1 Related work

In [2] Yu presents a GP system utilizing polymorphic higher-order functions[1] and lambda abstractions. Important point of interest in this work is use of `foldr` 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.

In [4] Briggs and ONeill 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 then 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.

In [3] by Binard and Felty even stronger type system (*System F*) is used. 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 GP[1].

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 in [5].

References

[1] John R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[2] T. Yu. *Hierachical processing for evolving recursive and modular programs using higher order functions and lambda abstractions*. Genetic Programming and Evolvable Machines, 2(4):345380, December 2001. ISSN 1389-2576.

[3] Binard, F., Felty, A.: Genetic programming with polymorphic types and higher-order functions. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation, ACM (2008) 1187-1194

[1] Higher-order function is a function taking another function as input parameter.

[4] Forrest Briggs, Melissa ONeill. *Functional Genetic Programming and Exhaustive Program Search with Combinator Expressions.* International Journal of Knowledge-based and Intelligent Engineering Systems, Volume 12 Issue 1, Pages 47-68, January 2008.

[5] Henk Barendregt, Wil Dekkers, Richard Statman, *Lambda Calculus With Types*. Cambridge University Press, 2010.