TYPABILITY AND TYPE INFERENCE IN ATOMIC POLYMORPHISM

M. CLARENCE PROTIN [a] AND GILDA FERREIRA [b]

[a] Rua Defensores da Liberdade 10 2° Esq., 7050-230 Montemor-o-Novo, Portugal
e-mail address: cprotin@sapo.pt

[b] Universidade Aberta, 1269-001 Lisboa, Portugal; Centro de Matemática, Aplicações Fundamentais
e Investigação Operacional — Faculdade de Ciências da Universidade de Lisboa, 1749-016 Lisboa,
Portugal
e-mail address: gmferreira@fc.ul.pt

ABSTRACT. It is well-known that typability, type inhabitation and type inference are
undecidable in the Girard-Reynolds polymorphic system **F**. It has recently been proven
that type inhabitation remains undecidable even in the predicative fragment of system **F**
in which all universal instantiations have an atomic witness (system **F$_{at}$**). In this paper we
analyze typability and type inference in Curry style variants of system **F$_{at}$** and show that
typability is decidable and that there is an algorithm for type inference which is capable of
dealing with non-redundancy constraints.

INTRODUCTION

Type inhabitation, typability and type inference in Girard-Reynolds polymorphic system **F**
(also known as the second-order polymorphically typed lambda calculus) [Gir71, Rey74,
GLT89] are known to be undecidable [Lö76, DR19, Wel99].

Type inhabitation is the following problem: given a type A, is there a term having that
type? I.e. is there a term M such that $\vdash M : A$? Via the Curry-Howard isomorphism it
corresponds to asking if a formula is provable.

Typability goes in the other way around:

(Typ) Given a term M is there a type A and a type environment Γ such that $\Gamma \vdash M : A$?

Via the Curry-Howard isomorphism typability corresponds to asking if a construction is
a proof of a formula, i.e. is there A such that M is the proof of A?

What about being able to find the actual types (and type environments) which satisfy the condition above if the answer is affirmative? In this paper by the problem of *type inference* we mean the following:

(Ti) Given a term M such that (Typ) admits an affirmative answer find a procedure to generate types A and type environments Γ such that $\Gamma \vdash M : A$.

Having an effective procedure for (Typ) the formulation above is readily seen to be equivalent to the standard formulation.

It has recently been proven [Pro21] that type inhabitation remains undecidable even in a very weak fragment of system $\mathbf{F}$ (known as system $\mathbf{F_{at}}$ or atomic polymorphism [Fer06, FF13]) where only atomic instantiations are allowed.

What about typability and type inference for $\mathbf{F_{at}}$? Is typability decidable? Can we obtain an algorithm for type inference? In this paper we show that typability is decidable and we give a procedure for type inference.

In analyzing the typing properties of $\mathbf{F_{at}}$, the system is considered not in its original Church style formulation (in which case they are trivial) but in Curry style variants where terms do not come with full type annotation.

The paper is structured as follows. In the next section we recall the system involved in the present study: system $\mathbf{F_{at}}$.

In Section 2 we show that typability is decidable for $\mathbf{F_{at}}$ in *Curry style* (according to the terminology of [Wel99][p.122] for $\mathbf{F}$). In Section 3 we show that typability for $\mathbf{F_{at}}$ in the *Polymorphic Curry style* (in which terms contain additional polymorphic typing information) is also decidable, the proof also furnishing a procedure for type inference restricted to *redundant* typings. In Section 4 we analyze non-redundant typability, show how we can check for such typings and generate them if they exist. In Section 5 it is remarked that the results of Section 4 yield as a particular case a method for type inference for Polymorphic Curry style $\mathbf{F_{at}}$. This section also contains various other remarks and considerations about future work.

1. System $\mathbf{F_{at}}$

The atomic polymorphic system $\mathbf{F_{at}}$ [FF13] is the fragment of Girard/Reynolds system $\mathbf{F}$ [Gir71, Rey74] induced by restricting to atomic instances the elimination inference rule for $\forall$, and the corresponding proof term constructor. System $\mathbf{F_{at}}$ was originally proposed as a natural and appealing framework for full intuitionistic propositional calculus (**IPC**) [Fer06, SF20]. $\mathbf{F_{at}}$ expresses the connectives of **IPC** in a uniform way avoiding *bad connectives* (according to Girard [GLT89], page 74) and avoiding commuting conversions. This explains the usefulness of the system in proof theoretical studies [Fer17a, Fer17b]. For related work in the topic see [PTP22], where the authors investigate predicative translations of **IPC** into system $\mathbf{F_{at}}$ using the equational framework presented in [TPP19] and giving an elegant semantic explanation (relying on parametricity) of the syntactic results on atomic polymorphism. Originally system $\mathbf{F}$ and system $\mathbf{F_{at}}$ were presented in *Church style* where types are embedded in terms. The questions of typability and type inference, mentioned in the Introduction, are meaningful only when we are in (variants of) *Curry style*, i.e., terms

are untyped and the type information is kept apart. Each term has a set (which may be empty) of possible types. In what follows we present $\mathbf{F_{at}}$ in *Polymorphic Curry style*.

The types/formulas in $\mathbf{F_{at}}$ are exactly the ones of system $\mathbf{F}$:

$$A, B ::= X \mid A \to B \mid \forall X.A$$

The terms in $\mathbf{F_{at}}$ are given by

$$M, N ::= x \mid MN \mid \lambda x.M \mid MX \mid \Lambda X.M$$

having the following typing/inference rules

$$\text{(VAR)} \ \frac{}{\Gamma \vdash x : \Gamma(x)} \ x \in dom(\Gamma) \qquad \text{(ABS)} \ \frac{\Gamma \cup \{x : A\} \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \qquad \text{(APP)} \ \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

$$\text{(GEN}_X) \ \frac{\Gamma \vdash M : A}{\Gamma \vdash \Lambda X.M : \forall X.A} \ X \notin FTV(\Gamma) \qquad \text{(INST)} \ \frac{\Gamma \vdash M : \forall X.A}{\Gamma \vdash MY : A[Y/X]}$$

where a type environment Γ is a finite set $x_1 : A_1, \ldots, x_n : A_n$ where x_i are assumption (term) variables and A_i are types. We denote the set $\{x_1, \ldots, x_n\}$ by $dom(\Gamma)$, the set $\{A_1, \ldots, A_n\}$ by $ran(\Gamma)$ and write $\Gamma(x) = A$ whenever $x : A \in \Gamma$. We denote the free variables in the types in $ran(\Gamma)$ by $FTV(\Gamma)$.

Note that, as stressed in the beginning of this section, the difference between system $\mathbf{F_{at}}$ and system $\mathbf{F}$ lies in the restriction of the universal instantiation rule to type variables.

We can drop some of the information in the terms and present $\mathbf{F_{at}}$ alternatively in *Curry style* (see [Wel99], page 122 in the context of system $\mathbf{F}$) which differs from the *Polymorphic Curry style* in the terms allowed

$$M, N ::= x \mid MN \mid \lambda x.M$$

having the following type inference rules

$$\text{(VAR)} \ \frac{}{\Gamma \vdash x : \Gamma(x)} \ x \in dom(\Gamma) \qquad \text{(ABS)} \ \frac{\Gamma \cup \{x : A\} \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \qquad \text{(APP)} \ \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

$$\text{(GEN}_X) \ \frac{\Gamma \vdash M : A}{\Gamma \vdash M : \forall X.A} \ X \notin FTV(\Gamma) \qquad \text{(INST)} \ \frac{\Gamma \vdash M : \forall X.A}{\Gamma \vdash M : A[Y/X]}$$

In what follows, we use $A\{Y/X\}$ to denote a type that results from A by replacing some (possibly all) free occurrences of X by Y. Observe that if we have $\Gamma \vdash M : A$ in *Curry style* derived using the (INST) rule then we have $\Gamma \vdash M : \forall X.A\{X/Y\}$ for some type variables X and Y. Likewise if we have $\Gamma \vdash MY : A$ in *Polymorphic Curry style* then $\Gamma \vdash M : \forall X.A\{X/Y\}$ for some type variable X.

2. Typability for the Curry system

In this Section we show that the typability problem for $\mathbf{F_{at}}$, considering the system in *Curry style*, is decidable. The strategy will be to reduce the typability problem to the corresponding problem in the simply typed lambda-calculus $\lambda^\to$ which is known to be decidable. To do this, although not strictly necessary, we will employ a theorem which holds more generally for Polymorphic Curry style terms.

Fix a type variable $\circ$. Given a type A of $\mathbf{F_{at}}$ we define $\langle A \rangle$ as the type of $\lambda^{\rightarrow}$ given by $\langle X \rangle := \circ$, $\langle B \rightarrow C \rangle := \langle B \rangle \rightarrow \langle C \rangle$ and $\langle \forall X.A \rangle := \langle A \rangle$. Let M be a term of $\mathbf{F_{at}}$ in *Polymorphic Curry style* (or in *Curry style*). We define $[M]$ as the term of $\lambda^{\rightarrow}$ given by $[x] := x$, $[MN] := [M][N]$, $[\lambda x.M] := \lambda x.[M]$, $[MX] := [M]$, $[\Lambda X.M] := [M]$. Finally for a type environment Γ for $\mathbf{F_{at}}$ we define the environment $\langle \Gamma \rangle$ such that $dom(\langle \Gamma \rangle) := dom(\Gamma)$ and $\langle \Gamma \rangle(x) := \langle \Gamma(x) \rangle$.

The following lemma is easily proved by induction on the structure of the type.

Lemma 2.1. *Let A be an arbitrary type and X and Y be type variables. Then*

$$\langle \forall X.A \rangle \; = \; \langle A\{Y/X\} \rangle \; = \; \langle A \rangle.$$

Theorem 2.2. *Given a term M in* Polymorphic Curry style, *a type environment Γ and a type A in $\mathbf{F_{at}}$,*

$$\text{if } \Gamma \vdash_{\mathbf{F_{at}}} M : A \text{ then } \langle \Gamma \rangle \; \vdash_{\lambda^{\rightarrow}} [M] : \; \langle A \rangle.$$

Proof. The proof is done by induction on the complexity of M. Let M be a variable x. Then it is typed with (VAR) $\Gamma \vdash_{\mathbf{F_{at}}} x : \Gamma(x)$ and obviously $\langle \Gamma \rangle \vdash_{\lambda^{\rightarrow}} x : \langle \Gamma \rangle(x) = \langle \Gamma(x) \rangle$. Consider an application $\Gamma \vdash_{\mathbf{F_{at}}} MN : A$. Then $\Gamma \vdash_{\mathbf{F_{at}}} M : B \rightarrow A$ and $\Gamma \vdash_{\mathbf{F_{at}}} N : B$ for some B. By induction hypothesis $\langle \Gamma \rangle \vdash_{\lambda^{\rightarrow}} [M] : \langle B \rangle \rightarrow \langle A \rangle$ and $\langle \Gamma \rangle \vdash_{\lambda^{\rightarrow}} [N] : \langle B \rangle$ hence $\langle \Gamma \rangle \vdash_{\lambda^{\rightarrow}} [M][N] = [MN] : \langle A \rangle$. If we have an abstraction $\Gamma \vdash_{\mathbf{F_{at}}} \lambda x.M : A \rightarrow B$ then $\Gamma \cup \{x : A\} \vdash_{\mathbf{F_{at}}} M : B$. By induction hypothesis $\langle \Gamma \cup \{x : A\} \rangle \vdash_{\lambda^{\rightarrow}} [M] : \langle B \rangle$ hence

$$\langle \Gamma \rangle \cup \; \{x : \langle A \rangle\} \vdash_{\lambda^{\rightarrow}} [M] : \langle B \rangle$$

Using (ABS) we get

$$\langle \Gamma \rangle \; \vdash_{\lambda^{\rightarrow}} \lambda x.[M] : \langle A \rangle \; \rightarrow \; \langle B \rangle$$

Which by definition of $[\cdot]$ yields

$$\langle \Gamma \rangle \; \vdash_{\lambda^{\rightarrow}} [\lambda x.M] : \langle A \rightarrow B \rangle$$

In the case of universal application we have $\Gamma \vdash_{\mathbf{F_{at}}} MX : A$. But then $\Gamma \vdash_{\mathbf{F_{at}}} M : \forall Y.A\{Y/X\}$. By induction hypothesis $\langle \Gamma \rangle \; \vdash_{\lambda^{\rightarrow}} [M] : \langle \forall Y.A\{Y/X\} \rangle \overset{\text{Lemma 2.1}}{=} \langle A \rangle$ and since $[MX] = [M]$ we get

$$\langle \Gamma \rangle \; \vdash_{\lambda^{\rightarrow}} [MX] : \langle A \rangle.$$

Finally the case of universal abstraction, $\Gamma \vdash_{\mathbf{F_{at}}} \Lambda X.M : \forall X.A$ follows from $[\Lambda X.M] = [M]$ and $\langle \forall X.A \rangle \; = \; \langle A \rangle$ and the induction hypothesis for $\Gamma \vdash_{\mathbf{F_{at}}} M : A$. $\square$

Corollary 2.3. *If M is Curry typable then M is $\lambda^{\rightarrow}$ typable.*

Proof. Immediate by Theorem 2.2, noticing that being M a term of $\mathbf{F_{at}}$ in *Curry style*, $[M] = M$. $\square$

Note that, given a $\lambda^{\rightarrow}$-term, the inverse implication is also valid, not only for the Curry system but for the Polymorphic Curry system.

Corollary 2.4. *The typability problem is decidable in Curry system $\mathbf{F_{at}}$.*

Proof. From the previous note and Corollary 2.3 we have that Curry typability is equivalent to $\lambda^{\rightarrow}$ typability and the latter is known to be decidable. $\square$

3. Typability for Polymorphic Curry terms

In this section we address the more complex question of typability of $\mathbf{F_{at}}$ presented in *Polymorphic Curry style*.

Let $\circ$ and $\bullet$ be two fixed different type variables in $\mathbf{F_{at}}$.

We define the subsystem $\mathbf{F_{at}^{\circ\bullet}}$ of $\mathbf{F_{at}}$ whoses types are given by

$$A, B := \circ \mid A \to B \mid \forall \bullet .A$$

and terms by

$$M, N := x \mid MN \mid \lambda x.M \mid M \bullet \mid \Lambda \bullet .M$$

The type inference rules for $\mathbf{F_{at}^{\circ\bullet}}$ are the same as the ones for $\mathbf{F_{at}}$ except for the universal inference rules that are given by

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \Lambda \bullet .M : \forall \bullet .A} \ (\text{GEN}_\bullet) \qquad \frac{\Gamma \vdash M : \forall \bullet .A}{\Gamma \vdash M\bullet : A} \ (\text{INST})$$

Given a type A of $\mathbf{F_{at}}$ we define $[A]$ as the type of $\mathbf{F_{at}^{\circ\bullet}}$ given by $[X] = \circ$, $[A \to B] = [A] \to [B]$ and $[\forall X.A] = \forall \bullet .[A]$. For a type environment Γ for $\mathbf{F_{at}}$ we define the type environment $[\Gamma]$ for $\mathbf{F_{at}^{\circ\bullet}}$ such that $dom([\Gamma]) = dom(\Gamma)$ and $[\Gamma](x) = [\Gamma(x)]$.

We define a map from the terms of $\mathbf{F_{at}}$ to the terms of $\mathbf{F_{at}^{\circ\bullet}}$ given by $|x| = x$, $|MN| = |M||N|$, $|\lambda x.M| = \lambda x.|M|$, $|MX| = |M|\bullet$ and $|\Lambda X.M| = \Lambda \bullet .|M|$.

Analogously to Lemma 2.1 we have the following result:

Lemma 3.1. *Let A be an arbitrary type in $\mathbf{F_{at}}$ and X and Y be type variables. Then* $[A\{Y/X\}] = [A]$.

3.1. Reduction to $\mathbf{F_{at}^{\circ\bullet}}$.

Proposition 3.2. *Let M be a term in $\mathbf{F_{at}}$. If there are a type environment Γ and a type A in $\mathbf{F_{at}}$ such that $\Gamma \vdash_{\mathbf{F_{at}}} M : A$ then $[\Gamma] \vdash_{\mathbf{F_{at}^{\circ\bullet}}} |M| : [A]$.*

Proof. The proof is done by induction on the structure of M. Let M be a variable x. Then it is typed with (VAR) $\Gamma \vdash_{\mathbf{F_{at}}} x : \Gamma(x)$ and, since $|x| = x$, obviously $[\Gamma] \vdash_{\mathbf{F_{at}^{\circ\bullet}}} |x| : [\Gamma](x) = [\Gamma(x)]$.

Consider an application $\Gamma \vdash_{\mathbf{F_{at}}} MN : A$. Then $\Gamma \vdash_{\mathbf{F_{at}}} M : B \to A$ and $\Gamma \vdash_{\mathbf{F_{at}}} N : B$ for some type B. By induction hypothesis $[\Gamma] \vdash_{\mathbf{F_{at}^{\circ\bullet}}} |M| : [B \to A] = [B] \to [A]$ and $[\Gamma] \vdash_{\mathbf{F_{at}^{\circ\bullet}}} |N| : [B]$ hence, since $|MN| = |M||N|$, we have $[\Gamma] \vdash_{\mathbf{F_{at}^{\circ\bullet}}} |MN| : [A]$.

If we have an abstraction $\Gamma \vdash_{\mathbf{F_{at}}} \lambda x.M : A \to B$ then $\Gamma \cup \{x : A\} \vdash_{\mathbf{F_{at}}} M : B$. By induction hypothesis $[\Gamma \cup \{x : A\}] \vdash_{\mathbf{F_{at}^{\circ\bullet}}} |M| : [B]$ hence

$$[\Gamma] \cup \{x : [A]\} \vdash_{\mathbf{F_{at}^{\circ\bullet}}} |M| : [B]$$

$$[\Gamma] \vdash_{\mathbf{F_{at}^{\circ\bullet}}} \lambda x.|M| : [A] \to [B]$$

$$[\Gamma] \vdash_{\mathbf{F_{at}^{\circ\bullet}}} |\lambda x.M| : [A \to B]$$

In the case of INST we have $\Gamma \vdash_{\mathbf{F_{at}}} MX : A$. But then $\Gamma \vdash_{\mathbf{F_{at}}} M : \forall Y.A\{Y/X\}$. By induction hypothesis $[\Gamma] \vdash_{\mathbf{F_{at}^{\circ\bullet}}} |M| : [\forall Y.A\{Y/X\}] = \forall \bullet .[A\{Y/X\}] \overset{\text{Lemma 3.1}}{=} \forall \bullet .[A]$ and applying $|M|\bullet = |MX|$ we get

$$[\Gamma] \vdash_{\mathbf{F_{at}^{\circ\bullet}}} |MX| : [A].$$

Finally consider the case of (GEN_X), $\Gamma \vdash_{\mathbf{F_{at}}} \Lambda X.M : A = \forall X.B$. We have $\Gamma \vdash_{\mathbf{F_{at}}} M : B$ with X not belonging to the free type variables in Γ. By induction $[\Gamma] \vdash_{\mathbf{F_{at}^{\circ\bullet}}} |M| : [B]$. Then since $\bullet$ does not belong to the free type variables in $[\Gamma]$, using $(\text{GEN}_\bullet)$ we get $[\Gamma] \vdash_{\mathbf{F_{at}^{\circ\bullet}}} \Lambda \bullet .|M| : \forall \bullet .[B] = [\forall X.B]$. Since $\Lambda \bullet .|M| = |\Lambda X.M|$, the result follows. $\qquad\square$

Note that we work modulo α-equivalence for type variables, in particular we assume the name of the bound variables is always appropriately chosen. Since in $\mathbf{F_{at}^{\circ\bullet}}$, the type variable $\bullet$ does not occur in the scope of any quantifier it is clear that if we substitute occurrences of $\bullet$ in a type A of $\mathbf{F_{at}^{\circ\bullet}}$ by any type variables other $\circ$, then we obtain a α-equivalent type A' in $\mathbf{F_{at}}$.

Lemma 3.3. *Assume that $\Gamma \vdash_{\mathbf{F_{at}^{\circ\bullet}}} M : A$. Let M' be the term of $\mathbf{F_{at}}$ which results from substituting some occurrences of subterms of the form $\Lambda\bullet.N$ in M by $\Lambda X.N$ with X some type variable different from $\circ$ and $N\bullet$ in M by NY with Y any type variable (different occurrences of the subterms may be replaced using different type variables). Then $\Gamma \vdash_{\mathbf{F_{at}}} M' : A'$ where A' is α-equivalent to A.*

Proof. The proof is done by induction on the structure of M.

If M is an assumption variable x and $\Gamma \vdash_{\mathbf{F_{at}^{\circ\bullet}}} x : A$ then $x' = x$ and the result follows immediately since $\Gamma \vdash_{\mathbf{F_{at}^{\circ\bullet}}} x : A$ implies $\Gamma \vdash_{\mathbf{F_{at}}} x : A$.

If M is NS and $\Gamma \vdash_{\mathbf{F_{at}^{\circ\bullet}}} NS : A$ then it comes from (APP).

Thus we have $\Gamma \vdash_{\mathbf{F_{at}^{\circ\bullet}}} N : B \to A$ and $\Gamma \vdash_{\mathbf{F_{at}^{\circ\bullet}}} S : B$ for some type B.

Let us write $M' = (NS)' = N'S'$ where N' and S' are the result of applying restrictions of the substitution defining M'. Then by induction hypothesis $\Gamma \vdash_{\mathbf{F_{at}}} N' : (B \to A)'$ and $\Gamma \vdash_{\mathbf{F_{at}}} S' : B'$ for some type B' α-equivalent to B and some type $(B \to A)'$ α-equivalent to $B \to A$. Thus $(B \to A)'$ is of the form $C \to D$ where C is α-equivalent to B, and thus α- equivalent to B', and D is α-equivalent to A. Applying (APP) we obtain $\Gamma \vdash_{\mathbf{F_{at}}} N'S' = (NS)' : D$ with D α-equivalent to A.

If M is $\lambda x.N$ then $\Gamma \vdash_{\mathbf{F_{at}^{\circ\bullet}}} \lambda x.N : A \to B$ which comes from (ABS) and we have $\Gamma \cup \{x : A\} \vdash_{\mathbf{F_{at}^{\circ\bullet}}} N : B$. Let us write $M' = \lambda x.N'$. Then by induction hypothesis $\Gamma \cup \{x : A\} \vdash_{\mathbf{F_{at}}} N' : B'$ with B' α-equivalent to B. But using (ABS) we obtain $\Gamma \vdash_{\mathbf{F_{at}}} \lambda x.N' : A \to B'$. Then the result follows since $M' = \lambda x.N'$ and $A \to B'$ is α-equivalent to $A \to B$.

If M is $N\bullet$ then $\Gamma \vdash_{\mathbf{F_{at}^{\circ\bullet}}} N\bullet : A$ comes from (INST) and we have $\Gamma \vdash_{\mathbf{F_{at}^{\circ\bullet}}} N : \forall \bullet .A$.

Let $M' = N'X$ (where X may be $\bullet$). By induction hypothesis $\Gamma \vdash_{\mathbf{F_{at}}} N' : (\forall \bullet .A)'$ where $(\forall \bullet .A)'$ is α-equivalent to $\forall \bullet .A$. If we write $(\forall \bullet .A)' = \forall Y.B$ (where Y can be $\bullet$) then it is clear that B must be α-equivalent to A. Applying (INST) for X yields $\Gamma \vdash_{\mathbf{F_{at}}} N'X : B$ since Y is not free in B. The result then follows, since $M' = N'X$ and B is α-equivalent to A.

If M is $\Lambda \bullet .N$ then $\Gamma \vdash_{\mathbf{F_{at}^{\circ\bullet}}} \Lambda \bullet .N : \forall \bullet .A$ comes from $(\text{GEN}_\bullet)$ and we have $\Gamma \vdash_{\mathbf{F_{at}^{\circ\bullet}}} N : A$. Let us write $M' = \Lambda X.N'$ where X can be $\bullet$ but not $\circ$. By induction hypothesis $\Gamma \vdash_{\mathbf{F_{at}}} N' : A'$ with A' α-equivalent to A. Applying (GEN_X) (since the proviso is trivially satisfied) yields $\Gamma \vdash_{\mathbf{F_{at}}} \Lambda X.N' : \forall X.A'$. The result then follows since $\forall X.A'$ is α-equivalent to $\forall \bullet .A$. $\qquad\square$

Corollary 3.4. *Given a Polymorphic Curry-term M, if $|M|$ is typable in $\mathbf{F_{at}^{\circ\bullet}}$ then M is typable in $\mathbf{F_{at}}$.*

Proof. Let M be a term in $\mathbf{F_{at}}$ which does not contain a subterm of the form $\Lambda \circ .N$. Then M results from $|M|$ by a substitution process as in Lemma 3.3 and the result follows. Let

M be a term in $\mathbf{F_{at}}$ which contains a subterm of the form $\Lambda \circ .N$. Note that the typability of a term in $\mathbf{F_{at}}$ does not change if we rename all the type variables appearing in the term. Thus renaming $\circ$ to a fresh type variable the result follows as above. $\qquad\square$

Hence combining Proposition 3.2 and Corollary 3.4 we get

Theorem 3.5. *A Polymorphic Curry-term M is typable in $\mathbf{F_{at}}$ if and only if $|M|$ is typable in $\mathbf{F_{at}^{\circ\bullet}}$.*

3.2. Typability in $\mathbf{F_{at}^{\circ\bullet}}$. In what follows we work in $\mathbf{F_{at}^{\circ\bullet}}$.

We define the set $\mathbb{S}$ of *type schemes* by

$$\sigma, \tau := \alpha \,|\, \sigma \to \tau \,|\, \forall \bullet .\sigma$$

where α is a type scheme variable.

A *generalised environment* is a finite set of elements of the form $x : \sigma$, where x is an assumption variable and σ is a type scheme. Unlike in the usual environments x may occur more than once.

By a *multisystem* we mean a finite set[1] $\{e_1, \ldots, e_k\}$ such that each e_i is a multiequation[2] $a_1 = \cdots = a_j$ with the a_i in $\mathbb{S}$. Given a generalised environment D we get a multisystem $\tilde{D}$ in the following way: let $D[x]$ be the set of all elements σ_i in $\mathbb{S}$ such that $x : \sigma_i \in D$. Then we form the multiequation $\sigma_1 = \cdots = \sigma_k$ where the σ_i are all the elements in $D[x]$. The multisystem $\tilde{D}$ is defined by taking the union of such multiequations for all x in the domain of D.

Let M be a term of $\mathbf{F_{at}^{\circ\bullet}}$. Assume w.l.o.g. that M is in *Barendregt form*, that is, the names of all the bound assumption variables are chosen to be distinct between themselves as well as distinct from the free assumption variables[3]. A *tagging* $\mathcal{A}$ for M is a map which assigns subterms and bound assumption variables[4] of M to type scheme variables in such a way that distinct subterms/bound variables are assigned distinct type scheme variables. We denote the image of the tagging of a subterm/bound variable N by $\mathcal{A}(N)$. Here we are using strict (typographical) equality of subterms.

Definition 3.6. Let M be a term of $\mathbf{F_{at}^{\circ\bullet}}$ and $\mathcal{A}$ a tagging for M. The *associated multisystem* of M is the set $\{e_N\}_{N \in \mathcal{T}}$ where $\mathcal{T}$ is the set of subterms of M which are not assumption variables and e_N are multiequations defined as follows:
- if $N := SP$, $\mathcal{A}(N) = \alpha_1$, $\mathcal{A}(S) = \alpha_2$ and $\mathcal{A}(P) = \alpha_3$ then e_N is $\alpha_2 = \alpha_3 \to \alpha_1$
- if $N := \lambda x.S$, $\mathcal{A}(N) = \alpha_1$, $\mathcal{A}(x) = \alpha_2$ and $\mathcal{A}(S) = \alpha_3$ then e_N is $\alpha_1 = \alpha_2 \to \alpha_3$
- if $N := S\bullet$, $\mathcal{A}(N) = \alpha_1$ and $\mathcal{A}(S) = \alpha_2$ then e_N is $\alpha_2 = \forall \bullet .\alpha_1$
- if $N := \Lambda \bullet .S$, $\mathcal{A}(N) = \alpha_1$ and $\mathcal{A}(S) = \alpha_2$ then e_N is $\alpha_1 = \forall \bullet .\alpha_2$

Lemma 3.7. *Let M be a term in $\mathbf{F_{at}^{\circ\bullet}}$ such that $\Gamma \vdash_{\mathbf{F_{at}^{\circ\bullet}}} M : A$ and let $\mathcal{M}$ be an associated multisystem of M. Then it is possible to replace each type scheme variable in $\mathcal{M}$ by a type in $\mathbf{F_{at}^{\circ\bullet}}$ in such a way that the multiequations in $\mathcal{M}$ hold.*

---

[1] Or equivalently a finite sequence.

[2] We consider equality between type schemes as usual for types.

[3] For example the term $M = (\lambda x.x)(\lambda x.x)$ is to be presented in the form $(\lambda x.x)(\lambda y.y)$ for y an assumption variable distinct from x. The subterms $\lambda x.x$ and $\lambda y.y$ are considered distinct.

[4] The bound variables are needed in the definition because in terms like $M := \lambda x.yz$, we want to have tags not just for the subterms y, z, yz and M but also for x.

Proof. Suppose that $\Gamma \vdash_{\mathbf{F}_{\mathbf{at}}^{\circ\bullet}} M : A$. Then Γ defines a typing for each subterm/bounded variable N of M, which we denote by $\mathcal{B}(N)$. Let $\mathcal{A}$ be a tagging for M and $\mathcal{M}$ be its associated multisystem. Let us prove that replacing the type scheme variables $\alpha = \mathcal{A}(N)$ occurring in $\mathcal{M}$ by $\mathcal{B}(N)$ the multiequations (on $\mathbf{F}_{\mathbf{at}}^{\circ\bullet}$ types) hold.

Let e be a multiequation in $\mathcal{M}$.

If e is of the form $\alpha_1 = \alpha_2 \rightarrow \alpha_3$ then two situations may occur: i) there are terms S and P such that $\alpha_1 = \mathcal{A}(S)$, $\alpha_2 = \mathcal{A}(P)$ and $\alpha_3 = \mathcal{A}(SP)$ or ii) there are terms x and S such that $\alpha_2 = \mathcal{A}(x)$, $\alpha_3 = \mathcal{A}(S)$ and $\alpha_1 = \mathcal{A}(\lambda x.S)$. In case i), after the replacement, we obtain the $\mathcal{B}(S) = \mathcal{B}(P) \rightarrow \mathcal{B}(SP)$ and in case ii) we obtain $\mathcal{B}(\lambda x.S) = \mathcal{B}(x) \rightarrow \mathcal{B}(S)$. Hence the equations hold.

If e is of the form $\alpha_1 = \forall \bullet .\alpha_2$ then two situations may occurs: i) there is a term S such that $\alpha_1 = \mathcal{A}(S)$ and $\alpha_2 = \mathcal{A}(S\bullet)$ or ii) there is a term S such that $\alpha_1 = \mathcal{A}(\Lambda \bullet .S)$ and $\alpha_2 = \mathcal{A}(S)$. In case i), after the replacement, we obtain the $\mathcal{B}(S) = \forall \bullet .\mathcal{B}(S\bullet)$ and in case ii) we obtain $\mathcal{B}(\Lambda \bullet .S) = \forall \bullet .\mathcal{B}(S)$. Hence the equations hold. □

In general given a term M and a tagging $\mathcal{A}$, a *solution* to the associated multisystem $E := \{e_N\}_{N \in \mathcal{T}}$ is an assignment f of types of $\mathbf{F}_{\mathbf{at}}^{\circ\bullet}$ to the type scheme variables in E such that the resulting set of multiequations (now in $\mathbf{F}_{\mathbf{at}}^{\circ\bullet}$) hold.

Given a multiequation e we denote by $e[i]$ the ith element counting from the left. We denote by $e \setminus e[i]$ the multiequation which results from eliminating the ith term from e: if e is $\sigma_1 = \cdots = \sigma_i = \cdots = \sigma_n$ then $e \setminus e[i]$ is $\sigma_1 = \cdots = \sigma_{i-1} = \sigma_{i+1} = \cdots = \sigma_n$. If e is a single equation, $e \setminus e[i]$ consists in eliminating the equation e. Given two multiequations e_1 and e_2 we denote by $e_1.e_2$ the result of concatenating the two multiequations into a single multiequation in the expected way.

We consider the following rules to transform (reduce) a multisystem F.

$$\frac{E \qquad e \qquad e'}{E \quad e \setminus e[i].e'} \; (\mathrm{Join}_{i,j}), \text{ if } e[i] = e'[j]$$

$$\frac{E \qquad e}{E \quad e \setminus e[i] \quad \sigma = \tau \quad \sigma' = \tau'} \; (\mathrm{Arr}_{i,j}), \text{ if } e[i] = \sigma \rightarrow \sigma' \text{ and } e[j] = \tau \rightarrow \tau', \, i \neq j$$

$$\frac{E \qquad e}{E \quad e \setminus e[i] \quad \sigma = \tau} \; (\mathrm{Quant}_{i,j}), \text{ if } e[i] = \forall \bullet .\sigma \text{ and } e[j] = \forall \bullet .\tau, \, i \neq j$$

where E are the other multiequations in the multisystem F.

Remark 3.8. Let E be a multisystem whose multiequations hold for a particular substitution of type scheme variables by types in $\mathbf{F}_{\mathbf{at}}^{\circ\bullet}$. By applying the rules above, the multisystem E' obtained from E is such that its multiequations still hold under the same substitution.

Definition 3.9. Given a multisystem E, a *reduction sequence* is a sequence of applications of the above rules. A multisystem is called *irreducible* if it is no longer possible to apply a rule to it. By a *clash* we mean a multiequation that contains type schemes both of the form $\sigma \rightarrow \sigma'$ and $\forall \bullet .\sigma$.

The following observation is immediate:

Lemma 3.10. *The rules (considered in both directions) are sound. More precisely, for any substitution of the type scheme variables for types of $\mathbf{F}_{\mathbf{at}}^{\circ\bullet}$ the rules are sound for the standard notion of type equality.*

Definition 3.11. The length of a type scheme is defined inductively as follows:

$l(\alpha) = 1$
$l(\sigma \to \tau) = l(\sigma) + l(\tau) + 1$
$l(\forall \bullet .\tau) = l(\tau) + 1$

Lemma 3.12. *All reduction sequences of a multisystem are finite.*

Proof. Given a multisystem E we consider the sequence $\sigma_1, \ldots, \sigma_k$ of all the type schemes which appear in all the multiequations of E with any order. Take the length of these type schemes to obtain a sequence of natural numbers $n_1, \ldots, n_k$. We denote this sequence by $l(E)$. Consider the effect of the rules on the sequences $l(E)$. (Join_{ij}) simply removes an element from the sequence. $(\text{Arr}_{i,j})$ replaces an element n_k with four natural numbers $m, o, p, q < n_k$ and $(\text{Quant}_{i,j})$ replaces an element n_k with two natural numbers $m, p < n_k$. Thus reductions starting from a sequence $\sigma_1, \ldots, \sigma_k$ terminate in at most $4^{n_1} + \cdots + 4^{n_k}$ steps. $\square$

Remark 3.13. A multiequation of an irreducible multisystem contains at most one type scheme of the form $\sigma \to \tau$ and at most one type scheme of the form $\forall \bullet .\sigma$.

Definition 3.14. A *resolution* of a term M is an irreducible multisystem without clashes obtained by reducing the associated multisystem of M.

It is immediate that a resolution of a term M only contains multiequations of the following three forms (modulo permutations of the order):

1) $\alpha_1 = \cdots = \alpha_n$
2) $\alpha_1 = \cdots = \alpha_n = \forall \bullet .\sigma$
3) $\alpha_1 = \cdots = \alpha_n = \sigma \to \tau$

For 2) and 3) above, we call the multisystems $\alpha_1 = \cdots = \alpha_n$ the *body* and $\forall \bullet .\sigma$ and $\sigma \to \tau$ the *head*. We also refer to multisystems of type 1) as bodies.

Definition 3.15. Given a resolution E of a term M, its *minimisation* is constructed as follows: choose a single type scheme variable from each multiequation of type 1) and from each body of multiequations of types 2) and 3). We say that these chosen type scheme variables are *associated* to their respective bodies. Then discard all type scheme variables in the bodies of the multiequations except for the previously selected type scheme variables. Let α be a scheme variable occurring in a head of a multiequation e. If α occurs in the body of some multiequation, then we replace it in e by the type scheme variable associated to that body.

Definition 3.16. Let E be the minimisation of a resolution of a term M. Its *associated digraph* $G(M)$ is constructed as follows. Its vertices are the type scheme variables that appear in E and we have a directed edge (α_1, α_2) whenever there is a multiequation e in E such that α_2 is in its body and α_1 is in its head.

We illustrate the previous concepts with an example.
Consider the term $\Lambda \bullet .((\Lambda \bullet .x_1)\bullet)(\lambda x_1.(x_0(\Lambda \bullet .x_2)))$. We tag the subterms and bound assumption variables:

$$\Lambda \bullet .((\Lambda \bullet .x_1)\bullet)(\lambda x_1.(x_0(\Lambda \bullet .x_2))) : \alpha_0$$
$$((\Lambda \bullet .x_1)\bullet)(\lambda x_1.(x_0(\Lambda \bullet .x_2))) : \alpha_1$$
$$((\Lambda \bullet .x_1)\bullet) : \alpha_2$$
$$\lambda x_1.(x_0(\Lambda \bullet .x_2)) : \alpha_3$$
$$\Lambda \bullet .x_1 : \alpha_4$$
$$x_0(\Lambda \bullet .x_2) : \alpha_5$$
$$\Lambda \bullet .x_2 : \alpha_6$$
$$x_1 : \alpha_7$$
$$x_0 : \alpha_8$$
$$x_2 : \alpha_9$$

The associated multisystem consists of the equations:

$$\alpha_0 = \forall \bullet .\alpha_1$$
$$\alpha_2 = \alpha_3 \to \alpha_1$$
$$\alpha_4 = \forall \bullet .\alpha_2$$
$$\alpha_3 = \alpha_7 \to \alpha_5$$
$$\alpha_8 = \alpha_6 \to \alpha_5$$
$$\alpha_6 = \forall \bullet .\alpha_9$$
$$\alpha_4 = \forall \bullet .\alpha_7$$

Applying the rules we get the irreducible multisystem:

$$\alpha_0 = \forall \bullet .\alpha_1$$
$$\alpha_4 = \forall \bullet .\alpha_2$$
$$\alpha_7 = \alpha_2 = \alpha_3 \to \alpha_1$$
$$\alpha_3 = \alpha_7 \to \alpha_5$$
$$\alpha_8 = \alpha_6 \to \alpha_5$$
$$\alpha_6 = \forall \bullet .\alpha_9$$

Applying minimisation yields:

$$\alpha_0 = \forall \bullet .\alpha_1$$
$$\alpha_4 = \forall \bullet .\alpha_2$$
$$\alpha_2 = \alpha_3 \to \alpha_1$$
$$\alpha_3 = \alpha_2 \to \alpha_5$$
$$\alpha_8 = \alpha_6 \to \alpha_5$$
$$\alpha_6 = \forall \bullet .\alpha_9$$

There are no clashes. The associated graph has vertices

$$V = \{\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_8, \alpha_9\}$$

and directed edges

$$E = \{(\alpha_0, \alpha_1), (\alpha_4, \alpha_2), (\alpha_2, \alpha_1), (\alpha_2, \alpha_3), (\alpha_3, \alpha_2), (\alpha_3, \alpha_5), (\alpha_8, \alpha_6), (\alpha_8, \alpha_5), (\alpha_6, \alpha_9)\}.$$

Lemma 3.17. *A term M is typable in $\mathbf{F}_{\mathrm{at}}^{\circ\bullet}$ if and only if it has a resolution with an associated digraph not containing a cycle.*

Proof. Let M be typable in $\mathbf{F}_{\mathbf{at}}^{\circ\bullet}$. By Lemma 3.7 and Remark 3.8 we have that M has a resolution. Also, it cannot contain a cycle for this would imply an impossible condition on the possible types of subterms of M. On the other hand if M has a resolution and its associated digraph $G(M)$ does not contain a cycle then we obtain a typing for M as follows. We tag the terminal vertices (those with no paths leaving them) of $G(M)$ with $\circ$. Since there are no cycles we can tag the rest of the vertices inductively as follows. Assume that the descendants of a vertex α have already been tagged. If it has two descendants α_1 and α_2 tagged with A and B then we tag α with $A \to B$. If it has a single descendant β tagged with A then we tag it with $\forall \bullet .A$. It is easy to see by the construction of the associated graph that these are the only two possible situations. Each vertex of the associated digraph is also associated to a body in the resolution of M. Substitute all type schemes in the body by the new type associated to the vertex. With this substitution the multiequations of the resolution continue to hold. Substituting these new values in a $(x : \sigma)$ of the inferred typing of M for each x yields an environment rendering M typable in $\mathbf{F}_{\mathbf{at}}^{\circ\bullet}$. $\square$

Since it is decidable if a term M has a resolution and whether its associated digraph has a cycle we get that:

Lemma 3.18. *Typability in* $\mathbf{F}_{\mathbf{at}}^{\circ\bullet}$ *is decidable.*

Hence, combining the previous lemma with Theorem 3.5 yields:

Theorem 3.19. *Typability for* $\mathbf{F}_{\mathbf{at}}$ *in* Polymorphic Curry style *is decidable.*

The proof of Lemma 3.17 yields a method of type inference for $\mathbf{F}_{\mathbf{at}}^{\circ\bullet}$ similar to the case of simply typed calculus [Rém02][1.4.3]. The proof of Corollary 3.4 allow us to extend such type inference strategy to a method of type inference for $\mathbf{F}_{\mathbf{at}}$ restricted to redundant typings.

4. Non-redundant Typability

A natural question arises: when is a term M of $\mathbf{F}_{\mathbf{at}}$ typable with non-redundant quantifiers? For example, if a term M contains subterms $N(xX)$ and $N(xY)$, where x is the same free or bound assumption variable (after imposing the Barendregt condition), then the type of x must be of the form $\forall Z.A$ and the types of xX and xY must be equal. The type of xX is $A[X/Z]$ and the type of xY is $A[Y/Z]$. Thus Z cannot occur in A and we see that $\forall Z$ is redundant.

We have seen by Corollary 2.3 that typable Curry terms coincide with the typable terms of $\lambda^{\to}$. If a term M has a derivation yielding a typing $\Gamma \vdash M : A$ in which all the quantifiers in Γ and A are redundant then by lemma 3.11 of [Wel99] (which carries over from $\mathbf{F}$ to $\mathbf{F}_{\mathbf{at}}$) M has a derivation yielding a typing $\Gamma' \vdash M : A'$ in which all the redundant quantifiers in Γ and A have been eliminated, that is, we obtain canonically a derivation in $\lambda^{\to}$. Clearly non-redundancy is very relevant for a meaningful notion of type inference in $\mathbf{F}_{\mathbf{at}}$. A major advantage of the Polymorphic Curry style presentation of $\mathbf{F}_{\mathbf{at}}$ is that the second-order information present in the terms allows us to impose fine-grained non-redundancy conditions in the procedure for type inference.

In this Section we consider $\mathbf{F}_{\mathbf{at}}$ in Polymorphic Curry style.

Definition 4.1. Given a term M in $\mathbf{F}_{\mathbf{at}}$ typable with a type environment Γ, we say that it is a *non-redundant typing* if for any subterm of M of the form NX or $\Lambda X.N$ if we consider

the induced typings[5] $\Gamma' \vdash N : \forall Y.B$ in the first case or $\Gamma' \vdash N : B$ in the second, then Y occurs free in B in the first case or X occurs free in B in the second.

Non-redundancy involves imposing a set of *positive constraints* relative to variable occurrences for the types of certain subterms (which we call *polymorphic subterms*). We can discard these constraints partially (i.e. only impose them on a subset of such subterms) or totally without losing typability. Typability in general requires only a set of *negative constraints* imposed by the proviso of rule (GEN_X). In this section we will address the problem of typability in the setting of positive constraints on a choice of polymorphic subterms which includes both the (empty) case corresponding to general typability and the case of non-redundancy above. Given a selection c of occurrences of polymorphic subterms of a term M we call a typing of M which satisfies the positive constraints on these subterms *c-non-redundant*.

We now define the set $\mathbb{S}'$ of *type schemes* by

$$\sigma, \tau := \alpha | \sigma[X/Y] | \sigma \to \tau | \forall X.\sigma$$

where α is a type scheme variable and X and Y are type variables of $\mathbf{F_{at}}$. Type schemes of the form αs where s is a (possibly empty) string of expressions of the form $[X/Y]$ are called *variants* of α. If a type scheme is not a variant of an α then it is called *complex*. We also write $\alpha[X_1/Y_1]...[X_n/Y_n]$ as $\alpha[X_1...X_n/Y_1...Y_n]$ or $\alpha[\bar{X}/\bar{Y}]$.

We consider that $(\sigma \to \tau)[X/Y] = \sigma[X/Y] \to \tau[X/Y]$ and $(\forall Z.\sigma)[X/Y] = \forall Z.\sigma[X/Y]$ (for X and Y distinct from Z) and $(\forall Z.\sigma)[X/Z] = \forall Z.\sigma$.

If X, Y, Z, W are distinct type variables we consider that $\sigma[X/Y][Z/W] = \sigma[Z/W][X/Y]$. Also we consider that $\sigma[Y/X][W/X] = \sigma[Y/X]$ and $\sigma[X/X] = \sigma$.

We define a *generalised environment* as previously. A *multisystem* is also defined as previously.

Given a term M in $\mathbf{F_{at}}$, we define a *tagging* $\mathcal{A}$ exactly as before. We also have a *variable tagging* $\mathcal{V}$ which assigns to each *occurrence* of a subterm of the form NX or $\Lambda X.N$ of M a new type variable Y. We denote the value of $\mathcal{V}$ sometimes by X_{NX} or $X_{\Lambda X.N}$ when there is only one occurrence. We also use the notation $NX : \alpha$ for $\alpha = \mathcal{A}(NX)$. The *associated multisystem* is defined as $F \cup \{e_N\}_{N \in \mathcal{T}}$ where $\mathcal{T}$ is the set of occurrences of subterms of M which are not assumption variables and F is as described below. When N is an occurrence of an arrow application or an arrow abstraction e_N is defined as in the previous section (note that occurrences of the same subterm of this form give rise to the same multiequations). In the case of an occurrence of a subterm of the form NX we have that e_{NX} is

$$\alpha_1 = \alpha_2^W[X/W]$$

where $\alpha_1 = \mathcal{A}(NX)$, $\alpha_2 = \mathcal{A}(N)$, $W = \mathcal{V}(NX)$ and α_2^W is a fresh type scheme variable identified uniquely by α_2 and W. Also $e_{\Lambda X.N}$ is

$$\alpha_1 = \forall W.\alpha_2[W/X]$$

where $\alpha_1 = \mathcal{A}(\Lambda X.N)$, $\alpha_2 = \mathcal{A}(N)$ and $W = \mathcal{V}(\Lambda X.N)$.

Finally F consists of equations $\alpha = \forall W.\alpha^W$ for all occurences of subterms of M of the form NX, where $\alpha = \mathcal{A}(N)$ and W is the value of $\mathcal{V}$ for each occurrence of the subterm.

---

[5]Given a typing $\Gamma \vdash M : A$ and a subterm N of M there is a uniquely determined type environment Γ' and type B such that $\Gamma' \vdash N : B$. We call this the *induced* typing on N by $\Gamma \vdash M : A$.

Given a term M and taggings $\mathcal{A}$ and $\mathcal{V}$ we define a *solution* to the associated multisystem $F \cup \{e_N\}_{N \in \mathcal{T}}$ exactly as before.

We consider a binary predicate $Occ(X, \sigma)$ over the sorts of type variables and type scheme variables which will be interpreted as meaning that X must occur free in the assignment of σ for any solution of the associated multisystem.

We associate to a term M a set of formulas $\mathcal{B}(M)$ (called *basic constraints*) in the following way: for all subterms of M of the form $\Lambda X.S$ we add

$$\bigwedge_{\alpha_i} \neg Occ(X, \alpha_i)$$

where $\alpha_i = \mathcal{A}(x)$ for the free assumption variables x in S. Note that these are the constraints required for typability in general.

Given a selection c for a term M we define a set $\mathcal{K}_c(M)$ which includes $\mathcal{B}(M)$ and a possibly empty set of positive constraints constructed as follows: for subterms of M chosen by c of the form $\Lambda X.S$ we add

$$Occ(X, \mathcal{A}(S))$$

and for subterms of M chosen by c of the form NX we add

$$Occ(W, \alpha^W)$$

with $\alpha = \mathcal{A}(N)$ and W is the value of $\mathcal{V}$ for that occurrence of the subterm.

Note that $Occ(X, \mathcal{A}(S))$ implies that $Occ(W, \mathcal{A}(S)[W/X])$ where $W = \mathcal{V}(\Lambda X.S)$.

For other properties of substitution and the occurrence predicate (which will be used in the proofs ahead) see the *occurrence axioms* listed at the beginning of Section 4.1.

Note that if we consider the term $M = \lambda xy.y(I(xX))(I(xY))$ where $I = \lambda z.z$ then by the definition of tagging we must first ensure that the bound variables are all distinct. Thus the second occurrence of the subterm I would become $I' = \lambda w.w$ where w is an assumption variable distinct from z. We also take occasion to note that it can happen that for a term N with a *redundant* typing $\Gamma \vdash N : A$ we can still have that A does not contain any redundant quantifiers.

When we write $\sigma[X/Y]$ we are assuming that σ represents a possible type A in which X is free for Y in A. We call such values of σ *adequate* for $[X/Y]$. For a sequence of substitutions σs we define A being adequate to s inductively. If s is a single substitution then the definition is as previously. Otherwise let $s = s'[Y/X]$. Then a type A is adequate for s if As' is adequate for $[Y/X]$.

A partial substitution $\{X/Y\}$ is formally given by two type variables X and Y and a strictly increasing (possibly empty) sequence of natural numbers $n_1, \ldots, n_k$. The interpretation is as follows. We consider the sequence of all the free occurrences of Y as they occur from the left. Then the free occurrences of Y in positions $n_1, \ldots, n_k$ in that sequence are substituted by X. Obviously such a partial substitution can only be applied to types that have at least n_k free occurrences of Y.

We thus can extend the notion of adequacy to partial substitutions $\{Y/X\}$ and to partial substitution sequences s.

Note that if $B = A[X/Y]$ with A adequate for $[X/Y]$ then $A = B\{Y/X\}$ for some partial substitution $\{Y/X\}$ with Y free for X in B for the occurrences of X which are substituted. If $A = Bs$ then $B = A\mathfrak{s}$ where $\mathfrak{s}$ is some sequence of partial substitutions involving the variables in s. We can always assume that s represents a *simultaneous* substitution and likewise for $\mathfrak{s}$.

Since a term has a c-non-redundant typing if and only if it has a c-non-redundant typing in which all the bound variables are distinct and different from the type variables that appear in the term (and if we need to rename a bound variable we can do this in such a way that this situation is preserved) we may assume that the solutions (type values) we are considering are adequate for the substitutions that appear in the associated multisystem.

Lemma 4.2. *Let M be a term in $\mathbf{F_{at}}$ such that $\Gamma \vdash_{\mathbf{F_{at}}} M : A$ is a c-non-redundant typing and let $\mathcal{M}$ be its associated multisystem for a given $\mathcal{A}$ and $\mathcal{V}$. Then we can replace each type scheme variable in $\mathcal{M}$ by a type in such a way that the multiequations in $\mathcal{M}$ hold as well as the corresponding constraints $\mathcal{K}_c(M)$.*

Proof. This is shown in the same way as in Lemma 3.7. We need only consider two further cases. Consider a c-non-redundant typing Γ for M and let $\Gamma \vdash NX : A$ and $\Gamma \vdash N : B$ for a subterm NX of M. Consider a given occurrence of NX with W the value of $\mathcal{V}$ for that occurrence. Let $\alpha = \mathcal{A}(NX)$ and $\beta = \mathcal{A}(N)$. Now B must be of the form $\forall Y.C$ and we have by (INST) that $A = C[X/Y]$. Renaming the bound variable Y to W (we can assume that there are no variable clashes) we get that $B = \forall W.C[W/Y]$. Hence $A = C[X/Y] = C[W/Y][X/W]$. Hence we put $\beta^W = C[W/Y]$ and the equation $\alpha = \beta^W[X/W]$ holds. And also $\beta = \forall W.\beta^W$. The other case is similar. That the constraints hold follows from the definition of c — non-redundancy. $\square$

Given a multiequation e we denote by $e[\bar{X}/\bar{Y}]$ the multiequation for which

$$e[\bar{X}/\bar{Y}][i] = e[i][\bar{X}/\bar{Y}]$$

Given a term M with a variable tagging $\mathcal{V}$ we denote by $\circ$ a type variable which does not occur among the variables in the substitutions of the associated multisystem of M. Let $\bar{W}$ be the sequence of all such variables and consider the substitution sequence $[\circ/\bar{W}]$. Then for any substitution s involving variables of $\bar{W}$ we have that $As[\circ/\bar{W}] = A[\circ/\bar{W}]$. Here, when considering A as part of the solution to the associated multisystem, without loss of generality we can assume that the types A never contain $\circ$ either as a free or bound variable. Given M and $\mathcal{V}$ we use the notation $\bigcirc = [\circ/\bar{W}]$.

We consider the following rules to transform a multisystem F. We have $(\mathrm{Arr}_{i,j})$ as in the previous section. In addition, we have the following two rules:

$$\frac{E \qquad e}{E \quad e \setminus e[i] \quad \sigma = \tau[X/Y] \quad \tau = \sigma[Y/X]} \ (\mathrm{Quant}_{i,j}), \ e[i] = \forall X.\sigma, \ e[j] = \forall Y.\tau, \ i \neq j$$

$$\frac{E \qquad e \qquad e'}{E \quad (e \setminus e[i]) \bigcirc .e'\bigcirc} \ (\mathrm{Join}_{i,j}), \ e[i] = \alpha s, \ e'[j] = \alpha s'$$

In the above E are the other multiequations in the multisystem F. Note that (Arr) and (Quant) are bi-directional.

Definition 4.3. Given a multisystem E, a *reduction sequence* is a series of applications of the above rules. A multisystem is called *irreducible* if it is no longer possible to apply a rule to it.

Remark 4.4. We can transform a reduction sequence in the above sense into a reduction sequence in $\mathbf{F}_{\mathbf{at}}^{\circ\bullet}$ by applying the translation $[\cdot]$ of the previous section where αs in $\mathbb{S}'$ is translated via $[\cdot]$ to α in $\mathbb{S}$.

This guarantees the existence of an irreducible multisystem for typable terms (see Lemma 3.12). Alternatively, we can characterise irreducible multisystems as those which translate to irreducible multisystems in $\mathbf{F}_{\mathbf{at}}^{\circ\bullet}$.

Definition 4.5. A type scheme variable α in a multisystem is called *terminal* if no variant of it can be expressed in terms of complex type schemes with variants of the other type scheme variables in the multisystem. Terminals are organised into *groups*. Two terminals α, α' belong to the same group iff $\alpha s = \alpha' s'$ for some s, s'.

Note that the reduction rules are sound for the standard notion of equality and the terminals are preserved by reduction.

Remark 4.6. To obtain a solution to a multisystem it is not enough to give values to the terminal scheme variables. The type structure of the non-terminals is, however, determined once we have the value of the terminals, as is easily seen by Remark 4.4.

What happens is that the free variables of the non-terminals are not uniquely determined by the terminals. For example, consider the multisystem associated to the term $(xX)y$. Let $(xX)y : \alpha_1$, $y : \alpha_2$, $xX : \alpha_3$ and $x : \alpha_4$. The associated multisystem contains $\alpha_3 = \alpha_2 \to \alpha_1$, $\alpha_3 = \alpha_4^W[X/W]$ and $\alpha_4 = \forall W.\alpha_4^W$ where $W = \mathcal{V}(xX)$. Here α_1 and α_2 are the terminals. Consider the values $\alpha_1 = (X \to X)$ and $\alpha_2 = \circ$. Then we have the following distinct solutions:

$$\alpha_3 = \circ \to (X \to X) \text{ and } \alpha_4 = \forall W.\circ \to (X \to W)$$
$$\alpha_3 = \circ \to (X \to X) \text{ and } \alpha_4 = \forall W.\circ \to (W \to W)$$

A *structure* is an expression built up from a "hole" symbol and $\forall X.$ and $\to$. We say that a type A is *built up* from types $A_1, \ldots, A_n$ if A can be obtained by considering a certain structure in which the "holes" are filled with types among $A_1, \ldots, A_n$.

Lemma 4.7. *Consider the associated multisystem of a term M with a c-non-redundant typing. Given a solution consider the values of the terminals $A_1, \ldots, A_n$. Then for each non-terminal αs (in particular α) there is a partial substitution sequence $\mathfrak{s}_1, \ldots, \mathfrak{s}_m$ such that the value of αs is built up from $A_1\mathfrak{s}_1, \ldots, A_n\mathfrak{s}_m$, where $m \geq n$ is bounded by the length of M and the A_i may be repeated in this list.*

Remark 4.8. Since the associated multisystem determines the structure for each non-terminal, to specify a solution it is enough to give values for the terminals and a sequence $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$ for each non-terminal applying to the terminals which occur in its decomposition defined by the associated multisystem.

Proof of Lemma 4.7. Consider the irreducible multisystem of M. Each non-terminal α occurs in a unique multiequation containing a complex type scheme, an arrow or quantifier element. We call the type scheme variables that occur in this complex type scheme the *descendants* of α. In this way we can associate each non-terminal with a uniquely determined non-cycling digraph $G(\alpha)$ having terminals as leaves. To prove the lemma proceed by induction on the structure of $G(\alpha)$.

If α is non-terminal then its multiequation contains either $\alpha s = \forall X.\sigma$ or $\alpha s = \sigma \to \tau$. Note that if a type A is built up from types A_i then $A\mathfrak{s}$ is built up from types $A\mathfrak{s}_i$ where $\mathfrak{s}_i$ are restrictions of $\mathfrak{s}$. If $\alpha s = \sigma \to \tau$ then by hypothesis σ and τ are built from partial substitutions on terminals. But $\alpha = (\sigma \to \tau)\mathfrak{s}$. Decomposing $\mathfrak{s}$ into partial substitution sequences $\mathfrak{s}'$ and $\mathfrak{s}''$ on σ and τ respectively we get that $\alpha = \sigma\mathfrak{s}' \to \tau\mathfrak{s}''$. The case of $\alpha s = \forall X.\sigma$ is analogous. $\square$

Definition 4.9. An *occurrence matrix* of a term M is a set of formulas which consists of $Occ(X, \alpha)$ or $\neg Occ(X, \alpha)$ for each type variable X and type scheme variable α occurring in the associated multisystem of M.

Remark 4.10. If a term M has a c-non-redundant typing then it generates a solution to its associated multisystem which in turn generates an occurrence matrix. Notice that given this matrix we can determine all the occurrences or non-occurrences of the type variables of the multisystem in substitutions αs of type scheme variables. It is clear that $\mathcal{K}_c(M)$ has to be satisfied, since the typing is c-non-redundant.

Lemma 4.11. *Let M be a term with a c-non-redundant type environment Γ and let $\star$ be a type variable not occurring in Γ nor in the associated multisystem of M. Then M has a c-non-redundant type environment Γ' in which the free type variables either occur in the associated multisystem of M or are $\star$.*

Proof. Notice that substitution sequences preserve equality. We use the following fact: let $\bar{R}$ be all the type variables in Γ which do not occur in the associated multisystem of M. In particular the variables in $\bar{R}$ do not occur among the variables in s in a reduction sequence. Consider any equation $As = Bt$. Then we have that $As[\star/\bar{R}] = Bt[\star/\bar{R}]$ and so $A[\star/\bar{R}]s = B[\star/\bar{R}]t$. Thus if we apply $[\star/\bar{R}]$ to the types in Γ it is clear that we still obtain a typing and thus a solution to the associated multisystem. The typing is still c-non-redundant because we are not altering the variables belonging to the occurrence matrix of M. $\square$

We call such a typing *adequate*. Hence to look for c-non-redundant typings for a typable term M we need only look for adequate typings. Hence to look for solutions to a multisystem we need only look for types with free variables occurring in the multisystem or equal to a distinguished type variable $\star$. We call such solutions *adequate* solutions and the values attributed to the type scheme variables adequate values. The following obvious result is important:

Lemma 4.12. *Given a term M and adequate values for the terminals of its associated multisystem there are only a finite number of possible adequate values on the non-terminals.*

This follows from Lemma 4.7 noticing that the partial substitution sequences can involve only the finite number of variables which occur in the associated multisystem.

Given a type A, its *variable sequence* is the sequence, as they occur, of all free type variables in A[6]. Note that given a solution to the associated multisystem of a term M the values of terminals in the same group have the same structure.

Lemma 4.13. *Consider an adequate solution to the associated multisystem of a term M which yields a c-non-redundant typing. Then if we substitute the values of the terminals in a given group by types having a different structure with the same variable sequences we can obtain another solution which also yields a c-non-redundant typing.*

---

[6]For example for $A := X \to (Y \to X)$ the variable sequence is X, Y, X.

Proof. By Lemma 4.7 the non-terminals are given by partial substitution sequences on terminals and since the variable sequences of the terminals are the same after altering the structure we can use the same partial substitution sequences to obtain the values of the non-terminals. Consider αs and $\alpha' s'$ belonging to the same multiequation in the reduced multisystem and let α and α' have values A and A'. Then $As = A's'$. The values A and A' are built up from partial substitutions on the values of the terminals in $G(\alpha) = G(\alpha')$. Let these values be $A_1, \ldots, A_n$ and $A'_1, \ldots, A'_n$. So to check if $As = A's'$ (after changing the structure on terminals keeping the variable sequence) we only need to check if $A_i s = A'_i s'$ for $i = 1, \ldots, n$ with the changes above. We do not have to worry about renamed bound variables in A or A' because we are working within a fixed irreducible multisystem and thus with a fixed structure for A and A'. But observe that if $Bs = B's'$ then $Cs = C's'$ where C has the variable sequence as B and C' the same variable sequence as B'. It follows that when we alter the structure of the values of terminals in the above way that $\alpha s = \alpha' s'$ still holds for the altered values. c-non-redundancy follows by observing that the alteration of the structure does not affect the occurrence matrix. $\qquad \square$

Hence we can now assume that in our adequate solutions the terminals are given by types of the form $X_1 \to X_2 \to \ldots \to X_n$. We call this the *minimal form*. We refer to the position of X_i as the ith place. We call n the *size* of the type. By the result of removing the ith place from a type in minimal form (of a variable sequence of size greater than 1) we mean the type $X_1 \to \ldots \to X_{i-1} \to X_{i+1} \to \ldots \to X_n$.

Given an adequate solution, consider the values of terminals in a group in minimal form. They will have the same length n. A place k is called *essential* if after we alter the solution in such a way that this place is removed from the values of all terminals in a group the occurrence matrix changes. The values of the other type scheme variables are given by the restrictions of their respective partial substitutions sequences on terminals. These values are a solution because if for two terminals α and α' their values in minimal form satisfy $\alpha s = \alpha' s'$ then they continue to satisfy this equation if we remove a place from both values.

Notice that removing a place cannot alter a non-occurrence constraint.

Lemma 4.14. *If a term M has a c-non-redundant typing then it has a c-non-redundant typing given by an adequate solution in minimal form such that the maximum length of its values on the terminals is bounded by the order $\mathcal{O}(mn)$, where n is the number of type variables and m is the number of type scheme variables in the associated multisystem of M.*

Proof. Consider an adequate minimal solution for the associated multisystem of M. Given a group of terminals assume that a place is not essential. Then if we remove it we obtain a solution which gives a c-non-redundant typing. Hence we only need to study the number of essential places on values of terminals. Observe that *removing two places in a given group cannot affect the occurrence condition for the same type variable and the same type scheme*. For each place n consider the set $S(n)$ of affected pairs (X, α). Then the sets $S(n)$ are disjoint for different n. It follows that the number of essential places in a group of non-terminals cannot exceed the product of the number of type variables and the number of type scheme variables in the associated multisystem. $\qquad \square$

The bound in the lemma can in fact be sharpened to the order of the cardinality of $\mathcal{K}_c(M)$, for in the proof above we are in fact only interested in not affecting certain pairs (X, α) corresponding to $\mathcal{K}_c(M)$.

As a corollary we get:

Theorem 4.15. *Given a typable term M in $\mathbf{F_{at}}$ it is decidable whether it has a c-non-redundant typing.*

Given a term which admits a *c*-non-redundant typing, the proof of the theorem above furnishes us with a method for type inference yielding an adequate, minimal *c*-non-redundant typing. In particular it furnishes us a method of type inference for typable terms.

Note that when *c* chooses all the polymorphic subterms of a term M then *c*-non-redundancy coincides with non-redundancy (cf. Definition 4.1).

In the following subsection we give concrete examples on how we check for non-redundant typability and how we can find minimal typings whose values on terminals have only essential places.

4.1. **Examples.** In order to give some practical examples we consider a multi-sorted first-order language with equality endowed with a single binary predicate Occ and two sorts $TypeVariable$ and $TypeScheme$. Occ has signature $TypeVariable \times TypeScheme$. The occurrence theory $\mathcal{C}$ includes, besides the standard axioms for equality, the following *occurrence axioms*:

$$Occ(X, \sigma \to \tau) \leftrightarrow Occ(X, \sigma) \vee Occ(X, \tau)$$

$$Occ(Y, \forall X.\sigma) \leftrightarrow Occ(Y, \sigma) \quad \text{for } Y \neq X$$

$$\neg Occ(X, \forall X.\sigma)$$
$$\neg Occ(X, \sigma[Y/X]) \quad \text{for } X \neq Y$$
$$Occ(X, \sigma) \leftrightarrow Occ(X, \sigma[Z/Y]) \quad \text{for } X \neq Y \text{ and } Z \neq X$$

$$Occ(X, \sigma) \to Occ(Y, \sigma[Y/X])$$

$$\sigma[Y/X] = \sigma[Z/X] \wedge Occ(X, \sigma) \to Y = Z$$

If type scheme variables are interpreted as types and we assume that for all the resulting expression $A[Y/X]$, Y is free for X in A, then these axioms are clearly sound for the ordinary notion of free occurrence of a variable in a type.

Consider the example of the beginning of this section: a term M containing subterms of the form $N(xX)$ and $N(xY)$.

Let $x : \alpha_3$, $xX : \alpha_2$, $xY : \alpha_1$. Put $X_1 = \mathcal{V}(xX)$ and $X_2 = \mathcal{V}(xY)$. The associated multisystem, after applying some rules, will include $\alpha_1 = \alpha_2$. It will also include

$$\alpha_3 = \forall X_1 \alpha_3^{X_1}$$
$$\alpha_3 = \forall X_2 \alpha_3^{X_2}$$
$$\alpha_2 = \alpha_3^{X_1}[X/X_1]$$
$$\alpha_1 = \alpha_3^{X_2}[Y/X_2]$$

Also $Occ(X_1, \alpha_3^{X_1})$ and $Occ(X_2, \alpha_3^{X_2})$.

Applying (Quant) to the first two equations yields:

$$\alpha_3^{X_1} = \alpha_3^{X_2}[X_1/X_2]$$
$$\alpha_3^{X_2} = \alpha_3^{X_1}[X_2/X_1]$$

Hence

$$\alpha_2 = \alpha_3^{X_2}[X_1/X_2][X/X_1]$$
$$\alpha_1 = \alpha_3^{X_2}[Y/X_2]$$

Since X_1 does not occur in $\alpha_3^{X_2}$ we get

$$\alpha_2 = \alpha_3^{X_2}[X/X_2]$$
$$\alpha_1 = \alpha_3^{X_2}[Y/X_2]$$

Hence, since $\alpha_2 = \alpha_1$ by the last occurrence axiom we get that $X = Y$, a contradiction.

Consider the term $(y(xX))(xY)$. Set $x : \alpha_1$, $xX : \alpha_2$, $xY : \alpha_3$, $y(xX) : \alpha_4$, $(y(xX))(xY) :$ α_5, $y : \alpha_6$ and $X_1 = \mathcal{V}(xX)$, $X_2 = \mathcal{V}(xY)$. The associated multisystem contains:

$$\alpha_6 = \alpha_2 \to \alpha_4$$
$$\alpha_4 = \alpha_3 \to \alpha_5$$
$$\alpha_1 = \forall X_1.\alpha_1^{X_1}$$
$$\alpha_1 = \forall X_2.\alpha_1^{X_2}$$
$$\alpha_2 = \alpha_1^{X_1}[X/X_1]$$
$$\alpha_3 = \alpha_1^{X_2}[Y/X_2]$$

and also $Occ(X_1, \alpha_1^{X_1})$, $Occ(X_2, \alpha_1^{X_2})$.

Applying (Quant) to the third and fourth equations yields:

$$\alpha_1^{X_1} = \alpha_1^{X_2}[X_1/X_2]$$
$$\alpha_1^{X_2} = \alpha_1^{X_1}[X_2/X_1]$$

Hence $\neg Occ(X_2, \alpha_1^{X_1})$.

Let us put $\alpha_1^{X_1} = X_1$. Then $\alpha_1 = \forall X_1.X_1$ (so $x : \forall X_1.X_1$), $\alpha_2 = X, \alpha_3 = Y$ and $\alpha_6 = X \to Y \to \alpha_5$ (so $y : X \to Y \to \alpha_5$).

Consider now $(xX)Y$.
Let $x : \alpha_1, xX : \alpha_2, (xX)Y : \alpha_3$, $X_1 = \mathcal{V}(xX), X_2 = \mathcal{V}((xX)Y)$.
The associated multisystem contains:

$$\alpha_3 = \alpha_2^{X_2}[Y/X_2]$$
$$\alpha_2 = \alpha_1^{X_1}[X/X_1]$$

with $Occ(X_2, \alpha_2^{X_2})$ and $Occ(X_1, \alpha_1^{X_1})$. Since the associated multisystem contains:

$$\alpha_1 = \forall X_1.\alpha_1^{X_1}$$
$$\alpha_2 = \forall X_2.\alpha_2^{X_2}$$

we get

$$\alpha_1^{X_1}[X/X_1] = \forall X_2.\alpha_2^{X_2}$$

Occurrence axioms yield $Occ(X, \alpha_1^{X_1}[X/X_1])$ and thus $Occ(X, \alpha_2^{X_2})$. We also have $Occ(X_2, \alpha_2^{X_2})$. Also it is clear that $\neg Occ(X_1, \alpha_2^{X_2})$. Take $\alpha_2^{X_2} = X_2 \to X$. Since

$$\alpha_1^{X_1}[X/X_1] = \forall X_2.X_2 \to X$$

and $Occ(X_1, \alpha_1^{X_1})$ we must have $\alpha_1^{X_1} = \forall X_2.X_2 \to X_1$ and hence

$$x : \forall X_1.\forall X_2.X_2 \to X_1$$

is a non-redundant typing.

Another example is $\Lambda X.(xY)$.

Put $x : \alpha_1, xY : \alpha_2, \Lambda X.(xY) : \alpha_3$ and $X_1 = \mathcal{V}(xY)$. Then $\alpha_3 = \forall X.\alpha_2$, $\alpha_2 = \alpha_1^{X_1}[Y/X_1]$ and $\alpha_1 = \forall X_1.\alpha_1^{X_1}$ with $Occ(X_1, \alpha_1^{X_1})$, $Occ(X, \alpha_2)$ and $\neg Occ(X, \alpha_1)$.

Substitution yields $Occ(X, \alpha_1^{X_1}[Y/X_1])$ and $\neg Occ(X, \forall X_1.\alpha_1^{X_1})$ and the occurrence axioms imply the contradiction $Occ(X, \alpha_1^{X_1})$ and $\neg Occ(X, \alpha_1^{X_1})$.

A more interesting example: $(\Lambda X.\lambda x.x)Y$. In this case we get the non-redundant typing $x : X$ as can easily be checked.

5. Final Remarks

Remark 5.1. A procedure for type inference will take as input a typable term and generate a typing (or class of typings which is general enough in some sense). By the results of Section 4 we see that the output of such a procedure will involve not only specifying occurrence (or non-occurrence) constraints for variables and terminal type scheme variables but also will involve a bound on the number of occurrences that may be required to obtain a valid typing.

For Lemma 4.14 in the case in which c does not choose any polymorphic subterms, that is, $\mathcal{K}_c(M) = \mathcal{B}(M)$ does not contain any positive constraints, we obtain a method of type inference for $\mathbf{F_{at}}$ in Polymorphic Curry style.

Further work will involve a detailed analysis of the complexity of the procedure involved, preferably in the context of an actual implementation.

Remark 5.2. In a recent work [PT21] Pistone and Tranchini proved that typability is decidable for Curry style $\mathbf{F_{at}}$. They do this by means of an alternative approach which involves reducing typability to the type checking problem (TC). To show that TC is decidable, they employ a decidable restriction, modeled after $\mathbf{F_{at}}$, of the general undecidable second-order unification problem. The restricted problem is shown to be decidable by reduction to the first-order case where the problem of finding variable cycles is decidable. Type checking is reduced to such a restricted second-order unification problem. Our present work on the other hand analyzes typability not only for Curry style terms but also for Polymorphic Curry style terms and most of the paper is focused on obtaining a procedure for type inference embodying possible constraints.

The question we address in this paper of finding non-redundant typings for Polymorphic Curry terms can be posed also for Curry terms. Note that the question of the typability of a Curry term M can be seen as involving the analysis of possible Polymorphic Curry terms N such that $[N] = M$. An assumption variable x occurring in a Curry term M is called

non-applying if it does not occur in a subterm of the form xy. We call a Polymorphic Curry term N such that $[N] = M$ *non-trivial* for M if N is not equal to M or obtained from M by replacing non-applying assumption variables x in M by xX for some type variable X. A typing for such an N is called a *non-trivial* typing of M. For the Curry system the following important questions can be asked:

(1) Given a Curry term M is it decidable whether M has a non-trivial non-redundant typing?[7]
(2) In searching for a non-trivial non-redundant typing for M do we need to consider only a finite number (bounded by the complexity of M) of Polymorphic Curry terms N such that $[N] = M$?
(3) for M having a non-trivial non-redundant typing is there a procedure to generate such typings?

It seems plausible that question (2) can be answered in the affirmative, specially if we consider arguments based on strong normalisation as in Remark 5.3. In this case it is readily seen that the results of the present work allow us to answer (1) in the affirmative and that we obtain an effective procedure for question (3).

We also observe that in an implementation of $\mathbf{F_{at}}$ in actual programming languages the questions above are relevant. Redundant typing can be seen as problematic in terms of being computationally insignificant and a waste of resources.

Remark 5.3. We mentioned in the above remark that TC for $\mathbf{F_{at}}$ in the Curry system has been shown to be decidable in [PT21] using a second-order unification technique. It seems there could be an alternative approach (perhaps computationally more efficient) based on the Polymorphic Curry style and the fact that $\mathbf{F_{at}}$ has strong normalisation. Given an environment Γ, a type A and a Curry term M, the above problem is equivalent to finding a Polymorphic Curry term N such that $[N] = M$ and $\Gamma \vdash N : A$.

The number of instantiations in a derivation of N is clearly bounded by the number of universal abstractions in the derivation and the total number of universal abstractions in Γ, so we only need to study the bounds of these. But what about series of repeated applications of INST and GEN on the same variable? It would seem that we need normalisation arguments like [Wel99] [Lemma 3.2 or Lemma 3.4]. Also

Lemma 5.4. *Suppose $\Gamma \vdash_{\mathbf{F_{at}}} M : A$ where M is a Polymorphic Curry term and let M contain a universal abstraction on X, with X not occurring in Γ or in A. Then there is a term M' such that $[M] = [M']$, $\Gamma \vdash_{\mathbf{F_{at}}} M' : A$ and M' has one less occurrence of universal abstractions on X.*

In the case in which Γ and A are canonical (in the terminology of [Wel99]), that is, have no redundant quantifiers, this follows from [Wel99] Lemma 3.11.

It seems plausible that using this lemma and normalisation ($\mathbf{F_{at}}$ has strong normalisation [FF13]) we can show that there are only a finite number of possible terms N with $[N] = M$ which we need to consider to decide if $\Gamma \vdash N : A$. Of course if M is not normal then there can be no normal N such that $[N] = M$. However we could restrict reduction to polymorphic redexes only. In the derivation of such reduced terms a INST can never come after a GEN. Also by canonicity sequences of GENs must be of length bounded by the structure of Γ. Sequences of INST are bounded by the number of previous GENs. Hence

---

[7]This is non trivial. For example given a Curry term M, one cannot simply consider $\Lambda X.M$ for a certain X as the example $(\lambda x.xy)z$ shows.

the number of combinations of INST and GEN is finite and bounded by the structure of Γ and A and thus we need only consider a finite number of terms N such that $[N] = M$ and the problem above is very likely decidable. We intend to proceed this line of research in future work.

Acknowledgments

References

[DR19]   A. Dudenhefner and J. Rehof. A simpler undecidability proof for system F inhabitation. *TYPES 2018, Leibniz International Proceedings in Informatics (LIPIcs)*, 130, 2019.

[Fer06]   F. Ferreira. Comments on predicative logic. *Journal of Philosophical Logic*, 35:1–8, 2006.

[Fer17a] G. Ferreira. Eta-conversions of IPC implemented in atomic F. *Logic Journal of the IGPL*, 25(2):115–130, 2017.

[Fer17b] G. Ferreira. Rasiowa–Harrop disjunction property. *Studia Logica*, 105:649–664, 2017.

[FF13]   F. Ferreira and G. Ferreira. Atomic polymorphism. *The Journal of Symbolic Logic*, 78(1):260–274, 2013.

[Gir71]   J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application a l'elimination des coupures dans l'analyse et la theorie des types. In *Proceedings of the second scandinavian logic symposium (J. E. Fenstad, editor)*, pages 63–92. North Holland, 1971.

[GLT89]   J-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[Lö76]   M. H. Löb. Embedding first order predicate logic in fragments of intuitionistic logic. *The Journal of Symbolic Logic*, 41(4):705–718, 1976.

[Pro21]   M. Clarence Protin. Type inhabitation of atomic polymorphism is undecidable. *Journal of Logic and Computation*, 31(2):416–425, 2021.

[PT21]   Paolo Pistone and Luca Tranchini. What's Decidable About (Atomic) Polymorphism? In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction (FSCD 2021)*, volume 195 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:23, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14265, doi:10.4230/LIPIcs.FSCD.2021.27.

[PTP22]   P. Pistone, L. Tranchini, and M. Petrolo. The naturality of natural deduction (II). On atomic polymorphism and generalized propositional connectives. *Studia Logica*, 110:545–592, 2022.

[Rém02]   D. Rémy. Using, Understanding, and Unraveling the OCaml Language. In Gilles Barthe, editor, *Applied Semantics. Advanced Lectures. LNCS 2395.*, pages 413–537. Springer Verlag, 2002.

[Rey74]   J. C. Reynolds. Towards a theory of type structure. In *Colloque sur la programmation (B. Robinet, editor)*, volume 19 of *Lecture Notes in Computer Science*. Springer Verlag, 1974.

[SF20]   J. Espírito Santo and G. Ferreira. A refined interpretation of intuitionistic logic by means of atomic polymorphism. *Studia Logica*, 108:477–507, 2020.

[TPP19]   L. Tranchini, P. Pistone, and M. Petrolo. The naturality of natural deduction. *Studia Logica*, 107:195–231, 2019.

[Wel99]   J. B. Wells. Typability and type checking in System F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98(1-3):111–156, 1999.