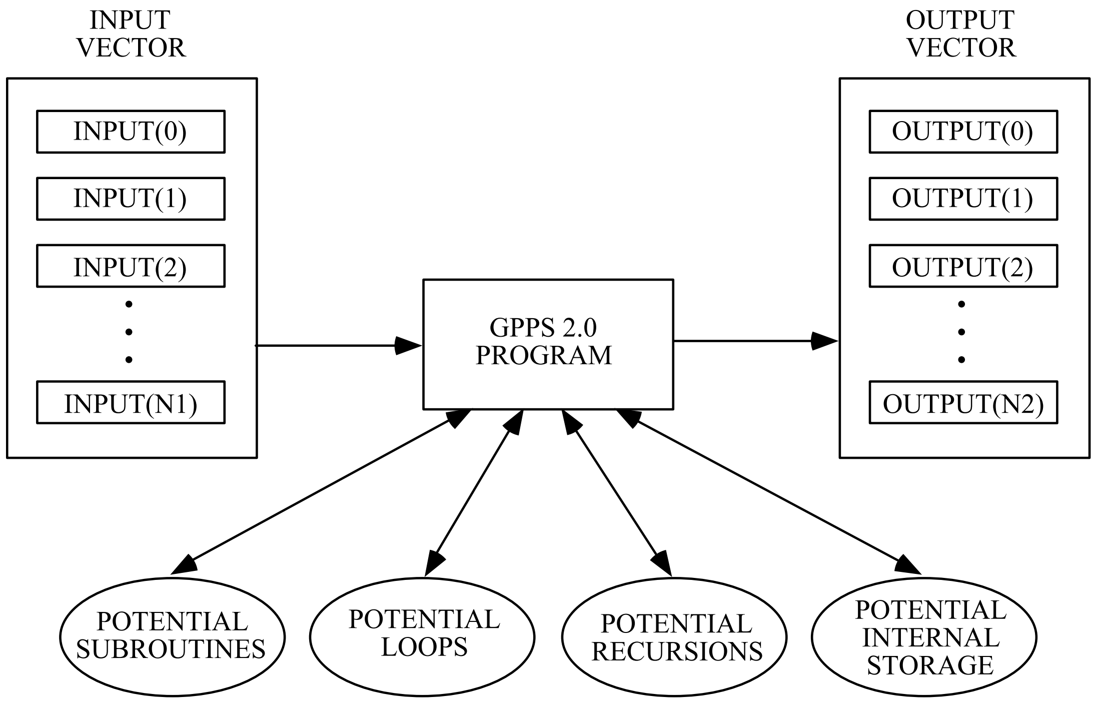
GENETIC PROGRAMMING PROBLEM SOLVER (GPPS)

GENETIC PROGRAMMING PROBLEM SOLVER (GPPS) — VERSION 2.0



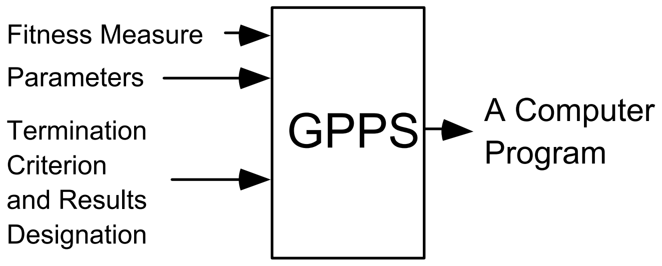
GENETIC PROGRAMMING PROBLEM SOLVER (GPPS)

- **GPPS provides a general-purpose method for automatically creating computer programs that solve, or approximately solve, problems.**
- **GPPS uses a standardized set of functions and terminals and thereby eliminates the need for the user to prespecify a function set and terminal set for the problem.**
- **GPPS uses architecture-altering operations to create, duplicate, and delete subroutines and loops (and, in GPPS 2.0, recursions and internal storage) during the run. Thus, GPPS eliminates the need for the user to specify in advance whether to employ subroutines, loops, recursions, and internal storage in solving a given problem. GPPS similarly eliminates the need for the user to specify the number of arguments possessed by each subroutine.**

GPPS — OBSERVATIONS

- **The four arithmetic functions of addition, subtraction, multiplication, and division and a conditional branching operator (e.g., the three-argument "If Greater Than Zero" `IFGTZ`) constitute the core of the repertoire of primitive machine code instructions for virtually every general-purpose computer.**
- **Floating-point numbers can represent Boolean-valued variables, integer-valued variables, and floating-point variables**
- **Inputs to the to-be-evolved computer programs can, without loss of generality and without a significant sacrifice in convenience, be received in an input vector.**
- **Outputs can be similarly handled by means of an output vector.**
- **Problem-specific side-effecting functions (e.g., robotic control functions) can be handled, without loss of generality, by means of an output interface that interprets the numbers produced in the output vector.**

THREE PREPARATORY STEPS WITH GENETIC PROGRAMMING PROBLEM SOLVER (GPPS)



THREE PREPARATORY STEPS — CONTINUED

- **When GPPS is used to solve a problem, the program architecture, the function set, and the terminal set do not change from problem to problem.**
  - **That is, the first and second preparatory steps of genetic programming are eliminated.**
- **Thus, there are only three problem-specific preparatory steps with GPPS, namely determining the fitness measure, determining the run's control parameters and determining the termination criterion and the method of result designation.**
- **GPPS makes it especially clear that the determination of the fitness measure (the third major preparatory step of genetic programming) is, as a general rule, the most important preparatory step in applying genetic programming to a problem.**

GENETIC PROGRAMMING PROBLEM SOLVER (GPPS 2.0)

Objective:	**Create, using the Genetic Programming Problem Solver 2.0, a computer program that takes the values of one independent floating-point variable *x* in the input vector and deposits a value into the output vector that closely matches the quadratic polynomial $2.718x^2 + 3.1416x$.**

Program architecture:	**One result-producing branch, `RPB`. Automatically defined loops, automatically defined recursions, automatically defined stores, and automatically defined function(s) and their arguments will be created during the run by the architecture-altering operations.**
Initial function set for the result-producing branches:	$\mathsf{F}_{\text{rpb-initial}}$ **= {`+`, `–`, `*`, %, `IFLTE`, `TOR`, `TAND`, `TNOT`, `RLI`, `WLO`, `FLOOR`}.**
Initial terminal set for the result-producing branches:	$\mathsf{T}_{\text{rpb-initial}}$ **= {$\leftarrow_{\text{bigger-reals}}$, `NINPUTS`, `NOUTPUTS`, `INDEX`}.**

Initial function set for the automatically defined functions:	**No automatically defined functions in generation 0. $F_{adf\text{-}initial} = \phi$.**
Initial terminal set for the automatically defined functions:	**No automatically defined functions in generation 0. $T_{adf\text{-}initial} = \phi$.**
Initial function set for automatically defined loops:	**No automatically defined loops in generation 0. $F_{adl\text{-}initial} = \phi$.**
Initial terminal set for automatically defined loops:	**No automatically defined loops in generation 0. $T_{adl\text{-}initial} = \phi$.**
Initial function set for automatically defined recursions:	**No automatically defined recursions in generation 0. $F_{adr\text{-}initial} = \phi$.**

Initial terminal set for automatically defined recursions:	**No automatically defined recursions in generation 0. $\mathtt{F}_{\text{adr-initial}} = \phi$.**
Potential function set for the result-producing branches:	**$\mathtt{F}_{\text{rpb-potential}}$ = {`ADL0`, `ADR0`, `SWB0`, `SWB1`, `ADF0`, `ADF1`, `ADF2`, `ADF3`}.**
Potential terminal set for the result-producing branches:	**$\mathtt{T}_{\text{rpb-potential}}$ = {`LBB0`, `SRB0`, `SRB1`}.**
Potential function set for the automatically defined functions:	**$\mathtt{F}_{\text{adf-potential}}$ = {`ADF0`, `ADF1`, `ADF2`, `ADF3`}.**
Potential terminal set for the automatically defined functions:	**$\mathtt{T}_{\text{adf-potential}}$ = {`ARG0`, `ARG1`, `NINPUTS`, `NOUTPUTS`, `INDEX`, $\leftarrow_{\text{bigger-reals}}$}.**
Potential function set for automatically defined loops:	**$\mathtt{F}_{\text{adl-potential}}$ = {`ADL0`, `ADL1`, `ADL2`, `ADL3`}**

Potential terminal set for automatically defined loops:	$T_{\text{adl-potential}}$ = {**NINPUTS**, **NOUTPUTS**, **INDEX**, $\leftarrow_{\text{bigger-reals}}$}.
Potential function set for automatically defined recursions:	$F_{\text{adr-potential}}$ = {**ADR0**, **ADR0**, **ADR1**, **ADR2**, **ADR3**}.
Potential terminal set for automatically defined recursions:	$T_{\text{adr-potential}}$ = {**NINPUTS**, **NOUTPUTS**, **INDEX**, $\leftarrow_{\text{bigger-reals}}$}.
Fitness cases:	**20 randomly chosen values of the independent variable *x* between –1.0 and +1.0.**

Raw fitness:	**Raw fitness is the sum, over the 20 fitness cases, of the absolute value of the difference between the value deposited in the output vector and the value of the quadratic polynomial $2.718x^2 + 3.1416x$.**
Standardized fitness:	**Same as raw fitness**
Hits:	**The number of fitness cases (0 to 20) for which the value deposited in the output vector is within 0.01 of the value of the quadratic polynomial $2.718x^2 + 3.1416x$.**
Wrapper:	**None.**

Parameters:	***M*** **= 120,000.** ***G*** **= 1,001.** **`NINPUTS`** **= 1.** **`NOUTPUTS`** **= 1.** ***Q*** **= 2,000.** ***D*** **= 60.** ***B*** **= 2%.** N_{rpb} **= 1.** S_{rpb} **= 500.** S_{adf} **= 100.** $N_{\text{max-adf}}$ **= 4.** $N_{\text{max-argument-adf}}$ **= 2.** $N_{\text{min-adf-arg}}$ **= 0.** $N_{\text{max-adl}}$ **= 1.** S_{adl} **= 100.** $N_{\text{max-argument-adl}}$ **= 0.** $N_{\text{min-argument-adl}}$ **= 0.** $N_{\text{max-adl-executions}}$ **= 3.** $N_{\text{max-adr}}$ **= 1.** S_{adr} **= 100.** $N_{\text{max-argument-adr}}$ **= 0.** $N_{\text{min-argument-adr}}$ **= 0.** $N_{\text{max-adr-executions}}$ **= 9.** $N_{\text{max-ads}}$ **= 2.**
Result designation:	**Best-so-far pace-setting individual.**
Success predicate:	**A program scores 20 hits.**

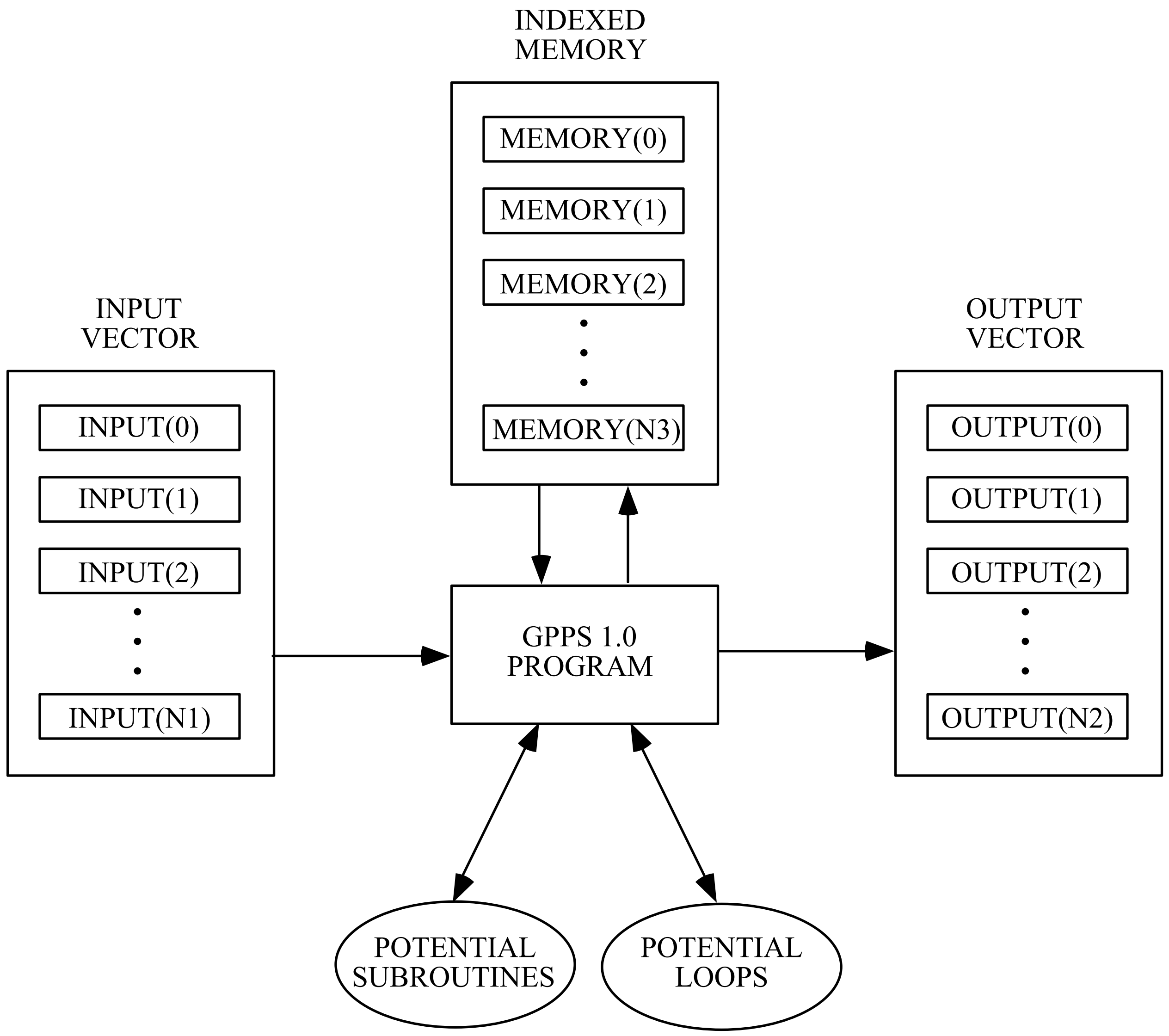
GPPS 1.0 — OVERALL CAPABILITIES

- **GPPS 1.0 is capable of automatically creating computer programs with**
  - **various numbers of inputs,**
  - **various numbers of outputs,**
  - **a main result-producing branch consisting of a to-be-evolved sequence of steps,**
  - **to-be-evolved numbers of automatically defined functions (ADFs) each possessing**
    - **a to-be-evolved number of arguments and**
    - **a to-be-evolved sequence of steps, and**
  - **to-be-evolved numbers of automatically defined loops (ADLs) each consisting of**
    - **a loop initialization branch consisting of a to-be-evolved sequence of steps,**
    - **a loop condition branch consisting of a to-be-evolved sequence of steps,**
    - **a loop body branch consisting of a to-be-evolved sequence of steps, and**
    - **a loop update branch consisting of a to-be-evolved sequence of steps, and**
  - **a fixed number of cells of indexed memory.**

GPPS 2.0 — OVERALL CAPABILITIES

- **GPPS 2.0 has the additional capabilities of handling programs with**
  - **an initially-unspecified number of automatically defined recursions (with each ADR consisting of a recursion condition branch, a recursion body branch, a recursion update branch, and a recursion ground branch), and**
  - **an initially-unspecified amount and type of internal storage as implemented by automatically defined stores (ADSs).**

FLOW OF INFORMATION IN A COMPUTER PROGRAM USING GPPS 1.0



FUNCTIONS IN GPPS 1.0

- **GPPS 1.0 contains the following functions:**
  - **arithmetic functions**
    - **addition (`+`),**
    - **subtraction (`-`),**
    - **multiplication (`*`),**
    - **protected division `%`,**
  - **conditional branching operators**
    - **"If Greater Than Zero" `IFGTZ`,**
    - **"If Equal Zero" `IFEQZ`,**
  - **numerically valued logical functions**
    - **conjunction `TAND`,**
    - **disjunction `TOR`,**
    - **negation `TNOT`,**
  - **input reading function**
    - **read linear input `RLI`,**
  - **writing and reading functions for indexed memory**
    - **write indexed memory `WIM`,**
    - **read indexed memory `RIM`,**
  - **output writing and reading functions**
    - **write linear output `WLO`,**
    - **read linear output `RLO`,**
  - **conversion function**
    - **`FLOOR`,**

TERMINALS IN GPPS 1.0

- **GPPS 1.0 contains the following terminals:**
  - **potential automatically defined functions such as**
    - **`ADF0`,**
    - **`ADF1`,**
    - **`ADF2`,**
    - **`ADF3`,**
  - **potential terminals representing the dummy variables (formal parameters) of the potential automatically defined functions such as**
    - **`ARG0`,**
    - **`ARG1`,**
    - **`ARG2`,**
    - **`ARG3`,**
  - **potential terminals representing the return value of the loop body branch of each potential automatically defined loop, such as**
    - **`LBB0`**
    - **`LBB1`**
  - **terminals**
    - **floating-point random constants,**
    - **a constant specifying the number of inputs, `NINPUTS`**
    - **a constant specifying the number of outputs, `NOUTPUTS`**
    - **the loop index, `INDEX`**

EXPLANATION OF TERMINALS IN GPPS 1.0

`NINPUTS` is an externally established, invariant terminal that specifies the number of input(s) for the problem in the input vector.

`NOUTPUTS` is an externally established, invariant terminal that specifies the number of output(s) for the problem in the output vector.

`INDEX` is the loop index for automatically defined loops. It is externally initialized to zero prior to execution of a program. It remains zero if there are no automatically defined loops in the program. It is externally initialized to zero as the beginning of execution of each automatically loop. It is externally incremented by one after the end of each execution of a loop update branch. If it is referenced outside of an automatically defined loop, it returns its leftover value.

EXPLANATION OF FUNCTIONS IN GPPS 1.0

`IFEQZ` ("If Equal Zero") is the three-argument conditional branching operator that evaluates and returns its second argument if its first argument (the condition) is equal to zero, but otherwise evaluates and returns its third argument

`IFGTZ` ("If Greater Than Zero") conditional branching operator

`TAND` is the two-argument numerical-valued conjunctive function returning a floating-point +1.0 if both of its arguments are positive, but returning –1.0 otherwise. `TAND` is a short-circuiting (optimized) function in the sense that its second argument will not be evaluated (and any side-effecting function contained therein will remain unexecuted) if its first argument is negative.

EXPLANATION OF FUNCTIONS IN GPPS 1.0 — CONTINUED

`TOR` is the two-argument numerical-valued disjunctive function returning a floating-point +1.0 if one or both of its arguments is positive, but returning –1.0 otherwise. `TOR` is a short-circuiting (optimized) function in the sense that its second argument will not be evaluated (and any side-effecting function contained therein will remain unexecuted) if its first argument is positive.

`TNOT` is the one-argument numerical-valued negation function returning a floating-point +1.0 if its argument is negative, but returning –1.0 otherwise.
`RLI` ("Read Linear Input") is a one-argument function that returns the value of the element of the input vector specified by the argument. The argument is adjusted by flooring it and then taking it modulo the size (`NINPUTS`) of the input vector.

EXPLANATION OF FUNCTIONS IN GPPS 1.0 — CONTINUED

`WIM` ("Write Indexed Memory") is a two-argument function that writes the value returned by the first argument into the location of indexed memory specified by the second argument (adjusted in the same manner as above based on the size of indexed memory).

`RIM` ("Read Indexed Memory") is a one-argument function that returns the value of the element of the vector of indexed memory specified by the argument (adjusted in the same manner as above based on the size of the indexed memory).

EXPLANATION OF FUNCTIONS IN GPPS 1.0 — CONTINUED

`WLO` ("Write Linear Output") is a two-argument function that writes the value returned by the first argument into the location in the output vector specified by the second argument (adjusted in the same manner as above based on the size, `NOUTPUTS`, of the output vector).

`RLO` ("Read Linear Output") is a one-argument function that reads the location in the output vector specified by the argument (adjusted in the same manner as for `RLI`). This function enables the output vector to be used as an additional area of indexed memory.

`FLOOR` is the one-argument conversion function that floors its argument by reducing it to the next lower integer.

INITIALIZATION

- **All cells of indexed memory and all cells of the output vector are initialized to zero for each set of inputs (i.e., each fitness case).**
- **Note that if the fitness evaluation of a program requires that it be run through a series of time steps, the indexed memory nor the output vector are not initialized before each time step.**
- **For practical reasons, since the initialization, updating, and terminating of the iteration is controlled by branches that will be subject to vicissitudes of the evolutionary process, the total number of iterations that can be performed by any one iteration-performing branch is rationed.**