Type Inference for Stack-based Languages
Annotated Bibliography

Robert Kleffner

Northeastern University

J. Poial. Algebraic Specification of Stack Effects for Forth Programs. In *1990 FORML Conference Proceedings*, pages 12–14. The Forth Interest Group, 1990

This paper marks the beginning of interest in type inference for stack-based languages. While it does not call itself a type system, the stack effect notation that Poial formalizes has essentially the function same as a type system: it is a way to automatically check that a program does not get into 'stuck' states or try to underflow the stack.

J. Poial. Multiple Stack Effects of Forth Programs. In *1991 FORML Conference Proceedings, euroFORML*, pages 11–13, 1991

A continuation of the system explored in Poial's '90 paper, extended with finite intersections of stack-effects to allow the results of branches in if-statements to be different.

B. Stoddart and P. J. Knaggs. Type inference in stack based languages. *Formal Aspects of Computing*, 5(4):289–298, July 1993

Following Poial's work, Stoddart & Knaggs design a type inference algorithm based on the stack effect notation. Their notable contribution is the addition of polymorphic type variables similar to those in standard Hindley-Milner languages. They do not explore let-polymorphism.

C. Okasaki. Techniques for embedding postfix languages in Haskell. *Proceedings of the ACM SIGPLAN workshop on Haskell - Haskell '02*, pages 105–113, 2002

Okasaki, who does not cite any of the previous research and was likely unaware of its existence, unintentionally extends the power of type inference in stack-languages by embedding a stack language in Haskell. His embedding relies on polymorphic nested tuples at the type level, and can infer the types of higher order functions, something the previous systems could not do.

C. Diggins. Simple Type Inference for Higher-Order Stack-Oriented Languages, 2008

Diggins applies the ideas developed by Okasaki to a stack language inspired by Joy, and details a type inference algorithm for the language. His type syntax makes a distinction between stack type variables and individual type variables, so he does not require Okasaki's nested tuples, merging the syntactic cleanliness of stack-effects with the expressive power of Okasaki's system. However, he claims no theorems of any kind.