COSC 4P77 Evolutionary Computation B. Ross
(last modified Oct 26, 2005)

Notes on lilGP and its extended versions

1. Getting started

Download and install a version of lilGP suitable for your computer and OS. The COSC 4P77 web site has links to various versions. Some versions available include:

- *lilGP 1.1*: The original lilGP, which is readily compilable on Linux.
- *Patched lilGP 1.1*: This has some bug fixes and extensions. Most notably, it supports strong typing, so that your language can be organized with respect to data types.
- *BST lilGP*: This is an enhanced version of the Patched lilGP system. There are versions available for Linux, VS.NET and CodeWarrior.

Also download the lilGP manual, and Patch documentation (information about the strong typing scheme).

2. Main files

User-supplied files

- app.c, app.h
    - GA-related functions, including language table, program initialization, fitness evaluation, program termination, and statistical output.
    - The language table defines function calls (in function.c), argument information, and typing information (for patched lilGP and BST lilGP)
- appdef.h
    - misc definitions
- function.c, function.h
    - implementation of the language terminals and nonterminals
- input.file
    - this contains the user parameters for the experiment

Output files (by extension)

- bst: contains best performing solution for the run
- his: best per generation
- stt: various statistics (popn fitness, best fitness, ...)

3. Setting up language in app.c, function.c

In app.c, the function *app_build_function_sets* contains the terminal and nonterminal information needed by the GP system. The table in this function needs to be carefully coded, or else errors will result. The kinds of functions you define in the table are:

- data nonterminals (FUNC_DATA): the argments to the function are pre-evaluated before the function code is executed

- expression nonterminals (FUNC_EXPR): the function code will explicitly evaluate arguments as necessary. Used when efficiency required (eg. boolean expressions) or unwanted side-effects are to be avoided (eg. ant in maze)
- terminals (TERM_NORM): ordinary terminals have no arguments
- ephemeral random constants (TERM_ERC): these terminals are handled somewhat specially, as they are "executed" only when created, but then retain their value afterwards

Other information in this table refers to ADF's (modules), sub-populations (for pseudo-parallelism), and strong typing (for the patched and BSTlilgp versions). Be sure to follow the rest of the conventions in this function (eg. make sure fset.size is correct!).

One special rule for the strong typing versions: You need to supply at least one terminal and nonterminal for every type category in the language. For example, if you have boolean operators, you will also need a boolean nonterminal (True and false perhaps).

The other functions in app.c are callbacks, which you can use to put your own code for fitness evaluation, initialization, termination, testing, etc. Please look at the regression and ant applications for examples of how this code can be set up. You will need access many data structures used by the lilGP kernel. To do this, please follow the code used in the example applications. The lilGP documentation has further information.

You will define the C implementation of your terminals and nonterminals specified in the language table in app.c, within the file function.c. These functions are often very small. Be sure to follow the data structure conventions exactly as given in the example applications. Also be sure that your functions respect the closure condition, so that they execute error-free on any possible value handed to them.