Typed Genetic Programming over Lambda Calculus

T. Křen

Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic.

Abstract. Todo.

Introduction

Related work

Yu [2001] presents a GP system utilizing polymorphic higher-order functions[1] and lambda abstractions. Important point of interest in this work is use of `foldr`[2] function as a tool for *implicit recursion*, i.e. recursion without explicit recursive calls. The terminal set for constructing lambda abstraction subtrees is limited to use only constants and variables of that particular lambda abstraction, i.e., outer variables are not allowed to be used as terminals in this work. This is significant difference from our approach since we permit all well-typed normalized λ-terms. From this difference also comes different crossover operation. We focus more on term generating process; their term generation is performed in a similar way as the standard one, whereas our term generation also tries to utilize techniques of systematic enumeration.

Briggs and O'Neill [2008] present technique utilizing typed GP with combinators. The difference between approach presented in this work and our approach is that in this work terms are generated straight from *library* of combinators and no lambda abstractions are used. They are using more general polymorphic type system than us – the HindleyMilner type system. They also discuss the properties of exhaustive enumeration of terms and compare it with GP search. They also present interesting concept of *Generalized genetic operator* based on term generation.

Binard and Felty [2008] use even stronger type system (*System F*). But with increasing power of the type system comes increasing difficulty of term generation. For this reason evolution in this work takes interesting and nonstandard shape (fitness is associated with *genes* which are evolved together with *species* which together participate in creation of individuals). This differs from our approach, which tries to be generalization of the standard GPKoza [1992].

In contrast with above mentioned works our approach uses very simple type system (simply typed lambda calculus) and concentrates on process of generation able to generate all possible well-typed normalized lambda terms. In order to do so we use technique based on *inhabitation machines* described by Barendregt Barendregt et al. [2010].

Todo.

Conclusion

Todo.

Acknowledgments. ???

References

Barendregt, H., Dekkers, W., and Statman, R., *Lambda Calculus With Types*, Cambridge University Press, 2010.

---

[1] Higher-order function takes another function as an input parameter.

[2] In the functional programming language Haskell `foldr` can be defined as:

```
foldr f z []     = z
foldr f z (x:xs) = f x (foldr f z xs)
```

:

Binard, F. and Felty, A., Genetic programming with polymorphic types and higher-order functions, in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pp. 1187–1194, ACM, 2008.

Briggs, F. and O'Neill, M., Functional genetic programming and exhaustive program search with combinator expressions, *International Journal of Knowledge-Based and Intelligent Engineering Systems*, *12*, 47–68, URL `http://iospress.metapress.com/content/u614j13p67w66370/`, 2008.

Koza, J. R., *Genetic programming: on the programming of computers by means of natural selection*, MIT Press, Cambridge, MA, USA, 1992.

Peyton Jones, S. L., *The Implementation of Functional Programming Languages (Prentice-Hall International Series in Computer Science)*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987.

Yu, T., Hierarchical processing for evolving recursive and modular programs using higher-order functions and lambda abstraction, *Genetic Programming and Evolvable Machines*, *2*, 345–380, 2001.