Local Type Inference for Polarised System F with Existentials

ANONYMOUS AUTHOR(S)

CHANGE!! This paper addresses the challenging problem of type inference for Impredicative System F with existential types, a critical aspect of many programming languages. While System F serves as the basis for type systems in numerous languages, existing type inference techniques for Impredicative System F are undecidable due to the presence of existential (∃) and polymorphic (∀) types. Consequently, current algorithms are often ad-hoc and sub-optimal. This paper presents novel contributions in the form of a local type inference algorithm for Impredicative System F with existential types. The algorithm introduces innovative techniques, such as a unique combination of unification and anti-unification, a full correctness proof, and the use of control structures inspired by Call-By-Push-Value . Additionally, the paper discusses a type inference framework that allows the algorithm to be applied to different type systems, offering insights into the under-researched area of impredicative existential type inference.

CCS Concepts: • **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Additional Key Words and Phrases: Type Inference, System F, Call-by-Push-Value, Polarized Typing, Focalisation, Subtyping

ACM Reference Format:
Anonymous Author(s). 2018. Local Type Inference for Polarised System F with Existentials. *J. ACM* 37, 4, Article 111 (August 2018), 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

2 OVERVIEW

2.1 What types do we infer?

2.2 The Language of Types

The types of $F^{\pm}\exists$ are given in fig. 1. They are stratified into two syntactic categories (polarities): positive and negative, similarly to the Call-By-Push-Value system [Levy 2006]. The negative types represent computations, and the positive types represent values:

- α^- is a negative type variable, which can be taken from a context or introduced by ∃.
- a function $P \to N$ takes a value as input and returns a computation;
- a polymorphic abstraction $\forall\overrightarrow{\alpha^+}.\ N$ quantifies a computation over a list of positive type variables $\overrightarrow{\alpha^+}$. The polarities are chosen to follow the definition of functions.
- a shift $\uparrow P$ allows a value to be used as a computation, which at the term-level corresponds to a pure computation **return** v.
+ α^+ is a positive type variable, taken from a context or introduced by ∀.
+ $\exists\overrightarrow{\alpha^-}.\ P$, symmetrically to ∀, binds negative variables in a positive type P.
+ a shift $\downarrow N$, symmetrically to the up-shift, thunks a computation, which at the term-level corresponds to $\{c\}$.

0004-5411/2018/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

Fig. 1. Declarative Types of $F^{\pm}\exists$

Definitional Equalities. For simplicity, we assume alpha-equivalent terms equal. This way, we assume that substitutions do not capture bound variables. Besides, we equate $\forall\overrightarrow{\alpha^+}.\ \forall\overrightarrow{\beta^+}.\ N$ with $\forall\overrightarrow{\alpha^+}, \overrightarrow{\beta^+}.\ N$, as well as $\exists\overrightarrow{\alpha^-}.\ \exists\overrightarrow{\beta^-}.\ P$ with $\exists\overrightarrow{\alpha^-}, \overrightarrow{\beta^-}.\ P$, and lift these equations transitively and congruently to the whole system.

2.3 The Language of Terms

In fig. 2, we define the the language of terms of $F^{\pm}\exists$. The language combines System F with the Call-By-Push-Value approach.

- $+$ x denotes a (positive) term variable; Ilya: why no negatives?
- $+$ $\{c\}$ is a value corresponding to a thunked or suspended computation;
- $\pm$ $(c : N)$ and $(v : P)$ allow one to annotate positive and negative terms;
- $-$ **return** v is a pure computation, returning a value;
- $-$ $\lambda x : P.\ c$ and $\Lambda\alpha^+.\ c$ are standard lambda abstractions. Notice that we require the type annotation for the argument of λ;
- $-$ **let** $x = v;\ c$ is a standard let, binding a value v to a variable x in a computation c;
- $-$ Applicative let forms **let** $x : P = v(\vec{v});\ c$ and **let** $x = v(\vec{v});\ c$ operate similarly to the bind of a monad: they take a suspended computation v, apply it to a list of arguments, bind the result (which is expected to be pure) to a variable x, and continue with a computation c. If the resulting type of the application is unique, one can omit the type annotation, as in the second form: it will be inferred by the algorithm;
- $-$ **let**$^{\exists}(\overrightarrow{\alpha^-}, x) = v;\ c$ is the standard unpack of an existential type: expecting v to be an existential type, it binds the packed negative types to a list of variables $\overrightarrow{\alpha^-}$, binds the body of the existential to x, and continues with a computation c.

Missing constructors. Notice that the language does not have first-class applications: their role is played by the applicative let forms, binding the result of a *fully applied* function to a variable. Also notice that the language does not have a type application (i.e. the eliminator of $\forall$) and dually, it does not have pack (i.e. the constructor of $\exists$). This is because the instantiation of polymorphic and existential types is inferred by the algorithm. Ilya: refer to the extension chapter

Computation Terms

$$
\begin{array}{lll}
c,\ d & ::= & \\
 & \mid & (c : N) \\
 & \mid & \lambda x : P.\ c \\
 & \mid & \Lambda\alpha^+.\ c \\
 & \mid & \textbf{return}\ v \\
 & \mid & \textbf{let}\ x = v;\ c \\
 & \mid & \textbf{let}\ x : P = v(\vec{v});\ c \\
 & \mid & \textbf{let}\ x = v(\vec{v});\ c \\
 & \mid & \textbf{let}^{\exists}(\overrightarrow{\alpha^-}, x) = v;\ c
\end{array}
$$

Value Terms

$$
\begin{array}{lll}
v,\ w & ::= & \\
 & \mid & x \\
 & \mid & \{c\} \\
 & \mid & (v : P)
\end{array}
$$

Fig. 2. Declarative Terms of $F^{\pm}\exists$

2.4 The key ideas of the algorithm

3 DECLARATIVE SYSTEM

The declarative system serves as a specification of the type inference algorithm. It consists of two main parts: the subtyping and the type inference.

3.1 Subtyping

It is represented by a set of inference rules shown in fig. 3.

$\boxed{\Gamma \vdash N \leqslant M}$ Negative subtyping

$$\frac{}{\Gamma \vdash \alpha^- \leqslant \alpha^-}\ (\text{Var}^{\leqslant}_{-})$$

$$\frac{\Gamma \vdash P \simeq^{\leqslant} Q}{\Gamma \vdash \uparrow P \leqslant \uparrow Q}\ (\uparrow^{\leqslant})$$

$$\frac{\Gamma \vdash P \geqslant Q \quad \Gamma \vdash N \leqslant M}{\Gamma \vdash P \to N \leqslant Q \to M}\ (\to^{\leqslant})$$

$$\frac{\Gamma, \overrightarrow{\beta^+} \vdash \sigma : \overrightarrow{\alpha^+} \quad \Gamma, \overrightarrow{\beta^+} \vdash [\sigma]N \leqslant M}{\Gamma \vdash \forall \overrightarrow{\alpha^+}.\, N \leqslant \forall \overrightarrow{\beta^+}.\, M}\ (\forall^{\leqslant})$$

$\boxed{\Gamma \vdash N \simeq^{\leqslant} M}$ Negative equivalence

$$\frac{\Gamma \vdash N \leqslant M \quad \Gamma \vdash M \leqslant N}{\Gamma \vdash N \simeq^{\leqslant} M}\ (\simeq^{\leqslant}_{-})$$

$\boxed{\Gamma \vdash P \geqslant Q}$ Positive supertyping

$$\frac{}{\Gamma \vdash \alpha^+ \geqslant \alpha^+}\ (\text{Var}^{\geqslant}_{+})$$

$$\frac{\Gamma \vdash N \simeq^{\leqslant} M}{\Gamma \vdash \downarrow N \geqslant \downarrow M}\ (\downarrow^{\geqslant})$$

$$\frac{\Gamma, \overrightarrow{\beta^-} \vdash \sigma : \overrightarrow{\alpha^-} \quad \Gamma, \overrightarrow{\beta^-} \vdash [\sigma]P \geqslant Q}{\Gamma \vdash \exists \overrightarrow{\alpha^-}.\, P \geqslant \exists \overrightarrow{\beta^-}.\, Q}\ (\exists^{\geqslant})$$

$\boxed{\Gamma \vdash P \simeq^{\leqslant} Q}$ Positive equivalence

$$\frac{\Gamma \vdash P \geqslant Q \quad \Gamma \vdash Q \geqslant P}{\Gamma \vdash P \simeq^{\leqslant} Q}\ (\simeq^{\leqslant}_{+})$$

Fig. 3. Declarative Subtyping

Invariance of Shifts. An important restriction that we put on the subtyping system is that the subtyping on shifted types requires their equivalence, as shown in $(\downarrow^{\geqslant})$ and $(\uparrow^{\leqslant})$. Relaxing both of these invariants makes the system equivalent to System F , and thus, undecidable. However, $(\downarrow^{\geqslant})$ might be relaxed to a covariant form, as we will discuss in **??**.

As discussed in **??**, the polymorphic rules $(\forall^{\leqslant})$ and $(\exists^{\geqslant})$ are the only non-algorithmic ones. For convenience of representation, we compose the left-hand side rule and the right-hand side rule into one, and use substitution σ to represent instantiation.

The substitution application is defined in a standard way, avoiding capture of bound variables, and preserving the variables that are out of the substitution domain. The domain and the range of the substitutions are specified by notation $\Gamma_2 \vdash \sigma : \Gamma_1$. For instance, the notation $\Gamma, \overrightarrow{\beta^+} \vdash \sigma : \overrightarrow{\alpha^+}$ means that σ maps the variables from $\overrightarrow{\alpha^+}$ to (positive) types well-formed in $\Gamma, \overrightarrow{\beta^+}$.

3.2 Type Inference

3.3 Properties of the Declarative System

Now, we present selected properties of the declarative system, which are important for the correctness of the algorithm.

Lemma 1. *Variables do not have proper subtypes and supertypes*

Lemma 2. *Free Variable propagation:* $\Gamma \vdash N \leqslant M$ *implies* $\mathbf{fv}\,(N) \subseteq \mathbf{fv}\,(M)$.

$\boxed{\Gamma;\Phi \vdash c\colon N}$ Negative typing

$$\frac{\Gamma \vdash P \quad \Gamma;\Phi, x : P \vdash c\colon N}{\Gamma;\Phi \vdash \lambda x : P.\ c\colon P \to N}\ (\lambda^{\text{INF}})$$

$$\frac{\Gamma, \alpha^+;\Phi \vdash c\colon N}{\Gamma;\Phi \vdash \Lambda\alpha^+.\ c\colon \forall\alpha^+.\ N}\ (\Lambda^{\text{INF}})$$

$$\frac{\Gamma;\Phi \vdash v\colon P}{\Gamma;\Phi \vdash \mathbf{return}\ v\colon \uparrow P}\ (\text{RET}^{\text{INF}})$$

$$\frac{\Gamma;\Phi \vdash v\colon P \quad \Gamma;\Phi, x : P \vdash c\colon N}{\Gamma;\Phi \vdash \mathbf{let}\ x = v;\ c\colon N}\ (\text{LET}^{\text{INF}})$$

$$\frac{\Gamma;\Phi \vdash v\colon \downarrow M \quad \Gamma;\ \Phi \vdash M \bullet \vec{v} \Longrightarrow \uparrow Q \text{ unique} \quad \Gamma;\Phi, x : Q \vdash c\colon N}{\Gamma;\Phi \vdash \mathbf{let}\ x = v(\vec{v});\ c\colon N}\ (\text{LET}^{\text{INF}}_{@})$$

$$\frac{\Gamma \vdash P \quad \Gamma;\Phi \vdash v\colon \downarrow M \quad \Gamma;\ \Phi \vdash M \bullet \vec{v} \Longrightarrow \uparrow Q \quad \Gamma \vdash \uparrow Q \leqslant \uparrow P \quad \Gamma;\Phi, x : P \vdash c\colon N}{\Gamma;\Phi \vdash \mathbf{let}\ x : P = v(\vec{v});\ c\colon N}\ (\text{LET}^{\text{INF}}_{:@})$$

$$\frac{\Gamma;\Phi \vdash v\colon \exists\overrightarrow{\alpha}.\ P \quad \mathbf{nf}\ (\exists\overrightarrow{\alpha}.\ P) = \exists\overrightarrow{\alpha}.\ P \quad \Gamma, \overrightarrow{\alpha};\Phi, x : P \vdash c\colon N \quad \Gamma \vdash N}{\Gamma;\Phi \vdash \mathbf{let}^{\exists}(\overrightarrow{\alpha}, x) = v;\ c\colon N}\ (\text{LET}^{\text{INF}}_{\exists})$$

$$\frac{\Gamma \vdash M \quad \Gamma;\Phi \vdash c\colon N \quad \Gamma \vdash N \leqslant M}{\Gamma;\Phi \vdash (c : M)\colon M}\ (\text{ANN}^{\text{INF}}_{-})$$

$$\frac{\Gamma;\Phi \vdash c\colon N \quad \Gamma \vdash N \simeq^{\leqslant} N'}{\Gamma;\Phi \vdash c\colon N'}\ (\simeq^{\text{INF}}_{-})$$

$\boxed{\Gamma;\Phi \vdash v\colon P}$ Positive typing

$$\frac{x : P \in \Phi}{\Gamma;\Phi \vdash x\colon P}\ (\text{VAR}^{\text{INF}})$$

$$\frac{\Gamma;\Phi \vdash c\colon N}{\Gamma;\Phi \vdash \{c\}\colon \downarrow N}\ (\{\}^{\text{INF}})$$

$$\frac{\Gamma \vdash Q \quad \Gamma;\Phi \vdash v\colon P \quad \Gamma \vdash Q \geqslant P}{\Gamma;\Phi \vdash (v : Q)\colon Q}\ (\text{ANN}^{\text{INF}}_{+})$$

$$\frac{\Gamma;\Phi \vdash v\colon P \quad \Gamma \vdash P \simeq^{\leqslant} P'}{\Gamma;\Phi \vdash v\colon P'}\ (\simeq^{\text{INF}}_{+})$$

$\boxed{\Gamma;\ \Phi \vdash N \bullet \vec{v} \Longrightarrow M}$ Application typing

$$\frac{\Gamma \vdash N \simeq^{\leqslant} N'}{\Gamma;\ \Phi \vdash N \bullet \cdot \Longrightarrow N'}\ (\emptyset^{\text{INF}}_{\bullet\Rightarrow})$$

$$\frac{\Gamma;\Phi \vdash v\colon P \quad \Gamma \vdash Q \geqslant P \quad \Gamma;\ \Phi \vdash N \bullet \vec{v} \Longrightarrow M}{\Gamma;\ \Phi \vdash Q \to N \bullet v, \vec{v} \Longrightarrow M}\ (\to^{\text{INF}}_{\bullet\Rightarrow})$$

$$\frac{\vec{v} \neq \cdot \quad \overrightarrow{\alpha^+} \neq \cdot \quad \Gamma \vdash \sigma : \overrightarrow{\alpha^+} \quad \Gamma;\ \Phi \vdash [\sigma]N \bullet \vec{v} \Longrightarrow M}{\Gamma;\ \Phi \vdash \forall\overrightarrow{\alpha^+}.\ N \bullet \vec{v} \Longrightarrow M}\ (\forall^{\text{INF}}_{\bullet\Rightarrow})$$

Fig. 4. Declarative Subtyping

Lemma 3. *Subtyping is Reflexive and Transitive, and is preserved by substitution*

Lemma 4. $\Gamma \vdash N \simeq^{\leqslant} M$ *is equivalent to* $\mathbf{nf}\ (N) \simeq^{D} \mathbf{nf}\ (M)$.

Lemma 5 (Characterization of Supertypes).

4 THE ALGORITHM

5 CORRECTNESS

6 EXTENSIONS

7 RELATED WORK

8 CONCLUSION

[Botlan et al. 2003] [Dunfield et al. 2020]

REFERENCES

Didier Le Botlan and Didier Rémy (Aug. 2003). "MLF Raising ML to the Power of System F." In: *ICFP '03*. Uppsala, Sweden: ACM Press, pp. 52–63.

Jana Dunfield and Neel Krishnaswami (Nov. 2020). "Bidirectional Typing." In: arXiv: 1908.05839.

Paul Blain Levy (Dec. 2006). "Call-by-Push-Value: Decomposing Call-by-Value and Call-by-Name." In: *Higher-Order and Symbolic Computation* 19.4, pp. 377–414. DOI: 10.1007/s10990-006-0480-6.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009