

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 (\exists) and polymorphic (\forall) 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), 3 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 INTRODUCTION

2 OVERVIEW

2.1 The Language

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 \exists .
- a function $P \rightarrow N$ takes a value as input and returns a computation;
- a polymorphic abstraction $\forall \vec{\alpha}^+. N$ quantifies a computation over a list of positive type variables $\vec{\alpha}^+$. 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 \forall .
- + $\exists \vec{\alpha}^+. P$, symmetrically to \forall , binds negative variables in a positive type P .
- + a shift $\downarrow N$, symmetrically to the up-shift, thunk a computation, which at the term-level corresponds to $\{c\}$.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Association for Computing Machinery.

0004-5411/2018/8-ART111 \$15.00

<https://doi.org/XXXXXXX.XXXXXXX>

Negative declarative types

N, M, K	$::=$
	α^-
	$\uparrow P$
	$P \rightarrow N$
	$\forall \vec{\alpha}^+. N$

Positive declarative types

P, Q, R	$::=$
	α^+
	$\downarrow N$
	$\exists \vec{\alpha}^+. P$

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 $\forall \vec{\alpha}^+. \forall \vec{\beta}^+. N$ with $\forall \vec{\alpha}^+, \vec{\beta}^+. N$, as well as $\exists \vec{\alpha}^+. \exists \vec{\beta}^+. P$ with $\exists \vec{\alpha}^+, \vec{\beta}^+. P$, and lift these equations transitively and congruently to the whole system.

3 DECLARATIVE SYSTEM**4 ALGORITHM****5 PROOF****6 EXTENSIONS****7 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

99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147