# Typability and Type Checking in the Second-Order $\lambda$ -Calculus Are Equivalent and Undecidable \* (Preliminary Draft)

J. B. Wells
Dept. of Computer Science
jbw@cs.bu.edu
Boston University
Boston, MA 02215, USA

August 1, 1993

#### Abstract

We consider the problems of  $typability^1$  and  $type\ checking^2$  in the Girard/Reynolds secondorder polymorphic typed  $\lambda$ -calculus, for which we use the short name "System  $\mathbf{F}$ " and which we use in the "Curry style" where types are assigned to pure  $\lambda$ -terms. These problems have been considered and proven to be decidable or undecidable for various restrictions and extensions of System  $\mathbf{F}$  and other related systems, and lower-bound complexity results for System  $\mathbf{F}$  have been achieved, but they have remained "embarrassing open problems" for System  $\mathbf{F}$  itself. We first prove that type checking in System  $\mathbf{F}$  is undecidable by a reduction from semi-unification. We then prove typability in System  $\mathbf{F}$  is undecidable by a reduction from type checking. Since the reverse reduction is already known, this implies the two problems are equivalent. The second reduction uses a novel method of constructing  $\lambda$ -terms such that in all type derivations, specific bound variables must always be assigned a specific type. Using this technique, we can require that specific subterms must be typable using a specific, fixed type assignment in order for the entire term to be typable at all. Any desired type assignment may be simulated. We develop this method, which we call "constants for free", for both the  $\lambda K$  and  $\lambda I$  calculi.

<sup>\*</sup>This work is partly supported by NSF grant CCR-9113196.

<sup>&</sup>lt;sup>1</sup>Typability is also called type reconstruction.

<sup>&</sup>lt;sup>2</sup>Type checking is also called derivation reconstruction.

<sup>&</sup>lt;sup>3</sup>Robin Milner as quoted by Henk Barendregt in [1].

# 1 Basic Definitions and Notational Conventions

In this section we present definitions, notation, and nomenclature. We state precisely the problems of typability, type checking, and semi-unification.

# 1.1 System F

#### 1.1.1 $\lambda$ -Terms

 $\mathcal{V}$  is a countably infinite set of *object variables* (as opposed to *type variables*). We use small italic roman letters, e.g. v, w, x, y, z, etc., (possibly subscripted or primed) as metavariables ranging over  $\mathcal{V}$ .

The set  $\Lambda$  of  $\lambda$ -terms is the least such that:

$$\Lambda \supseteq \mathcal{V} \cup \{ (M \ N) \mid M, N \in \Lambda \} \cup \{ (\lambda x.M) \mid x \in \mathcal{V}, M \in \Lambda \}$$

A  $\lambda$ -term is therefore either an object variable or an application or an abstraction. We use capital italic roman letters, e.g. M, N, P, Q, R, S, etc., as metavariables ranging over  $\Lambda$ . With no loss of generality, we assume for every term  $M \in \Lambda$  that every object variable in M is bound at most once and no variable in M is both bound and free. If x is bound in M, it will be convenient to distinguish between the binding occurrence of x in M, always preceded by the symbol  $\lambda$ , and the bound occurrences of x.

We denote by FV(M) and BV(M) the sets of free and bound object variables in M, respectively.

If M and N are  $\lambda$ -terms, by  $M \equiv N$  we mean that M and N are identical. We write  $M \subset N$  to mean that M is a proper subterm of N, and  $M \subseteq N$  to include the possibility that  $M \equiv N$ .

We define K to be the usual K-combinator  $(\lambda x.\lambda y.x)$ .

#### 1.1.2 Types

 $\mathbb{V}$  is the set of all type variables. We use small Greek letters near the beginning of the alphabet, e.g.  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\delta$ , etc., (possibly subscripted or primed) as metavariables ranging over  $\mathbb{V}$ . We use  $\vec{\alpha}$ ,  $\vec{\beta}$ , etc., to denote finite sequences of type variables.

T is the set of all second-order types over V, which is the least set such that:

$$\mathbb{T} \supseteq \mathbb{V} \cup \{\, (\tau \!\to\! \tau') \mid \tau, \tau' \in \mathbb{T} \,\} \cup \{\, (\forall \alpha.\tau) \mid \alpha \in \mathbb{V}, \tau \in \mathbb{T} \,\}$$

A type is therefore either a type variable or a  $\rightarrow$ -type or a  $\forall$ -type. We use small Greek letters towards the end of the alphabet, e.g.  $\mu$ ,  $\nu$ ,  $\pi$ ,  $\rho$ ,  $\sigma$ ,  $\tau$ , etc., as metavariables ranging over  $\mathbb{T}$ . With no loss of generality, we assume for every type  $\tau \in \mathbb{T}$  that every type variable in  $\tau$  is bound at

most once and no variable in  $\tau$  is both bound and free. If  $\alpha$  is bound in  $\tau$ , it will be convenient to distinguish between the *binding* occurrence of  $\alpha$  in  $\tau$ , always preceded by the symbol  $\forall$ , and the bound occurrences of  $\alpha$ .

 $FTV(\tau)$  and  $BTV(\tau)$  denote the free and bound type variables of type  $\tau$ , respectively.

If  $\sigma = \forall \alpha. \tau$  and  $\alpha \notin FTV(\tau)$ , we say that " $\forall \alpha$ " is a redundant binding. We assume all types do not contain redundant bindings, without any loss of generality.

In this paper we consider two types  $\sigma$  and  $\tau$  to be the same if they can be made identical after  $\alpha$ -conversion and permutation of adjacent binding occurrences. For example,  $\forall \alpha. \forall \beta. \alpha \rightarrow \beta = \forall \beta. \forall \alpha. \alpha \rightarrow \beta \text{ and } \forall \alpha. \alpha \rightarrow \alpha = \forall \beta. \beta \rightarrow \beta.$ 

If  $\vec{\alpha}$  is the sequence  $\alpha_1 \cdots \alpha_k$ , we may write  $\forall \vec{\alpha}.\sigma$  instead of  $\forall \alpha_1, \dots, \forall \alpha_k.\sigma$ . As we do not distinguish between permutations of adjacent binding occurrences, we can usually view  $\vec{\alpha}$  as the set  $\{\alpha_1, \dots, \alpha_k\}$ . If  $\vec{\alpha} = \text{FTV}(\tau)$ , then we may write  $\forall .\tau$  as shorthand for  $\forall \vec{\alpha}.\tau$ , and the order of the type variables in  $\vec{\alpha}$  does not matter. On the other hand, when we simultaneously substitute types  $\tau_1, \dots, \tau_k$  for type variables  $\alpha_1, \dots, \alpha_k$  that are free in  $(\sigma)$ , we may write  $\sigma[\vec{\alpha}:=\vec{\tau}]$  instead of the more explicit  $\sigma[\alpha_1:=\tau_1,\dots,\alpha_k:=\tau_k]$ , and in this case the ordering of the variables in  $\vec{\alpha}$  determines which type gets substituted for each variable. We may write  $\bot$  as shorthand for  $\forall \alpha.\alpha$ .

For this paper, we define the notion of proper subtype, denoted as  $\sigma \subset \tau$ , as the smallest transitive relation satisfying the following property:

$$\sigma \subset \tau$$
 if there exist  $\vec{\alpha}$  and  $\rho$  such that  $\tau = \forall \vec{\alpha}. \sigma \rightarrow \rho$  or  $\tau = \forall \vec{\alpha}. \rho \rightarrow \sigma$ 

Note that this definition is unusual since  $\forall \beta. \sigma \not\subset \forall \alpha. \forall \beta. \sigma$ . We write  $\sigma \subseteq \tau$  to include the possibility that  $\sigma = \tau$ .

# 1.1.3 Derivations

We present System **F** in "Curry style", according to which untyped terms are assigned type expressions. The derivation rules of System **F** are shown in Figure 1.

We call a pair  $x:\sigma$  where  $x \in \mathcal{V}$  and  $\sigma \in \mathbb{T}$  a type assumption.<sup>4</sup> A finite sequence of type assumptions  $x_1:\sigma_1,\ldots,x_n:\sigma_n$  which associates at most one type  $\sigma$  with each variable x is a type assignment.<sup>5</sup> We call an expression of the form  $A \vdash M : \tau$  where A is a type assignment,  $M \in \Lambda$ , and  $\tau \in \mathbb{T}$  an assertion.<sup>6</sup> Since we assume that no object variable is bound twice in a  $\lambda$ -term, we may freely view a type assignment as a set.

We can view a type assignment A as a partial function from  $\mathcal{V}$  to  $\mathbb{T}$ . Thus, if we write  $A(x) = \sigma$ , this is the same as saying that the pair  $x:\sigma$  is in A. For  $A = \{x_1:\sigma_1,\ldots,x_n:\sigma_n\}$ , we define

$$FTV(A) = FTV(\sigma_1) \cup \cdots \cup FTV(\sigma_n)$$

<sup>&</sup>lt;sup>4</sup>A type assumption is also called a declaration or a type assignment.

<sup>&</sup>lt;sup>5</sup>A type assignment is also called a basis, an environment, or a context.

<sup>&</sup>lt;sup>6</sup>An assertion is also called a sequent, a type assignment formula, or a judgement.

VAR 
$$A \vdash x : \sigma \qquad A(x) = \sigma$$

$$APP \qquad \frac{A \vdash M : \sigma \rightarrow \tau \quad A \vdash N : \sigma}{A \vdash (M \mid N) : \tau}$$

$$ABS \qquad \frac{A \cup \{x : \sigma\} \vdash M : \tau}{A \vdash (\lambda x \mid M) : \sigma \rightarrow \tau}$$

$$INST \qquad \frac{A \vdash M : \forall \alpha . \sigma}{A \vdash M : \sigma [\alpha := \tau]}$$

$$GEN \qquad \frac{A \vdash M : \sigma}{A \vdash M : \forall \alpha . \sigma} \qquad \alpha \notin FTV(A)$$

Figure 1: Derivation Rules of System F.

A derivation  $\mathcal{D}$  in System  $\mathbf{F}$  is a finite sequence of assertions  $\Delta_1, \ldots, \Delta_n$  for some  $n \geq 1$ , where each assertion  $\Delta_j$  is obtained from one or two of the preceding assertions  $\Delta_1, \ldots, \Delta_{j-1}$  according to the inference rules of System  $\mathbf{F}$ . A typing of the  $\lambda$ -term M in System  $\mathbf{F}$  is a derivation in System  $\mathbf{F}$  whose last assertion is  $A \vdash M : \tau$  for some type assignment A and type  $\tau$ . We say that a  $\lambda$ -term M is typable in System  $\mathbf{F}$  if and only if there is a typing of M in System  $\mathbf{F}$ .

The typability problem: Given an arbitrary  $M \in \Lambda$ , is M typable in System F?

The type-checking problem: Given arbitrary  $M \in \Lambda$ , type assignment A, and type  $\tau \in \mathbb{T}$ , is there a valid derivation in System F that ends with the assertion  $A \vdash M : \tau$ ?

### 1.2 Semi-Unification

We present here not the full semi-unification problem, but rather the semi-unification problem restricted to a signature with a single binary function symbol. The definition is chosen because it will conveniently allow mapping semi-unification terms onto types of System  $\mathbf{F}$ .

We restrict ourselves to a first-order signature containing the single binary function symbol " $\rightarrow$ ". We use  $\mathbb{V}$  as the set of semi-unification variables. The set of semi-unification terms  $\mathcal{T}$  is the least set satisfying the relation:

$$\mathcal{T} \supseteq \mathbb{V} \cup \{ (\sigma \rightarrow \tau) \mid \sigma, \tau \in \mathcal{T} \}$$

It can easily be seen from this definition that  $\mathcal{T} \subseteq \mathbb{T}$ , the set of System **F** types. An *instance*  $\Gamma$  of semi-unification is a finite set of pairs:

$$\Gamma = \{ \tau_1 \dot{\leq} \mu_1, \ldots, \tau_n \dot{\leq} \mu_n \}$$

where for all  $i \in \{1, ..., n\}$  it is the case that  $\tau_i, \mu_i \in \mathcal{T}$ . A substitution is a function  $S : \mathbb{V} \mapsto \mathcal{T}$ . A substitution extends naturally to a  $\rightarrow$ -homomorphism  $S : \mathcal{T} \mapsto \mathcal{T}$ . A substitution S is a solution for an instance  $\Gamma$  of semi-unification if there also exist substitutions  $S_1, ..., S_n$  such that:

$$S_1(S(\tau_1)) = S(\mu_1), \ldots, S_n(S(\tau_n)) = S(\mu_n)$$

The semi-unification problem: Given an arbitrary instance  $\Gamma$  of semi-unification, does  $\Gamma$  have a solution?

# 2 Reduction from Semi-Unification to Type Checking

**Theorem 2.1** Semi-unification is reducible to type checking in System **F**. Type checking in System **F** is therefore undecidable.

**Proof:** Consider any instance  $\Gamma$  of the semi-unification problem of the following form:

$$\Gamma = \{ \tau_1 \dot{\leq} \mu_1, \ldots, \tau_n \dot{\leq} \mu_n \}$$

For all  $i \in \{1, ..., n\}$ , when we consider  $\tau_i$  and  $\mu_i$  as types rather than as semi-unification terms, it holds that  $BTV(\tau_i) = BTV(\mu_i) = \emptyset$  and  $FTV(\tau_i) \cup FTV(\mu_i) \subseteq \{\alpha_1, ..., \alpha_m\}$ . We now contruct an instance of the type-checking problem from the instance of the semi-unification problem. First, we contruct a type assignment A, which appears in figure 2. Then, we construct a  $\lambda$ -term M:

$$M \equiv Ky(p(\lambda j.j(e(\lambda c_1...c_m.K(g(d_1\vec{c})\cdots(d_n\vec{c})))))))))$$
$$(\Box(k(f_1\vec{c})(q_1.(j\Box)))\cdots(k(f_n\vec{c})(q_n(j\Box)))))))).$$

It can be checked that  $\Gamma$  has a solution if and only if there is a typing in System  $\mathbf{F}$  ending with the assertion  $A \vdash M : \alpha$ . (Full details will be in the final report.) Since semi-unification with a signature containing at least one function symbol of arity  $\geq 2$  is shown to be undecidable in [2], we can conclude that the problem of type checking in System  $\mathbf{F}$  is undecidable.

# 3 Constants for Free in System F

# 3.1 The Beginning

Consider this  $\lambda$ -term:

$$J \equiv (\lambda r.(\lambda y.\lambda z.r(yy)(yz))(\lambda x.Kx(x(xr)))(\lambda w.ww))$$

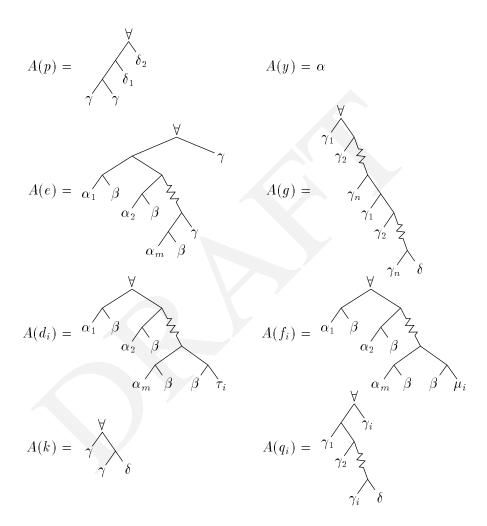


Figure 2: Type Assignment A.

Clearly J is typable with the following global type assignment:

$$A(r) = \bot$$

$$A(y) = \bigvee_{\alpha \quad \alpha \quad \alpha} A(z) = \bigvee_{\alpha \quad \alpha} A(z) = \bigvee_{\alpha \quad \alpha} A(z) = \bigvee_{\alpha \quad \alpha} A(z) = \bigcup_{\alpha \quad \alpha} A(z) =$$

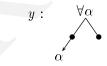
It is the case that in all derivations for J, the object variables x and y must be assigned the same types. (For x this means it has the type  $\alpha \rightarrow \alpha$  for some free type variable  $\alpha$ .) The proof follows.

Because of the subterm (ww), we know that the type variable at the leaf at the end of the leftmost path in the type assigned to w is quantified at the root of the type. We know the same for y. We depict this as follows:

$$w: \bigvee_{\gamma}^{\forall \gamma} y: \bigvee_{\alpha}^{\forall \alpha}$$

Since the abstraction over y is applied to the abstraction over x, we know that the type assigned to y has a " $\rightarrow$ " in it, which we depict like this:

Combining the two diagrams, we get this result:



Now we know the type of y matches the type derived for the abstraction over x. The only way quantification can occur at the root of the type of the abstraction over x is if the GEN rule is applied at the last step. Thus, we know that an earlier type derived for the abstraction over x looks like this, where  $\alpha$  is a free variable:

$$(\lambda x.Kx(x(xr))):$$

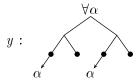
(We can assume INST never occurs after ABS.) Thus, we know this about the type assigned to x:

$$x: \bigwedge^{\bullet}$$

Due to the subterm (xr) we know the type of x cannot be simply  $\alpha$ , hence we know this:



Thus we know this about the type assigned to y:



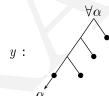
Considering again the type assigned to w and how this type must be embedded in the type assigned to z, we know this:

$$z: \frac{\forall \gamma}{\gamma}$$

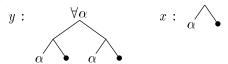
Consider the subterm (yz). The types assigned to y and z must be instantiated so that the left subtree of the instantiation of the type of y matches the instantiation of the type of z. We know that every instantiation of the type of z will match the pattern given in (1) for the type assigned to z. Thus, we know that the type assigned to y must be instantiated to match this pattern:

$$y: \begin{array}{c} \forall \gamma \\ \gamma \end{array}$$

Now suppose the leftmost path in the type assigned to y is at least 3 edges long, in other words suppose this:



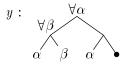
If this were the case, then the instantiation of the type assigned to of y could never match the pattern in (2) because a quantifier owning the leftmost path cannot be inserted at the necessary spot in (2) by instantiation. Thus, we know the leftmost path in the type of y must be exactly 2 edges long, and thus the leftmost path in the type of x must be 1 edge long:



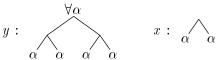
In the subterm (x(xr)), we know the type derived for (xr) must eventually be  $\alpha$ . Thus we know the type assigned to x must match one of these two patterns:

$$x: \bigwedge_{\alpha = \alpha} \quad \text{or} \quad x: \bigwedge_{\alpha = \beta}^{\forall \beta}$$

Suppose it were the latter, i.e.  $x : \forall \beta.(\alpha \rightarrow \beta)$ . Then the type assigned to y would have to match this pattern:



However, then the subterm (yy) could not be typed. Thus, we know both of these types in all typings of J:



Now that we have a term in which a certain variable must always be assigned the same type, we must figure out how to use this.

# 3.2 Using Contexts to Establish Constraints

The term context refers to a  $\lambda$ -term with one or more holes. We define here a notation to convert a normal  $\lambda$ -term into a context with one hole. For a  $\lambda$ -term M matching the following pattern:

$$(\cdots(\lambda x_1,\cdots(\lambda x_2,\cdots(\lambda x_n,P)\cdots)\cdots)\cdots)$$

we define  $C(M, \{x_1, \ldots, x_n\})$  as the following context:

$$(\cdots(\lambda x_1,\cdots(\lambda x_2,\cdots(\lambda x_n,KP[\cdot])\cdots)\cdots)\cdots)$$

Since we assume that all bound variables in a  $\lambda$ -term are named distinctly from each other, this is well-defined. The hole in the context  $C(M, \{x_1, \ldots, x_n\})$  occurs within the scope of the variables  $x_1, \ldots, x_n$ .

If all of these conditions hold:

- 1. The  $\lambda$ -term M is typable in System **F**.
- 2.  $C(M, \{x_1, \ldots, x_n\})$  is defined.
- 3. In all derivations in System F for M it holds that  $x_i$  is assigned the type  $\sigma_i$  exactly.

then we say that  $C(M, \{x_1, \ldots, x_n\})$  is a constant context for the set of types  $\{\sigma_1, \ldots, \sigma_n\}$  at the object variables  $\{x_1, \ldots, x_n\}$ . We also call M itself a constant context.

Given the existence of a  $\lambda$ -term M which is a constant context for  $\{\sigma_1,\ldots,\sigma_n\}$  at  $\{x_1,\ldots,x_n\}$ , we then define, for an arbitrary  $\lambda$ -term N, the notation  $E(N,y_1;\sigma_1,\ldots,y_n;\sigma_n)$  to be the  $\lambda$ -term  $C(M,\{x_1,\ldots,x_n\})[N[\vec{y}:=\vec{x}]]$ . We assume that  $\mathrm{BV}(M)\cap\mathrm{FV}(N)=\varnothing=\mathrm{BV}(M)\cap\mathrm{BV}(N)$ . In this embedding it is deliberate that the bindings of  $x_1,\ldots,x_n$  in M capture the occurrences of  $y_1,\ldots,y_n$  in N that are renamed to  $x_1,\ldots,x_n$ .

If  $\mathrm{FV}(N) = \{y_1, \ldots, y_n\}$ , then it should be clear that  $E(N, y_1 : \sigma_1, \ldots, y_n : \sigma_n)$  is typable in System  $\mathbf{F}$  if and only if there is a typing for N in System  $\mathbf{F}$  ending with an assertion  $A \vdash N : \tau$  where  $A = \{y_1 : \sigma_1, \ldots, y_n : \sigma_n\}$ . Thus, we can pretend that  $y_1, \ldots, y_n$  are now *constants* of types  $\sigma_1, \ldots, \sigma_n$  rather than object variables.

We now proceed to prove that there are constant contexts for every finite set of types. (To be precise, for every finite set of types up to a 1-to-1 renaming of all free variables.)

# 3.3 Preparation

**Lemma 3.1** There are constant contexts for every set  $\{\alpha_1 \rightarrow \alpha_1, \dots, \alpha_n \rightarrow \alpha_n\}$  such that for all  $i, j \in \{1, \dots, n\}$  it holds that  $\alpha_i = \alpha_j$  if and only if i = j.

**Proof:** By induction on the size of the set. Recall the definition of the  $\lambda$ -term J from section 3.1:

$$J \equiv (\lambda r.(\lambda y.\lambda z.r(yy(yz)))(\lambda x.Kx(x(xr)))(\lambda w.ww))$$

The base case of  $\{\alpha_1 \to \alpha_1\}$  is already done; it is J. The induction case proceeds as follows. Given that there is a constant context for the set  $\{\alpha_1 \to \alpha_1, \ldots, \alpha_{n-1} \to \alpha_{n-1}\}$  then the  $\lambda$ -term N defined as follows:

$$N \equiv E(J, y_1: \alpha_1 \rightarrow \alpha_1, \dots, y_{n-1}: \alpha_{n-1} \rightarrow \alpha_{n-1})$$

is a constant context for  $\{\alpha_1 \rightarrow \alpha_1, \dots, \alpha_n \rightarrow \alpha_n\}$ . This holds for the following reasons. In the typing of J, the type of x must match the pattern  $\alpha \rightarrow \alpha$ . Also, in the typing of the subterm  $(\lambda x.Kx(x(xr)))$  generalization must be performed over the type variable  $\alpha$ . If it were the case that  $\alpha = \alpha_i$  for some i < n, then this could not happen. Thus, N is typable only if  $\alpha \neq \alpha_i$  for all i < n.

**Lemma 3.2** There are constant contexts for every set  $\{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n, \forall \alpha.\alpha\}$  such that for all  $i, j \in \{1, \dots, n\}$  it holds that  $\alpha_i = \alpha_j$  if and only if i = j.

**Proof:** We know there is a constant context for  $\{\alpha_1 \rightarrow \alpha_1, \dots, \alpha_n \rightarrow \alpha_n\}$ . Define a  $\lambda$ -term N as follows:

$$N \equiv (\lambda x_1 \dots x_n . \lambda r. r(rr)(y_1r))(y_1\Box) \cdots (y_n\Box)$$

( $\square$  stands for some undetermined free variable not equal to any other.) Define a  $\lambda$ -term P as follows:

$$P \equiv E(N, y_1: \alpha_1 \rightarrow \alpha_1, \dots, y_n: \alpha_n \rightarrow \alpha_n)$$

P is a constant context for the desired set.

# 3.4 Closures of All Open Types with One Variable

**Lemma 3.3** There are constant contexts for every set  $\{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n, \sigma_1, \dots, \sigma_m\}$  such that for all  $i, j \in \{1, \dots, n\}$  it holds that  $\alpha_i = \alpha_j$  if and only if i = j and for all  $i \in \{1, \dots, m\}$  it holds that  $\sigma_i = \forall \alpha_1.\tau_i$  where  $BTV(\tau_i) = \emptyset$  and  $FTV(\tau_i) = \{\alpha_1\}$ .

**Proof:** By induction on types. We define a special metric d on types for the induction:

$$d(\forall \alpha.\tau) = d(\tau)$$

$$d(\rho \rightarrow \tau) = \max(e(\rho), d(\tau))$$

$$d(\alpha) = 0$$

$$e(\forall \alpha.\tau) = e(\tau)$$

$$e(\rho \rightarrow \tau) = \max(e(\rho), e(\tau)) + 1$$

$$e(\alpha) = 1$$

The auxiliary metric  $e(\tau)$  simply measures the height of  $\tau$  (letting  $\alpha$  be of height 1). For a type  $\tau$  such that  $\tau = \rho_1 \rightarrow \cdots \rightarrow \rho_k \rightarrow \alpha$  it holds that  $d(\tau)$  is 0 if k = 0 and otherwise  $d(\tau)$  is the maximum of  $e(\rho_1), \ldots, e(\rho_k)$ .

The base case, where for all  $i \in \{1, ..., m\}$  it holds that  $d(\sigma_i) = 0$ , is exactly the consequence of lemma 3.2. For the induction case we assume the claim holds where for all  $i \in \{1, ..., m\}$  it is the case that  $d(\sigma_i) \leq k$ . We prove it holds where for all  $i \in \{1, ..., m\}$  it is the case that  $d(\sigma_i) \leq k + 1$ . Suppose we are trying to find a constant context for the set:

$$\{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n, \sigma_1, \dots, \sigma_p, \tau_1, \dots, \tau_q\}$$

where for all  $i \in \{1, ..., p\}$  it holds that  $d(\sigma_i) \le k$  and for all  $i \in \{1, ..., q\}$  it holds that  $d(\tau_i) = k+1$ . We construct the desired constant context as follows.

Let 
$$R = \{\rho_{i,j} | 1 \le i \le q \text{ and } 1 \le j \le k(i)\}$$
.  
Let  $S = \{\pi \mid \exists \rho \in R. \ \pi \subseteq \rho\}$ .  
Let  $T = \{\varphi \rightarrow \alpha_1 \mid \exists (\pi \rightarrow \varphi) \in S\}$ .  
Let  $U = S \cup T \cup \{\alpha_1 \rightarrow \alpha_1\}$ .  
Enumerate  $U$  so that  $U = \{\mu_1, \dots, \mu_r\}$ .  
Let  $\nu_i = \begin{cases} \mu_i & \text{if } \mu_i \in \{\alpha_1 \rightarrow \alpha_1, \alpha_1\}, \\ \forall \alpha_1, \alpha_1 \rightarrow \mu_i & \text{otherwise.} \end{cases}$   
Let  $V = \{\nu_1, \dots, \nu_r\}$ .  
Let  $\mu_g = \alpha_1$ . Note that  $\nu_g = \alpha_1$  as well.  
Let  $f$  be a function such that for  $1 \le i \le q$  and  $1 \le j \le k(i)$ , it holds that  $p_{i,j} = \mu_{f(i,j)}$ .  
Let  $\varphi_i(j) = \begin{cases} \mu_j & \text{if } 1 \le j \le r, \\ \rho_{i,j-r} & \text{if } r+1 \le j \le r+k(i). \end{cases}$   
Let  $P_i = \{(c_g^{(i)}(c_h^{(i)}(c_h^{(i)}(c_h^{(i)}))) \mid \varphi_i(h) = \varphi_i(j) \rightarrow \theta \text{ and } \varphi_i(g) = \theta \rightarrow \pi\}$ .  
Enumerate  $P_i$  so that  $P_i = \{P_{i,1}, \dots, P_{i,s(i)}\}$ .  
Let  $N_0 \equiv \hat{r}$  (a  $\lambda$ -term).  
Let  $N_0 \equiv \hat{r}$  (a  $\lambda$ -term).  
Let  $N_i \equiv ((\lambda s_i, \hat{r}(x(s_1 a_1 \dots a_r a_{f(i,1)} \dots a_{f(i,k(i))}))) \qquad (y(s_i b_1 \dots b_r b_{f(i,1)} \dots b_{f(i,k(i))})) \qquad (\lambda c_1^{(i)} \dots c_r^{(i)} c_{r+1}^{(i)} \dots c_{r+k(i)}^{(i)} K c_g^{(i)}(\hat{r}P_{i,1} \dots P_{i,s(i)})))$ .  

$$(\lambda c_1^{(i)} \dots c_r^{(i)} c_{r+1}^{(i)} \dots c_{r+k(i)}^{(i)} K c_g^{(i)}(\hat{r}P_{i,1} \dots P_{i,s(i)})))$$
.  

$$(\lambda c_1^{(i)} \dots c_r^{(i)} c_{r+1}^{(i)} \dots c_{r+k(i)}^{(i)} K c_g^{(i)}(\hat{r}P_{i,1} \dots P_{i,s(i)})))$$
.  

$$(\lambda c_1^{(i)} \dots c_r^{(i)} c_{r+1}^{(i)} \dots c_{r+k(i)}^{(i)} K c_g^{(i)}(\hat{r}P_{i,1} \dots P_{i,s(i)})))$$
.

Let  $\tau_i = \forall \alpha_1.\rho_{i,1} \rightarrow \cdots \rightarrow \rho_{i,k(i)} \rightarrow \alpha_1.$ 

By induction we can construct a constant context for the set:

$$\{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n, \sigma_1, \dots, \sigma_n, \forall \alpha.\alpha\} \cup V$$

where  $n \geq 2$ .

Let 
$$D \equiv E(G, x: \alpha_1 \rightarrow \alpha_1, e: \alpha_1, y: \alpha_2 \rightarrow \alpha_2, f: \alpha_2, \hat{r}: \forall \alpha.\alpha, d_1: \nu_1, \dots, d_r: \nu_r).$$

It holds that D is a constant context for the set  $\{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n, \sigma_1, \dots, \sigma_p, \tau_1, \dots, \tau_q\}$ . The following type assumptions must occur in any typing of D:

$$\begin{array}{l} a_i: \ \mu_i \\ b_i: \ \mu_i[\alpha_1 := \alpha_2] \\ c_j^{(i)}: \ \varphi_i(j)[\alpha_1 := \beta] \ \text{where} \ \beta \notin \{\alpha_1, \dots, \alpha_n\} \\ s_i: \ \forall \alpha_1.\mu_1 {\rightarrow} \cdots {\rightarrow} \mu_r {\rightarrow} \rho_{i,1} {\rightarrow} \cdots {\rightarrow} \rho_{i,k(i)} {\rightarrow} \alpha_1 \\ t_i: \ \forall \alpha_1.\rho_{1,1} {\rightarrow} \cdots {\rightarrow} \rho_{i,k(i)} {\rightarrow} \alpha_1 \end{array} \ \ \text{(this is} \ \tau_i) \end{array}$$

# 3.5 Closures of All Open Types

**Lemma 3.4** There are constant contexts for every set C such that

$$C = \{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n, \sigma_1, \dots, \sigma_m\}$$

such that for all  $i, j \in \{1, ..., n\}$  it holds that  $\alpha_i = \alpha_j$  if and only if i = j and for all  $i \in \{1, ..., m\}$  it holds that  $\sigma_i = \forall \vec{\alpha}.\tau_i$  where  $BTV(\tau_i) = \emptyset$  and  $\vec{\alpha} = FTV(\tau_i) \subseteq \{\alpha_1, ..., \alpha_n\}$ .

**Proof:** As in lemma 3.3, the proof is by induction on a metric on types. We use the same metric. The construction is very similar to that for lemma 3.3, but we use the result of lemma 3.3 in proving it.

The base case is where for all  $i \in \{1, ..., m\}$  either  $|BTV(\sigma_i)| = 1$  or  $d(\sigma_i) = 0$ . This case is the consequence of lemma 3.3, since  $d(\sigma_i) = 0$  implies  $\sigma_i = \forall \alpha.\alpha$  which implies that  $|BTV(\sigma_i)| = 1$ .

Induction case: We assume the claim holds where for all  $i \in \{1, ..., m\}$  either  $|BTV(\sigma_i)| = 1$  or  $d(\sigma_i) \le k$ . We then prove it holds where for all  $i \in \{1, ..., m\}$  either  $|BTV(\sigma_i)| = 1$  or  $d(\sigma_i) \le k + 1$ .

Suppose we are trying to find a constant context for the set:

$$C = \{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n, \sigma_1, \dots, \sigma_p, \tau_1, \dots, \tau_q\}$$

where it holds that for all  $i \in \{1, ..., p\}$  either  $(|BTV(\sigma_i)| = 1 \text{ or } d(\sigma_i) \leq k)$  and that for all  $i \in \{1, ..., q\}$  it holds that  $d(\tau_i) = k + 1$ .

Let 
$$\tau_i = \forall \vec{\alpha}. \rho_{i,1} \rightarrow \cdots \rightarrow \rho_{i,k(i)} \rightarrow \alpha_{h(i)}$$
 where  $\alpha_{h(i)} \in \{\alpha_1, \dots, \alpha_n\}$ .  
Let  $R = \{\rho_{i,j} \mid 1 \leq i \leq q \text{ and } 1 \leq j \leq k(i)\}$ .

Let 
$$S = \{\pi \mid \exists \rho \in R. \ \pi \subseteq \rho\}$$
.  
Let  $T = \{\varphi \rightarrow \alpha_1 \mid \exists (\pi \rightarrow \varphi) \in S\}$ .  
Let  $U = S \cup T \cup \{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n\}$ .  
Enumerate  $U$  so that  $U = \{\mu_1, \dots, \mu_r\}$ .  
Let  $\nu_i = \begin{cases} \mu_i & \text{if } \mu_i \in \{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n\} \\ \forall \vec{\alpha}. \alpha_1 \rightarrow \dots \rightarrow \alpha_n \rightarrow \mu_i & \text{otherwise.} \end{cases}$   
Let  $V = \{\nu_1, \dots, \nu_r\}$ .  
Let  $g$  be a function s.t.  $\mu_{g(i)} = \alpha_{h(i)} = \nu_{g(i)}$ .  
Let  $\pi_i = \forall \alpha_1. \alpha_1 \rightarrow ((\mu_1 \rightarrow \dots \rightarrow \mu_r \rightarrow \rho_{i,1} \rightarrow \dots \rightarrow \rho_{i,k(i)} \rightarrow \alpha_1)[\alpha_2 := \alpha_1, \dots, \alpha_n := \alpha_1]) \rightarrow \alpha_1$ .  
Let  $\theta_i = \forall \alpha_1. \alpha_1 \rightarrow ((\rho_{i,1} \rightarrow \dots \rightarrow \rho_{i,k(i)} \rightarrow \alpha_1)[\alpha_2 := \alpha_1, \dots, \alpha_n := \alpha_1]) \rightarrow \alpha_1$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{i,j} = \mu_{f(i,j)} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{f(i,j)} = \mu_{f(i,j)} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{f(i,j)} = \mu_{f(i,j)} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{f(i,j)} = \mu_{f(i,j)} = \mu_{f(i,j)}$ .  
Let  $f$  be a function s.t.  $\rho_{f(i,j)} = \mu_{f(i,j)} = \mu_{f(i,j)} = \mu_{f(i,j)} = \mu_{f(i,j)} = \mu_{f(i$ 

By induction there is a constant context for the set:

$$\{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n, \sigma_1, \dots, \sigma_p, \forall \alpha.\alpha, \pi_1, \dots, \pi_q, \theta_1, \dots, \theta_q\} \cup V$$

where  $n \geq 2$ .

Let 
$$D \equiv E(G, x_1: \alpha_1 \rightarrow \alpha_1, \dots, x_n: \alpha_n \rightarrow \alpha_n, f_1: \alpha_1, \dots, f_n: \alpha_n, d_i: \nu_1, \dots, d_r: \nu_r, r: \forall \alpha.\alpha, m_1: \theta_1, \dots, m_q: \theta_q, e_1: \pi_1, \dots, e_q: \pi_q).$$

It holds that D is a constant context for the set C. In particular, the following type assumptions occur in every typing of D:

$$\begin{array}{l} a_i \ : \ \mu_i \\ c_{i,j} \ : \ \varphi_{i,j}[\alpha_1 := \beta_1, \ldots, \alpha_n := \beta_n] \ \text{where} \ \{\alpha_1, \ldots, \alpha_n\} \cap \{\beta_1, \ldots, \beta_n\} = \varnothing \\ s_i \ : \ \forall \vec{\alpha}.\mu_1 {\rightarrow} \cdots {\rightarrow} \mu_r {\rightarrow} \rho_{i,1} {\rightarrow} \cdots {\rightarrow} \rho_{i,k(i)} {\rightarrow} \alpha_{h(i)} \\ t_i \ : \ \forall \vec{\alpha}.\rho_{i,1} {\rightarrow} \cdots {\rightarrow} \rho_{i,k(i)} {\rightarrow} \alpha_{h(i)} \end{array}$$

## 3.6 Closures of All Types

**Lemma 3.5** There is a constant context for any set  $\{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n, \sigma_1, \dots, \sigma_t\}$  such that for all  $i, j \in \{1, \dots, n\}$  it holds that  $\alpha_i = \alpha_j$  if and only if i = j and for all  $i \in \{1, \dots, t\}$   $FTV(\sigma_i) = \emptyset$  and  $BTV(\sigma_i) \subseteq \{\alpha_1, \dots, \alpha_n\}$ .

**Proof:** By induction. Consider any set of types C such that:

$$C = \{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n, \sigma_1, \dots, \sigma_p, \tau_1, \dots, \tau_q\}$$

meeting the restriction stated above and such that:

- 1. For all  $i \in \{1, ..., p\}$  either  $d(\sigma_i) \leq k$  or  $\sigma_i = \forall \vec{\alpha}. \tau$  where  $BTV(\tau) = \emptyset$ .
- 2. For all  $i \in \{1, ..., q\}$  it holds that  $d(\tau_i) = k + 1$  and there are no  $\vec{\alpha}$  and  $\tau$  such that  $\tau_i = \forall \vec{\alpha}. \tau$  and BTV( $\tau$ ) =  $\emptyset$ .

The base case where k = 0 and q = 0 is a consequence of lemma 3.4. The first induction case will be to prove that if the claim holds for (k,q), then it also holds for (k,q+1). This will be the body of the proof. The second induction case is to prove that if the claim holds for (k,q) for all numbers q, then it also holds for (k+1,0). However, this is easy to see.

So for the first induction case, we suppose we have constant contexts for any set

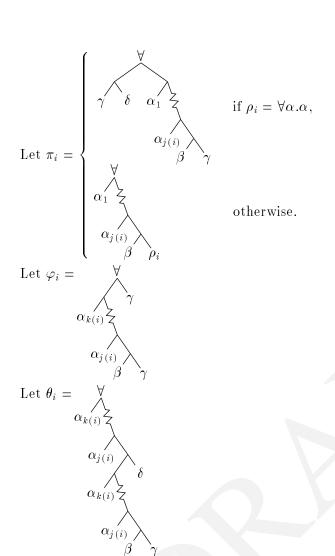
$$\{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n, \sigma_1, \dots, \sigma_n, \tau_1, \dots, \tau_n\}$$

meeting all of the restrictions stated above for k and q and then we prove that we have a constant context for the same set with the addition of the member  $\tau_{q+1}$  where  $d(\tau_{q+1}) = k+1$  and there are no  $\vec{\alpha}$  and  $\tau$  such that  $\tau_{q+1} = \forall \vec{\alpha}.\tau$  where  $\operatorname{BTV}(\tau) = \varnothing$  (i.e.  $\tau_{q+1}$  has quantifiers embedded below the root).

Let  $\tau_{q+1} = \forall \overrightarrow{\alpha_1}. \rho_1 \rightarrow \forall \overrightarrow{\alpha_2}. \rho_2 \rightarrow \cdots \forall \overrightarrow{\alpha_{s-1}}. \rho_{s-1} \rightarrow \forall \overrightarrow{\alpha_s}. \rho_s$  where  $\rho_s \in \{\alpha_1, \ldots, \alpha_n\}$ . Recall that we assume that no type variable is bound in more than one place. We know already that for all  $i \in \{1, \ldots, s\}$  it holds that  $\overrightarrow{\alpha_i} \cup \mathrm{FTV}(\rho_i) \cup \mathrm{BTV}(\rho_i) \subseteq \{\alpha_1, \ldots, \alpha_n\}$ .

Let k, j be functions such that k(i+1) = j(i) + 1 and  $k(i) \leq j(i) + 1$  and  $\overrightarrow{\alpha_i} = \{\alpha_{k(i)}, \dots, \alpha_{j(i)}\}$ . We renumber  $\alpha_1, \dots, \alpha_n$  as necessary so that we can define k and j.

Let 
$$V_i = \bigcup_{i \leq j \leq s} \overrightarrow{\alpha_j}$$
. Thus,  $V_1 = \{\alpha_1, \dots, \alpha_m\}$ . Note that  $m \leq n$ .



Let g be a value such that  $\rho_s = \alpha_g$ . Let  $h = g + 1 \mod n$ .

Let 
$$\mu = \begin{pmatrix} \forall \\ \epsilon \\ \delta \\ \gamma \\ \epsilon \end{pmatrix}$$
Let  $\eta = \begin{pmatrix} \forall \\ \gamma \\ \alpha \\ \delta \\ \beta \\ \alpha \end{pmatrix}$ 
Let  $\kappa = \begin{pmatrix} \forall \\ \alpha \\ \beta \\ \alpha \end{pmatrix}$ 

Let 
$$J_i \equiv \begin{cases} (t_i(\lambda w_i.r(xw_i)(yw_i))) & \text{if } \rho_i = \forall \alpha.\alpha, \\ t_i & \text{otherwise.} \end{cases}$$

$$\text{Let } M_i \equiv \begin{cases} (J_s c_1 \cdots c_m) & \text{if } i = s, \\ (p(J_i c_1 \cdots c_{j(i)}) N_{i+1}) & \text{for } 1 \leq i < s. \end{cases}$$

$$\text{Let } Q_i \equiv (m_i (J_i c_1 \cdots c_{j(i-1)} a_{k(i)} \cdots a_{j(i)} r) \cdots (J_{s-1} c_1 \cdots c_{j(i-1)} a_{k(i)} a_{j(s-1)} r)).$$

$$\text{Let } R_i \equiv Q_i [\vec{a} := \vec{b}].$$

$$\text{Let } S_i \equiv \begin{cases} (r(xQ_i)(yR_i)) & \text{if } \rho_s \in V_i, \\ (rQ_iR_i) & \text{otherwise.} \end{cases}$$

$$\text{Let } N_i \equiv ((\lambda h_i.k(j(g_ih_i)(\lambda m_i.S_i))(r(f_ia_{k(i)} \cdots a_{j(i)}h_i)(f_ib_{k(i)} \cdots b_{j(i)}h_i)))$$

$$(\lambda c_{k(i)} \cdots c_{i(i)}.\lambda z_i.M_i).$$

By induction, there is a constant context for the set:

$$\{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n, \forall \alpha.\alpha, \kappa, \mu, \eta, \vec{\pi}, \vec{\varphi}, \vec{\theta}, \sigma_1, \dots, \sigma_p, \tau_1, \dots, \tau_q\}$$

Let 
$$G = E(N_1, x: \alpha_g \to \alpha_g, y: \alpha_h \to \alpha_h, a_i: \alpha_1, \dots, a_n: \alpha_n, b_1: \alpha_2, \dots, b_{n-1}: \alpha_n, b_n: \alpha_1, r: \forall \alpha.\alpha, k: \kappa, j: \mu, p: \eta, t_1: \pi_1, \dots, t_s: \pi_s, g_1: \varphi_1, \dots, g_s: \varphi_s, f_1: \theta_1, \dots, f_s: \theta_s).$$

It holds that G is a constant context for the set:

$$\{\alpha_1 \rightarrow \alpha_1, \alpha_1, \dots, \alpha_n \rightarrow \alpha_n, \alpha_n, \sigma_1, \dots, \sigma_p, \tau_1, \dots, \tau_q, \tau_{q+1}\}$$

In particular, the following type assumptions and derivations must occur in any typing of G:

$$\begin{split} &m_{i} \,:\, (\forall \overrightarrow{\alpha_{i}}.\rho_{i} \rightarrow \cdots \forall \overrightarrow{\alpha_{s-1}}.\rho_{s-1} \rightarrow \forall \overrightarrow{\alpha_{s}}.\rho_{s})[\overrightarrow{\alpha}:=\overrightarrow{\beta}] \\ &c_{i} \,:\, \beta_{i} \text{ where } \forall j.\beta_{i} \neq \alpha_{j} \text{ and } (\beta_{i} = \beta_{j} \Leftrightarrow i = j) \\ &w_{i} \,:\, \forall \alpha.\alpha \\ &h_{i} \,:\, (\forall \overrightarrow{\alpha_{i}}.\alpha_{k(i)} \rightarrow \cdots \rightarrow \alpha_{j(i)} \rightarrow \Box \rightarrow \rho_{i} \rightarrow \forall \overrightarrow{\alpha_{i+1}}.\rho_{i+1} \rightarrow \cdots \rightarrow \forall \overrightarrow{\alpha_{s-1}}.\rho_{s} \rightarrow \forall \overrightarrow{\alpha_{s}}.\rho_{s})[\overrightarrow{\alpha}:=\overrightarrow{\beta}] \\ &M_{i} \,:\, (\rho_{i} \rightarrow \cdots \forall \overrightarrow{\alpha_{s-1}}.\rho_{s-1} \rightarrow \forall \overrightarrow{\alpha_{s}}.\rho_{s})[\overrightarrow{\alpha}:=\overrightarrow{\beta}] \\ &N_{i} \,:\, (\Box \rightarrow \forall \overrightarrow{\alpha_{i}}.\rho_{i} \rightarrow \cdots \forall \overrightarrow{\alpha_{s-1}}.\rho_{s-1} \rightarrow \forall \overrightarrow{\alpha_{s}}.\rho_{s})[\overrightarrow{\alpha}:=\overrightarrow{\beta}] \end{split}$$

## 3.7 All Types

**Lemma 3.6** There is a constant context for any set of types  $\{\sigma_1, \ldots, \sigma_p\}$ .

**Proof:** We define  $\alpha_1, \ldots, \alpha_n, \alpha_{n+1}, \ldots, \alpha_m$  so that it holds that:

$$\bigcup_{1 \le i \le n} \mathrm{FTV}(\sigma_i) = \{\alpha_1, \dots, \alpha_n\}$$

$$\bigcup_{1 \le i \le n} \operatorname{BTV}(\sigma_i) = \{\alpha_{n+1}, \dots, \alpha_m\}$$

For each  $i \in \{1, \ldots, p\}$ , let  $\tau_i$  be the type  $\forall .\alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow \beta \rightarrow \sigma_i$ . By lemma 3.5, there is a constant context for the set  $\{\alpha_1 \rightarrow \alpha_1, \alpha_1, \ldots, \alpha_m \rightarrow \alpha_m, \alpha_m, \beta \rightarrow \beta, \beta, \forall \alpha.\alpha, \tau_1, \ldots, \tau_p, \pi\}$  where  $\pi = \forall .(\alpha_1 \rightarrow \alpha_2) \rightarrow (\alpha_3 \rightarrow \alpha_1) \rightarrow \alpha_4$ . (For convenience, we assume  $m \geq 5$ ).

Let 
$$N_i \equiv \begin{cases} r & \text{if } i = 0, \\ (p(\lambda m_i.N_{i-1})(t_i a_1 \dots a_n)) & \text{otherwise.} \end{cases}$$
  
Let  $J \equiv E(N_n, a_1; \alpha_1, \dots, a_n; \alpha_n, t_i; \tau_1, \dots, t_n; \tau_n, r; \forall \alpha, \alpha, p; \pi).$ 

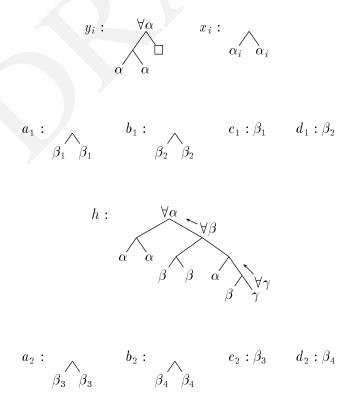
It holds that J is a constant context for the desired set  $\{\sigma_1, \ldots, \sigma_p\}$ .

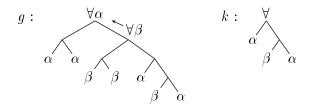
# 3.8 In the $\lambda I$ Calculus

The entire construction can be repeated for the  $\lambda I$  calculus

Let 
$$C_i[\cdot] \equiv ((\lambda y_i.r(y_iy_i)(y_i(\lambda w_iw_i))(y_i(\lambda z_i.\lambda v_i.z_iv_i)))(\lambda x_i.r(x_i(x_ir))[\cdot])).$$
  
Let  $H[\cdot] \equiv ((\lambda h.r(x_3(hx_1x_2(x_1r)(x_2r)))(x_1(hx_2x_3(x_2r)(x_3r)))[\cdot])(\lambda a_1b_1c_1d_1r(a_1(a_1c_1))(b_1(b_1d_1)))).$   
Let  $G[\cdot] \equiv ((\lambda g.r(x_1(gx_1x_2(x_1r)(x_2r)))(x_2(gx_2x_1(x_2r)(x_1r)))[\cdot])(\lambda a_2b_2c_2d_2.a_2(ha_2b_2c_2d_2))).$   
Let  $L \equiv ((\lambda k.r(x_1(k(x_1r)(x_2r)))(x_2(k(x_2r)(x_1r))))(grr)).$   
Let  $J \equiv (\lambda r.C_1[C_2[C_3[H[G[L]]]]).$ 

As can easily be determined by inspection, J is a term of the  $\lambda I$  calculus. Moreover, the following type assumptions must occur in every typing of J in System F:





The important thing to note is that the type assigned to the object variable k allows k to be used to simulate the K combinator. We can simply repeat the entire construction from this point using k wherever we used K before.

# 4 Reduction from Type Checking to Typability

**Theorem 4.1** Type checking in System  $\mathbf{F}$  is reducible to typability in System  $\mathbf{F}$ . Typability in System  $\mathbf{F}$  is therefore undecidable.

**Proof:** Consider an instance of the type-checking problem in which we are asked whether the assertion  $A \vdash M : \tau$  can be derived in System **F**. Let  $A = \{y_1 : \sigma_1, \ldots, y_n : \sigma_n\}$ . By lemma 3.6, we know there is a constant context for the set of types  $\{\sigma_1, \ldots, \sigma_n, \tau \rightarrow \alpha\}$  where  $\alpha$  is a fresh type variable. Let N be the  $\lambda$ -term (zM) where z is a fresh object variable. Let P be the following  $\lambda$ -term:

$$P \equiv E(N, y_1:\sigma_1, \ldots, y_n:\sigma_n, z:\tau \rightarrow \alpha)$$

It is the case that P is typable in System  $\mathbf{F}$  if and only if there is a derivation in System  $\mathbf{F}$  that ends with the assertion  $A \vdash M : \tau$ . Thus, type checking reduces to typability. Typability is therefore undecidable since by theorem 2.1 type checking is undecidable.

It is worth observing that since it has been known that typability easily reduces to type checking, the two problems in System  $\mathbf{F}$  are of equivalent difficulty. The reduction from typability to type checking is presented in [3].

# References

- [1] Henk Barendregt. Lambda calculi with types. Chapter to appear in forthcoming book. Technical Report 91-19, Department of Computer Science, Faculty of Mathematics and Computer Science, Catholic University Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, September 1991. E-mail: henk@cs.kun.nl.
- [2] Assaf J. Kfoury, Jerzy Tiuryn, and Pawel Urzyczyn. The undecidability of the semi-unification problem. *Information and Computation*, 102(1):83–101, 1993. Preliminary version in *Proc. STOC 1990*.

[3] Pawel Urzyczyn. Type reconstruction in  $\mathbf{F}_{\pmb{\omega}}.$  E-mail: urzy@mimuw.edu.pl, June 1993.