widehat{\alpha} \cdot \beta$**:** *From the definition of the algorithm, we have:*

$$\text{sppc}\,(\widehat{x}P\widehat{\alpha} \cdot \beta, \Gamma) = \langle S_r, (S_u\ \Delta_P \backslash \alpha \backslash \beta) \cup \{\beta : \overline{D}\} \rangle \tag{76}$$

$$\varphi = fresh \tag{77}$$

$$\langle S_P, \Delta_P \rangle = \text{sppc}\,(P, \Gamma \cup \{x : \varphi\}) \tag{78}$$

$$A = (S_P\ \varphi) \tag{79}$$

$$B = freshInstance\ typeof\ \alpha\ \Delta_P \tag{80}$$

$$\overline{C} = \forall\text{-}closure\ A \rightarrow B\ \langle (S_P\ \Gamma); \Delta_P \backslash \alpha \rangle \tag{81}$$

$$\langle S_u, \overline{D} \rangle = \begin{cases} unifyGen\ \overline{C}\ typeof\ \beta\ \Delta_P & if\ \beta \in \Delta_P \\ \langle id, \overline{C} \rangle & otherwise \end{cases} \tag{82}$$

$$S_r = (S_u \circ S_P \cap atoms(\Gamma)) \tag{83}$$

By Lemma 3.9 2, we must have $\Delta = \beta : \overline{G}, \Delta'$ *and there exist E,F such that*

$$P :\cdot (S\ \Gamma), x : E \vdash_{\text{SP}} \alpha : F, \Delta' \tag{84}$$

$$E \rightarrow F \lhd_{\langle (S\ \Gamma); \Delta' \rangle} \overline{G} \tag{85}$$

Define $S_E = \{(\varphi \mapsto E)\}$. *Then, by construction,* $(S_E \circ S\ \Gamma, x:\varphi) = ((S\ \Gamma), x:E)$. *By induction, using* (*Eq.* 78), *there exists* S_1 *such that*

$$S_E \circ S = S_1 \circ S_P \tag{86}$$

$$(S_1\ \Delta_P) \geq (\Delta', \alpha : F) \tag{87}$$

Let $\overline{B}$ = *typeof* $\alpha\ \Delta_P$ *(so that, by* (*Eq.* 80), B = *freshInst*($\overline{B}$)*). By Proposition 3.6 7, there exists* S_2 *such that dom*(S_2) *consists of only the fresh atomic types in* B = *freshInst*($\overline{B}$), *and* $(S_2 \circ S_1\ B) = F$. *Now we have*

$$(S_2 \circ S_1\ A \rightarrow B) = E \rightarrow F \tag{88}$$

$$(S_2 \circ S_1\ \Delta_P \backslash \alpha) = (S_1\ \Delta_P \backslash \alpha) \geq \Delta' \tag{89}$$

By using (*Eq.* 81) *with Proposition 3.20 3, we are able to show that* $(S_2 \circ S_1\ \overline{C}) \geq \forall$*-closure* $(S_2 \circ S_1\ A \rightarrow B)\ (S_2 \circ S_1\ \langle (S_P\ \Gamma); \Delta_P \backslash \alpha \rangle)$, *and, by our knowledge of dom*(S_2) *and using* (*Eq.* 88), *we can simplify this to obtain* $(S_1\ \overline{C}) \geq \forall$*-closure* $E \rightarrow F\ \langle (S_1 \circ S_P\ \Gamma); (S_1\ (\Delta_P \backslash \alpha)) \rangle$. *Now, by* (*Eq.* 87), *we have* $(S_1\ (\Delta_P \backslash \alpha)) \geq \Delta'$. *Using this,* (*Eq.* 86) *and the fact that dom*(S_E) = φ, *we use and using Proposition 3.20 4 to obtain* $(S_1\ \overline{C}) \geq \forall$*-closure* $E \rightarrow F\ \langle (S\ \Gamma); \Delta' \rangle$. *By Proposition 3.20 1, using* (*Eq.* 85), *we obtain* $\forall$*-closure* $E \rightarrow F\ \langle (S\ \Gamma); \Delta' \rangle \geq \overline{G}$, *and so by Proposition 3.6 1 we have*

$$(S_1\ \overline{C}) \geq \overline{G} \tag{90}$$

We claim that we can now show that, for some substitution S_3 *satisfying* $S_3 \circ S_u = S_1$, *we have* $(S_3\ \overline{D}) \geq \overline{G}$ *and* $(S_3\ (S_u\ (\Delta_P \backslash \alpha))) \geq \Delta'$, *from which (as we shall then show) we can complete the case easily. We consider two cases:*

($\beta \in \Delta_P$)**:** *Then, by* (*Eq.* 87), *we have* $\beta \in \Delta'$. *Since* $\Delta = \beta : \overline{G}, \Delta'$, *we must have* $\beta : \overline{G} \in \Delta'$. *Now, let* $\overline{H}$ = *typeof* $\beta\ \Delta_P$. *Then* $(S_1\ \overline{H}) \geq \overline{G}$. *By* (*Eq.* 90) *and Theorem 3.25 2 (and following* (*Eq.* 82)*), there exists* S_3 *such that* $S_3 \circ S_u = S_1$ *and* $(S_3\ \overline{D}) \geq \overline{G}$, *and therefore, by* (*Eq.* 87), *we obtain* $(S_3 \circ S_u\ (\Delta_P \backslash \alpha)) \geq \Delta'$ *as claimed.*

($\beta \notin \Delta_P$)**:** *Then, by* (*Eq.* 82), $(S_u, \overline{D}) = (id, \overline{C})$. *Let* $S_3 = S_1$, *and then trivially we have* $S_3 \circ S_u = S_1$ *and* $(S_3\ \overline{D}) \geq \overline{G}$ *(from* (*Eq.* 90)*) and* $(S_3 \circ S_u\ (\Delta_P \backslash \alpha)) \geq \Delta'$ *(by* (*Eq.* 87)*).*

Therefore, in both cases, we have:

$$(S_3\ \overline{D}) \geq \overline{G} \tag{91}$$
$$(S_3\ (S_u\ (\Delta_P \backslash \alpha))) \geq \Delta' \tag{92}$$
$$S_3 \circ S_u = S_1 \tag{93}$$

Therefore, we can deduce $(S_3\ (S_u\ (\Delta_P \backslash \alpha \backslash \beta))) \geq (\Delta' \backslash \beta)$*, and so it follows that* $(S_3\ (S_u\ (\Delta_P \backslash \alpha \backslash \beta)), \beta : \overline{D}) \geq \Delta'$ *as needed. Finally, by combining* (*Eq.* 86) *with* (*Eq.* 93)*, and applying Lemma 3.23 10, we obtain* $S_E \circ S = S_1 \circ S_P = S_3 \circ S_u \circ S_P = (S_3 \circ ((S_u \circ S_P) \backslash atoms(\Gamma))) \circ ((S_u \circ S_P) \cap atoms(\Gamma))$*. Now, noting that* $dom(S_E) = \{\varphi\}$*, we apply Lemma 3.22 2 and deduce that there exists a substitution* S_5 *such that* $S = S_5 \circ ((S_u \circ S_P) \cap atoms(\Gamma))$*, as required.*

$R = P\widehat{\alpha}\,[x]\,\widehat{y}Q$**:** *In accordance with the algorithm, we have:*

$$\langle S_P, \Delta_P \rangle = \mathrm{sppc}\,(P, \Gamma) \tag{94}$$
$$\varphi = fresh \tag{95}$$
$$\langle S_Q, \Delta_Q \rangle = \mathrm{sppc}\,(Q, (S_P\ \Gamma) \cup \{y : \varphi\}) \tag{96}$$
$$A = freshInstance\ typeof\ \alpha\ (S_Q\ \Delta_P) \tag{97}$$
$$B = (S_Q\ \varphi) \tag{98}$$
$$C = freshInstance\ typeof\ x\ (S_Q \circ S_P\ \Gamma) \tag{99}$$
$$S_u = unify\ C\ A \rightarrow B \tag{100}$$
$$\langle S_c, \Delta_c \rangle = unifyGenContexts\ (S_u \circ S_Q\ \Delta_P \backslash \alpha)\ (S_u\ \Delta_Q) \tag{101}$$
$$S_r = (S_c \circ S_u \circ S_Q \circ S_P \cap atoms(\Gamma)) \tag{102}$$
$$\mathrm{sppc}\,(P\widehat{\alpha}\,[y]\,\widehat{x}Q, \Gamma) = \langle S_r, \Delta_c \rangle \tag{103}$$

By Lemma 3.9 3, we have, for some Γ'*,*$\overline{D}$*,*E *and* F*, that*

$$\Gamma = \Gamma', x : \overline{D} \tag{104}$$
$$(S\ \overline{D}) \geq (E \rightarrow F) \tag{105}$$
$$P :\cdot (S\ \Gamma') \vdash_{\mathrm{SP}} \alpha : E, \Delta \tag{106}$$
$$Q :\cdot (S\ \Gamma'), y : F \vdash_{\mathrm{SP}} \Delta \tag{107}$$

For reference, we explicitly write

$$\overline{D} = \overrightarrow{\forall X_i}.D \tag{108}$$

By induction, using (*Eq.* 94) *and* (*Eq.* 106)*, there exists* S_1 *such that:*

$$S = S_1 \circ S_P \tag{109}$$

$$(S_1\ \Delta_P) \geq (\alpha : E, \Delta) \tag{110}$$

By (*Eq.* 104), (*Eq.* 107) *and weakening (Proposition 3.8 2) as necessary, we obtain*

$$Q :\cdot (S\ \Gamma), y : F \vdash_{\mathrm{SP}} \Delta \tag{111}$$

Now, let

$$S_F = \{(\varphi \mapsto F)\} \tag{112}$$

Then $(S_F \circ S_1\ ((S_P\ \Gamma), y : \varphi)) = ((S\ \Gamma), y : F)$ *by construction. Using* (*Eq.* 96) *and* (*Eq.* 111)*, by induction, there exists* S_2 *such that:*

$$S_F \circ S_1 = S_2 \circ S_Q \tag{113}$$

$$(S_2\ \Delta_Q) \geq \Delta \tag{114}$$

Now define (respecting (*Eq.* 97)*):*

$$\overline{A} = \mathit{typeof}\ \alpha\ (S_Q\ \Delta_P) \tag{115}$$

$$A = \mathit{freshInst}(\overline{A}) \tag{116}$$

By (*Eq.* 108) *amd* (*Eq.* 99)*,* $C = (S_Q \circ S_P\ D[\overrightarrow{\varphi_i/X_i}])$ *for fresh* $\overrightarrow{\varphi_i}$*. Now, using* (*Eq.* 113) *and* (*Eq.* 109)*, we have*

$$S_2 \circ S_Q \circ S_P = S_F \circ S_1 \circ S_P = S_F \circ S \tag{117}$$

and so $(S_2\ C) = (S_2\ S_Q \circ S_P\ D[\overrightarrow{\varphi_i/X_i}]) = (S\ D)[\overrightarrow{\varphi_i/X_i}]$ *given* (*Eq.* 112)*. By* (*Eq.* 105) *and Proposition 3.6 7, there exists* S_3 *such that*

$$\mathit{dom}(S_3) = \{\overrightarrow{\varphi_i}\} \tag{118}$$

$$(S_3 \circ S_2\ C) = (E \rightarrow F) \tag{119}$$

By (*Eq.* 110),(*Eq.* 95) *and* (*Eq.* 114) *we have* $(S_F \circ S_1\ \Delta_P) \geq (\alpha : E, \Delta)$*. By* (*Eq.* 113)*, this means that* $(S_2 \circ S_Q\ \Delta_P) \geq (\alpha : E, \Delta)$*, and in particular, by* (*Eq.* 115)*, we have* $(S_2\ \overline{A}) \geq E$*. By Proposition 3.6 7 and* (*Eq.* 116) *(in which, say* $\{\overrightarrow{\varphi_j}\}$ *are the fresh atomic types chosen), there exists* S_4 *such that*

$$\mathit{dom}(S_4) = \{\overrightarrow{\varphi_j}\} \tag{120}$$

$$(S_4 \circ S_2\ A) = E \tag{121}$$

(note that (*Eq.* 116) *implies that, up to choice of fresh atomic types,* $(S_2\ A) = \mathit{freshInst}(S_2\ \overline{A})$*, given that* S_2 *does not clash with the atomic types chosen).*

Due to (*Eq.* 118), (*Eq.* 120) *and* (*Eq.* 121)*, we deduce* $(S_4 \circ S_3 \circ S_2\ A) = E$*. Also, by* (*Eq.* 98)*,* $(S_4 \circ S_3 \circ S_2\ B) = (S_4 \circ S_3 \circ S_2 \circ S_Q\ \varphi) = F$*. By* (*Eq.* 119)*, we have* $(S_4 \circ S_3 \circ S_2\ C) = E \rightarrow F = (S_4 \circ S_3 \circ S_2\ A \rightarrow B)$*. By completeness of unification (Lemma 3.18 2), there exists a substitution* S_5 *such that*

$$S_4 \circ S_3 \circ S_2 = S_5 \circ S_u \tag{122}$$

Now, using (*Eq.* 122), (*Eq.* 113), (*Eq.* 118) *and* (*Eq.* 120)*, we obtain:*

$$\begin{aligned}
(S_5 \circ S_u \circ S_Q\ \Delta_P \backslash \alpha) &= (S_4 \circ S_3 \circ S_2 \circ S_Q\ \Delta_P \backslash \alpha) && (\mathit{Eq.}\ 122)\\
&= (S_4 \circ S_3 \circ S_F \circ S_1\ \Delta_P \backslash \alpha) && (\mathit{Eq.}\ 113)\\
&= (S_1\ \Delta_P \backslash \alpha) && (\mathit{Eq.}\ 118), (\mathit{Eq.}\ 120), (\mathit{Eq.}\ 112)\\
&\geq \Delta && (\mathit{Eq.}\ 110)
\end{aligned}$$

Similarly, we deduce $(S_5 \circ S_u\ \Delta_Q) = (S_2\ \Delta_Q) \geq \Delta$ *using* (*Eq.* 114)*. Therefore, by* (*Eq.* 101)*, there exists* S_6 *with* $S_5 = S_6 \circ S_5$ *and* $(S_6\ \Delta_C) \geq \Delta$*.*
Now, $S_4 \circ S_3 \circ S_F \circ S = S_4 \circ S_3 \circ S_2 \circ S_Q \circ S_P = S_5 \circ S_u \circ S_Q \circ S_P = S_6 \circ S_c \circ S_u \circ S_Q \circ S_P$*.*
Therefore, $S_4 \circ S_3 \circ S_F \circ S = S_6 \circ ((S_c \circ S_u \circ S_Q \circ S_P) \backslash \mathit{atoms}(\Gamma)) \circ ((S_c \circ S_u \circ S_Q \circ S_P) \cap \mathit{atoms}(\Gamma))$*, by Lemma 3.23 10. By applying Lemma 3.22 2, using* (*Eq.* 118)*,* (*Eq.* 120) *and* (*Eq.* 112)*, there exists* S_7 *such that* $S = S_7 \circ S_r$ *as required.*

$R = P\widehat{\alpha} \dagger \widehat{x}Q$**:** *In accordance with the algorithm, we have:*

$$\begin{aligned}
\langle S_P, \Delta_P \rangle &= \mathrm{sppc}\,(P, \Gamma) && (123)\\
\overline{A} &= \mathit{typeof}\ \alpha\ \Delta_P && (124)\\
\langle S_Q, \Delta_Q \rangle &= \mathrm{sppc}\,(Q, (S_P\ \Gamma) \cup \{x : \overline{A}\}) && (125)\\
\langle S_c, \Delta_c \rangle &= \mathit{unifyGenContexts}\ (S_Q\ \Delta_P \backslash \alpha)\ \Delta_Q && (126)\\
S_r &= (S_c \circ S_Q \circ S_P \cap \mathit{atoms}(\Gamma)) && (127)\\
\mathrm{sppc}\,(P\widehat{\alpha} \dagger \widehat{x}Q, \Gamma) &= \langle S_r, \Delta_c \rangle && (128)
\end{aligned}$$

By Lemma 3.9 4, there exists $\overline{B}$ *such that*

$$P :\cdot\ (S\ \Gamma) \vdash_{\mathrm{SP}} \alpha : \overline{B}, \Delta \tag{129}$$

$$Q :\cdot\ (S\ \Gamma), x : \overline{B} \vdash_{\mathrm{SP}} \Delta \tag{130}$$

By induction, using (*Eq*. 123)*, there exists* S_1

$$S = S_1 \circ S_P \tag{131}$$
$$(S_1\ \Delta_P) \geq (\alpha : \overline{B}, \Delta) \tag{132}$$

By (*Eq*. 124)*,* $(S_1\ \overline{A}) \geq \overline{B}$*. By* (*Eq*. 130) *and Proposition 3.8 4, we obtain*

$$Q :\cdot (S\ \Gamma), x : (S_1\ \overline{A}) \vdash_{SP} \Delta \tag{133}$$

. Note that

$$(S_1\ ((S_P\ \Gamma), x : \overline{A})) = ((S\ \Gamma), (S_1\ \overline{A}) :) \tag{134}$$

Therefore, by induction, using (*Eq*. 125)*, there exists* S_2 *such that*

$$S_1 = S_2 \circ S_Q \tag{135}$$
$$(S_2\ \Delta_Q) \geq \Delta \tag{136}$$

By (*Eq*. 132)*,* $(S_2\ S_Q\ \Delta_P \backslash \alpha) \geq \Delta$*. By* (*Eq*. 132)*,*(*Eq*. 136)*,*(*Eq*. 126) *and Theorem 3.25 2, there exists* S_3 *with* $S_2 = S_3 \circ S_c$*. Using* (*Eq*. 131) *and* (*Eq*. 135)*, we obtain as required:*

$$\begin{aligned} S &= S_3 \circ S_c \circ S_Q \circ S_P \\ &= (S_3 \circ ((S_c \circ S_Q \circ S_P) \backslash atoms(\Gamma))) \circ ((S_c \circ S_Q \circ S_P) \cap atoms(\Gamma)) \end{aligned}$$

References

[1] T. Griffin, A formulae-as-types notion of control, in: Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages, POPL'90, ACM Press, 1990, pp. 47–57.

[2] H. B. Curry, Functionality in Combinatory Logic, in: Proc. Nat. Acad. Sci. U.S.A., Vol. 20, 1934, pp. 584–590.

[3] H. B. Curry, R. Feys, Combinatory Logic Vol. I, North-Holland, Amsterdam, 1958.

[4] W. A. Howard, The formulae-as-types notion of construction, in: To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press, 1980, pp. 479–490.

[5] F. Barbanera, S. Berardi, A symmetric lambda-calculus for classical program extraction, in: Proceedings of TACS '94, Vol. 789 of Lecture Notes in Computer Science, Springer-Verlag, 1994.

[6] P.-L. Curien, H. Herbelin, The duality of computation, in: Proc. ICFP'00, ACM, 2000, pp. 233–243.

[7] M. Parigot, Proofs of strong normalisation for second order classical natural deduction, Journal of Symbolic Logic 62 (4) (1997) 1461–1479.

[8] C. Urban, Classical logic and computation, Ph.D. thesis, University of Cambridge (October 2000).

[9] Z. M. Ariola, H. Herbelin, Minimal classical logic and control operators, in: Proc. ICALP '03, Vol. 2719 of LNCS, Springer, 2003, pp. 871–885.

[10] Z. M. Ariola, H. Herbelin, A. Sabry, A type-theoretic foundation of continuations and prompts, in: ICFP '04: Proceedings of the ninth ACM SIGPLAN international conference on Functional programming, ACM, New York, NY, USA, 2004, pp. 40–53. doi:http://doi.acm.org/10.1145/1016850.1016860.

[11] P. de Groote, On the relation between the lambda-mu-calculus and the syntactic theory of sequential control, in: LPAR '94: Proc. 5th International Conference on Logic Programming and Automated Reasoning, Springer-Verlag, London, UK, 1994, pp. 31–43.

[12] C.-H. L. Ong, C. A. Stewart, A Curry-Howard foundation for functional computation with control, in: Proc. 24th Symp. on Principles of Programming Languages, ACM Press, New York, 1997, pp. 215–227.
URL `citeseer.ist.psu.edu/ong97curryhoward.html`

[13] T. Streicher, B. Reus, Classical logic, continuation semantics and abstract machines, Journal of Functional Programming 8 (6) (1998) 543–572.
URL `citeseer.ist.psu.edu/streicher98classical.html`

[14] A. J. Summers, Curry-howard term calculi for gentzen-style classical logics, Ph.D. thesis, Imperial College, University of London (November 2008).

[15] A. J. Summers, S. van Bakel, Approaches to polymorphism in classical sequent calculus, in: ESOP, 2006, pp. 84–99.

[16] J. Robinson, A machine-oriented logic based on the resolution principle, Journal of the ACM 12 (1) (1965) 23–41.

[17] J. Hindley, The principal type scheme of an object in combinatory logic, Transactions of the American Mathematical Society 146 (1969) 29–60.

[18] R. Milner, A theory of type polymorphism in programming, Journal of Computer and System Sciences 17 (1978) 348–375.

[19] J.-Y. Girard, Une extension de l'interprétation de Gödel à l'analyse et son application l'élimination des coupures dans l'analyse et la théorie des types, in: Proceedings of the Second Scandinavian Logic Symposium (Univ. Oslo, Oslo, 1970), Vol. 63, North-Holland, 1971, pp. 63–92.

[20] J. Reynolds, Towards a Theory of Type Structures, in: B. Robinet (Ed.), Proceedings of 'Colloque sur la Programmation', Paris, France, Vol. 19 of Lecture Notes in Computer Science, Springer-Verlag, 1974, pp. 408–425.

[21] L. Damas, R. Milner, Principal type-schemes for functional programs, in: Proceedings 9th ACM Symposium on Principles of Programming Languages, 1982, pp. 207–212.

[22] J. B. Wells, The essence of principal typings, in: Proc. 29th Int'l Coll. Automata, Languages, and Programming, Vol. 2380 of LNCS, Springer-Verlag, 2002, pp. 913–925.

[23] J. B. Wells, Typability and type checking in System F are equivalent and undecidable, Ann. Pure Appl. Logic 98 (1–3) (1999) 111–156.

[24] G. Gentzen, Untersuchungen über das logische schließen, Mathematische Zeitschrift 39 (1935) 176–210, 405–431.

[25] S. van Bakel, S. Lengrand, P. Lescanne, The language $\mathcal{X}$: circuits, computations and classical logic, in: Proc. ICTCS'05, Vol. 3701 of LNCS, Springer-Verlag, 2005, pp. 66–80.

[26] R. Harper, M. Lillibridge, Ml with callcc is unsound, message sent to the "types" mailing list. Archived at: `http://www.seas.upenn.edu/~sweirich/types/archive/1991/msg00034.html` (1991).

[27] K.-e. Fujita, Explicitly typed lambda μ-calculus for polymorphism an call-by-value, in: TLCA '99: Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications, Springer-Verlag, London, UK, 1999, pp. 162–176.

[28] M. Tofte, Type inference for polymorphic references, Inf. Comput. 89 (1) (1990) 1–34. doi:http://dx.doi.org/10.1016/0890-5401(90)90018-D.

[29] X. Leroy, Polymorphism by name for references and continuations, in: POPL '93: Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM, New York, NY, USA, 1993, pp. 220–231. doi:http://doi.acm.org/10.1145/158511.158632.

[30] A. Wright, Polymorphism for imperative languages without imperative types, Tech. Rep. TR93-200 (21, 1993).
URL `citeseer.ist.psu.edu/wright93polymorphism.html`

[31] J. C. Mitchell, G. D. Plotkin, Abstract types have existential type, ACM Trans. Program. Lang. Syst. 10 (3) (1988) 470–502. doi:http://doi.acm.org/10.1145/44501.45065.