REPTIL - promoting dialog between humanoid and computer

Israel Urieli
Ohio University, Athens OHIO 45701

Traditional humanoid math education and experience which has been inherited over generations is diametrically opposed to the computer infrastructure and environment. Unfortunately the designers of the traditional procedural languages (such as FORTRAN, BASIC, Pascal, C) have found it necessary to emulate the questionable humanoid math heritage, and have thus provided a computational tool rather than a problem solving tool. REPTIL is an attempt to bridge the gap between the humanoid and computer approaches. The syntax and structure of REPTIL have been described elsewhere (1), and in this paper we give an example of the goal directed approach to solution which REPTIL promotes.

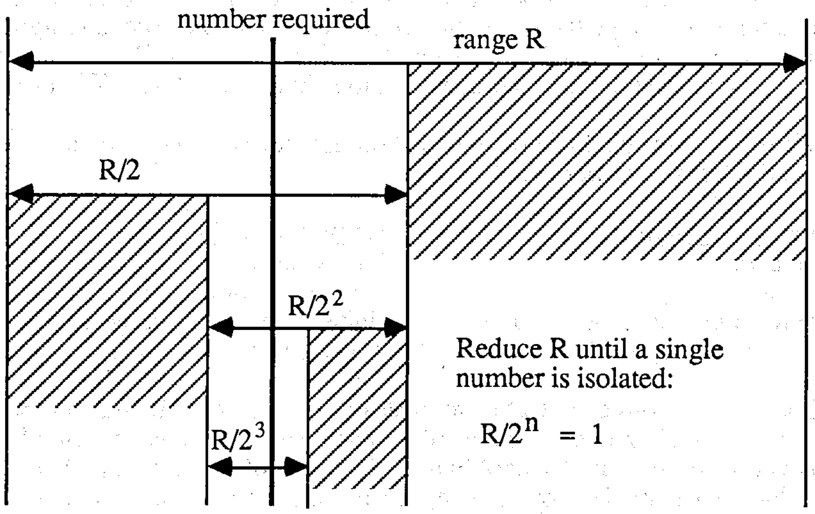
The main differences between the humanoid and computer math environments can be summarized in the following four counts:

	Humanoid environment	Computer environment
1.	decimal base	binary base
2.	infix notation	postfix notation
3.	infinite real number representation	finite integer number system
4.	explicit solution approach	goal directed recursive approach

The decimal base came from an unfortunate mistake by our sages who misinterpreted our thumb for a finger when in fact our creator meant each hand to be a hexadecimal digit. The infix notation is forced upon us at school in total disregard of the fact that when as infants we build with blocks we are developing an appreciation for stack manipulation. Thus any child will gleefully demonstrate that the top of stack allows orderly access of parameters and attempting to access the bottom of stack causes chaos. Presenting the world as an infinite continuum of floating point numbers is another distortion of the humanoid tradition. Dividing a pie among three hungry kids was fine until the math teacher said that each kid was really getting and infinite recurring decimal piece of pie and the computer said that the sum of the three portions was not equal to the whole. Instead of solving real problems, the humanoid mathematician is obsessed with evaluating irrational numbers, such as the value of π. The ancient Hebrews found $\pi = 3$ perfectly adequate for building the Second Temple, and using the rational fraction approximation 355/113 would allow one to locate Moscow within an accuracy of 11 feet. The fact that the computer sees the world as a set of discrete finite integers is a continuous cause of difficulty to the humanoids; they talk of truncation and roundoff errors, and cannot understand why when using a Taylor series expansion and double precision on an IBM 4341 the computer finds that sine(2910) = 49902.58337544370990 (actually it equals 0.5).

In this paper I wish to mainly concentrate on the fourth point, being the explicit solution versus the goal directed approach. Throughout school we are brainwashed with hundreds of problems of the type "If Johnny sells 5 apples at 10c each then how many bombs are required to destroy the humanoid species?" The traditional problem solving approach is thus to explicitly isolate the unknown in terms of some nebulous function of the various knowns and thus divorce the problem space from the solution space. Unfortunately this approach is continued through College, and we find typically that when the engineer has formulated the problem in terms of a differential equation set she then considers the problem solved and hands it over to the mathematician to obtain numerical solutions. The mathematician will then work in isolation devising all sorts of weird and wonderful schemes for controlling the accuracy and validity of the solution when possibly a simple physical constraint such as a conservation of mass could have avoided all that.

A concrete example is the problem of determining how many steps are required to find a telephone number in the directory. Apparently the humanoid instinctively understands that because the numbers are alphabetically ordered, a binary search is preferable to a sequential or random search; thus the small town bumpkin from Athens Ohio will not get flustered with the New York City directory, even though it is many orders of magnitude larger. The binary search will be somewhat similar to the diagram below:



I have presented this problem to many engineering students and they invariably come up with the explicit solution for n as:

$$n = \ln(R)/\ln(2)$$

where *ln* refers to the natural logarithm. We may try this gingerly on our computer with a small range of numbers, say 7, in order to validate it, thus:

$$n = \ln(7)/\ln(2) = 1.945910149/0.69314718 = 2.807354922$$

However since we have divorced the solution from the problem we have no idea what to do with this value. There is obviously an integer number of steps required hence should this value be rounded (= 3) or truncated (= 2)? Intuition will not help us here -- in our brainwashed zeal to find an explicit solution we have lost touch with reality.

Using REPTIL we would define a verb STEPS that uses a recursive or goal directed approach to solution. The range R is placed on the stack and checked to see if it is equal 1. If so then that is the stopping rule and no more steps are required (a zero is placed on the stack). If not then the range is divided by two and STEPS is invoked recursively. As we come out of each level of recursive nesting we add 1 to the number of steps. The verb definition follows:

```
<16>        'STEPS  DO:    \  R |P
1<16:>         DUP      \  R R |P
1<16:>         1        \  R R 1 |P
1<16:>         =?       \  R T/F |P  is range R on TOP = 1?
1<16:>         ?IFTRUE   \  stopping rule
2<16:?>           DROP 0  \  0 |P  no steps required if R=1
2<16:?>        ?ELSE     \   recursive step
2<16:?E>          2/     \   R/2 |P  bisect the range..
2<16:?E>          STEPS   \ ..and try again
2<16:?E>          1 +    \  one more step
2<16:?E>       ?ENDIF
1<16:>      :END
<16>
<16> DEC
<10>    7 STEPS =
2
<10>   20000 STEPS =  \ the Athens Ohio directory
14
<10>_
```

A few words of explanation are in order. The comments following the \ mainly document the Parameter stack manipulation, in which TOP is indicated by |P. The prompt presents three types of information, being the number of elements on the stack, the current base and any structured constructs opened which is used for both humanoid and computer nesting syntax checking. In this case the stack usage is by the system. Two of the four structured constructs of REPTIL are shown, the DO: :END defining verb set and the ?IFTRUE verb set. The DO: :END set is more powerful than Forth's colon definition and allows deferred nested definitions, thus it also replaces the CREATE DOES> of Forth. The ?IFTRUE verb set also includes ?IFFALSE and optional ?ELSEIF clauses.

Prefixing a name with a quote is used in some Forth-like languages, notably STOIC (2), PISTOL (3), and SPHERE (4). A unique aspect of REPTIL, being a token threaded language, is the association of the quote with the verb token. This gives the quote a fundamental significance similar to that of LISP, as follows: If the quoted name is found to be a predefined verb, then its token is pushed to TOP and its execution is suppressed, otherwise it is a candidate for becoming a verb. With this quote prefix notation, variables and constants are handled in a unique readable manner. The defining verb :IS creates a constant (it IS a value, and always will be) whereas :HAS creates a variable (it HAS a value cell which can contain a value, if so assigned). The content of a variable value cell is accessed simply by invoking it, leading to improved readability. Assignment is achieved by suppressing the variable action with a prefixed quote and using the assignment verbs <- (into) or <+ (addto). Similarly the address of the variable's value cell is obtained by using the verb CELL. All this is shown in the following example session:

```
<16>    'VALUE :HAS   \ create the variable VALUE
<16>    5 'VALUE <-   \ assign 5 to VALUE
<16>    42 'THE-ANSWER :IS  \ create the constant THE-ANSWER
<16>    THE-ANSWER 'VALUE <+   \ add 42 to the content of VALUE
<16>    VALUE =
47
<16>    'VALUE CELL =  \ the cell address of VALUE
40EC
<16>    ( 2 4 + , 6 8 + , 10 12 + , C )
4<16>   STACK?
6 E 22 C TOP
4<16>   + + + THE-ANSWER =?  \ are they equal? \  =
-1
<16>_
```

The creation of REPTIL has the advantage of hindsight, and its specification and development have been influenced by many different languages, suggestions, and language approaches. The token threaded infrastructure, decompiler and editor is based on RTL (5), many of the ideas of the defining verbs, prompts and basic structures are based on PISTOL (3), various other factors have been influenced by FORTRAN 77, Pascal, C, LISP, Logo and of course Forth. Putting it together has resulted in a unique combination of these various influences, and it is hoped that REPTIL can bridge the gap between humanoid and computer, and thus between education and application. The major advantages of REPTIL are that the postfix and stack notation is more readable; there is almost no context dependency of the verbs; there is a minimal but nevertheless complete and powerful set of structured constructs; recursion is intrinsically available. Over the past quarter REPTIL has been introduced into the Robotics 2 course in the Mechanical Engineering department at Ohio University with satisfying results.

References:
1. I Urieli,'REPTIL -- Bridging the gap between education and application', submitted to the Journal of Forth Application and Research.
2. J M Sachs, S K Burns, 'STOIC, an interactive programming system for dedicated computing', Software - Practice and Experience, Vol 13,1983, pp 1 - 16.
3. E E Bergmann, 'PISTOL - a Forth-like Portably Implemented Stack Oriented Language', Dr Dobb's Journal, Number 76, Feb 1983, pp 12 - 15.
4. E L Solley, 'SPHERE: an in-circuit development system with a Forth heritage', Proceedings of the 1984 Rochester Forth Conference, pp 25 - 31.
5. R Buege, 'Status Threaded Code', Proceedings of the 1984 Rochester Forth Conference, pp 103 - 104.