An Encoding Scheme for Generating λ-Expressions in Genetic Programming

Kazuto Tominaga, Tomoya Suzuki, and Kazuhiro Oka

Tokyo University of Technology, Hachioji, Tokyo 192-0982 Japan
tomi@acm.org, {tomoya,oka}@ns.it.teu.ac.jp

Abstract. To apply genetic programming (GP) to evolve λ-expressions, we devised an encoding scheme that encodes λ-expressions into trees. This encoding has closure property, i.e., any combination of terminal and non-terminal symbols forms a valid λ-expression. We applied this encoding to a simple symbolic regression problem over Church numerals and the objective function $f(x) = 2x$ was successfully obtained. This encoding scheme will provide a good foothold for exploring fundamental properties of GP by making use of lambda-calculus.

In this paper, we give an encoding scheme that encodes λ-expressions into trees, which is appropriate for evolving λ-expressions by genetic programming (GP). A λ-expression is a term in lambda-calculus [2], composed of *variables*, *abstraction* and *application*. Here are some examples of λ-expressions.

$(\lambda x.(\lambda y.(x(xy))))$ (let *2* denote this λ-expression)
$(\lambda x.(\lambda y.(\lambda z.((xy)(xy)z))))$ (let D denote this λ-expression)

A λ-expression can be regarded as a program. Giving the input *2* to the program D is represented by applying D to *2* as the following.

$$(D\ \ 2) \equiv ((\lambda x.(\lambda y.(\lambda z.((xy)(xy)z))))(\lambda x.(\lambda y.(x(xy)))))$$

By several steps of β-reduction, the above λ-expression is rewritten into

$$(\lambda x.(\lambda y.(x(x(x(xy)))))) \quad . \quad \text{(let } 4 \text{ denote this } \lambda\text{-expression)}$$

β-reduction steps can be regarded as the execution of the program, and *4* can be regarded as the output of the execution. If λ-expressions can be represented as trees, GP can evolve λ-expressions.

We devised an encoding scheme to handle λ-expressions in GP. This encoding scheme encodes a λ-expression into a tree. The non-terminal symbol `L` represents abstraction, and the non-terminal symbol `P` represents application. A terminal symbol is a positive integer and it represents a variable. A variable labeled by an integer n is bound by n-th abstraction in the path from the variable to the root. Example trees are `(L 1)`, `(L (L 1))` and `(L (L (P 2 1)))`, which represent λ-expressions $(\lambda x.x)$, $(\lambda x.(\lambda y.y))$ and $(\lambda x.(\lambda y.(xy)))$, respectively.

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2724, pp. 1814–1815, 2003.

This encoding scheme has the following good properties. One is that any tree encoded by this scheme is syntactically and semantically valid as a λ-expression. Crossover and mutation always produce valid individuals. Secondly the meaning of a subtree does not change by crossover since relation of a bound variable and the abstraction that binds the variable is preserved under crossover.

We applied this encoding to a simple symbolic regression problem. The objective function is $f(x) = 2x$ over Church numerals [2]. Church numerals represent natural numbers; examples are *2* and *4* shown above. The function f can be expressed as a λ-expression in many ways, and D is such a λ-expression. We implemented a β-reduction routine in C and incorporated it to lil-gp [3] to make the whole GP system. Main GP parameters are: population size 1000; maximum generation 1000; 10 % reproduction with elites 1 %, 89.9 % crossover and 0.1 % mutation. In the experiment, the objective function D was obtained in 13 out of 25 runs. The transition of the best individuals in a successful run is shown in Table 1 in their canonical forms. The fitness was measured based on structural difference between the correct answer and the output of the individual.

The experimental results indicate that the encoding scheme is appropriate for evolving λ-expressions by GP, and thus it will provide a good foothold for exploring fundamental property of GP by using lambda-calculus.

Table 1. Example transition of best individuals

Generation	Fitness	Best individual
0	153	$(\lambda x.x)$
233	150	$(\lambda x.(\lambda y.(\lambda z.(z((xy)(yy))))))$
346	147	$(\lambda x.(\lambda y.(\lambda z.(z((xy)(y(z(yy)))))))$
365	146	$(\lambda x.(\lambda y.(\lambda z.(z((xy)(y(y(yy)))))))$
376	145	$(\lambda x.(\lambda y.(\lambda z.(z((xy)(y(y(y(zy))))))))$
401	143	$(\lambda x.(\lambda y.(\lambda z.(z((xy)(y(y(y(z(yz))))))))$
403	82	$(\lambda x.(\lambda y.(\lambda z.(z((xy)(y(y((xz)(\lambda w.w))))))))$
414	81	$(\lambda x.(\lambda y.(\lambda z.(z((xy)(y((xz)(\lambda w.w))))))))$
420	80	$(\lambda x.(\lambda y.(\lambda z.((xy)(y(y((xz)(\lambda w.w))))))))$
422	9	$(\lambda x.(\lambda y.(\lambda z.(z((xy)(y(y((xy)(\lambda w.w))))))))$
425	6	$(\lambda x.(\lambda y.(\lambda z.((xy)(y(y((xy)(\lambda w.w))))))))$
433	2	$(\lambda x.(\lambda y.(\lambda z.((xy)((xy)(\lambda w.w))))))$
439	0	$(\lambda x.(\lambda y.(\lambda z.((xy)((xy)z)))))$

References

1. Koza, J.: Genetic Programming. MIT Press (1992)
2. Revesz, G.: Lambda-Calculus, Combinators, and Functional Programming. Cambridge University Press (1988)
3. lil-gp Genetic Programming System. `http://garage.cps.msu.edu/software/lil-gp/lilgp-index.html`