Universität Potsdam
Philosophische Fakultät

Hans-Joachim Petsche

In honour of Seymour Papert: „Empirical Modelling“ of Logo in Forth

Universität Potsdam

In honour of Seymour Papert: "Empirical Modelling" of Logo in Forth

Hans-Joachim Petsche

Potsdam University, Potsdam, Germany, e-mail: petsche@uni-potsdam.de

Abstract

Forth is nice and flexible but to a philosopher and teacher educator Logo is the more impressing language. Both are relatives of Lisp, but Forth has a reverse Polish notation where as Logo has an infix notation. Logo allows top down programming, Forth only bottom up. Logo enables recursive programming, Forth does not. Logo includes turtle graphics, Forth has nothing comparable. So what to do if you can't get Logo and have no information about its inner architecture? This should be a case of "empirical modelling": *How can you model observable results of the behaviour of Logo in terms of Forth?* The main steps to solve this problem are shown in the first part of the paper.

The second part of the paper discusses the problem of modelling and shows that the modelling of making and the modelling of recognition have the same mathematical structure. So "empirical modelling" can also serve for modelling desired behaviour of technical systems.

The last part of the paper will show that the heuristic potential of a problem which should be modeled is more important than the programming language. The Picasso construal shows, in a very simple way, how children of different ages can model emotional relations in human behaviour with a simple Logo system.

1 From Forth to Logo

1.1 The history of the idea

How does one get to the rather unusual idea of modelling the behaviour of the language Logo in the language Forth?

In 1985 I completed my doctoral thesis in the field of philosophical problems in applied mathematics and the role of computers in the new research field of experimental mathematics (Petsche 1985). Some key points of my doctoral thesis about "Mathematics as Driving Force of Scientific and Social Progress" were:

- The emergence of computers as *processing tools* changed mathematics in its characteristic as a *structural science. Simulation Modelling* moved into the focus of applied mathematics.
- The emergence of computers as *finite tools* led to a new status of *approximation mathematics* (see Blekhman, Myshkis, Panovko 1976).

And last but not least:

- The emergence of computers (of the third generation) as *dialog tools* became the hour of birth of *experimental mathematics*. (see also Moiseyev, 1979).

The possibility of a dialogue regime introduces an inductive moment into formal thinking. We are no longer bound to autonomous processing algorithms; we can interact, correct and modify formalisms. It is very interesting that in the same year (Beynon 1986), when I thought about the dialogue with computers, the fascination of the dialogue regime with computers led Meurig Beynon to his "paradigms for programming" and inspired him to the first ideas of Empirical Modelling.

The role of computers as dialog tools was very interesting and new, so I wanted to have a bit more experience in programming and in "what it means to work in dialogue with

the computer". And it happened that just at this time the first series of home computers (KC 85) were produced in the GDR (where I am from). But there was no possibility for me to buy one for private use (only a few hundred a month were sold at selected places). So I used the little amount of "Westgeld" (meaning foreign currency) I could get hold of (199 Deutschmarks) to buy an "Atari 800 XL" at the "Intershop" (a government-run retail store in which only hard currencies could be used). I did not have enough money for a floppy disk or datasette though. So I started with a basic decompiler which was so short, that I could enter it into the Atari within 20 minutes. I used this decompiler during a two week illness to decompile the whole operating system und write it down by hand (see Figure 1).

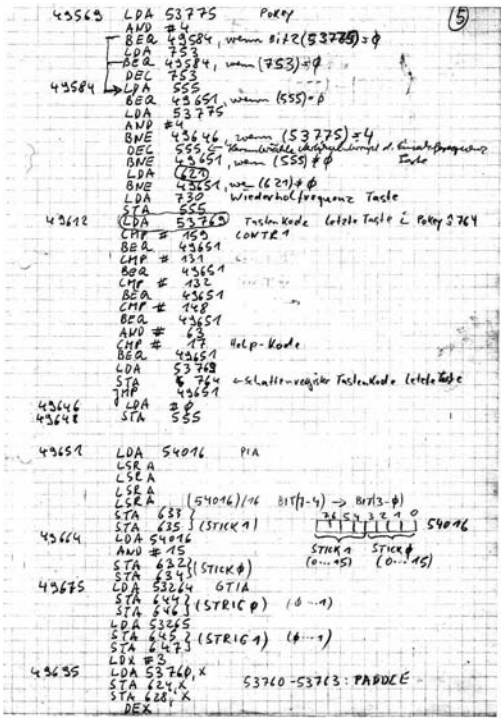


Figure 1. Part of a handwritten disassembled Atari OS

Then some friends helped me building an interface so I could use a normal cassette tape recorder to save and load data. Thanks to the simply structured assembler code of the 6502 processor the reverse engineering of Atari's operating system was not that hard. I never had again such a feeling of knowing what's really going on in a computer! The built-in Basic was not my cup of tea as a philosopher. So I turned to the easy to get Forth programming language. This language was at this time unique. It was a programming language as well as an operating system; it was able to compile programs as well as to work in an interpreter modus. It was a list-processing language like LISP.

Programming was the creation of words by words. You could develop your own private language. It seemed to be a wonderful programming language for philosophers...

But it uses Reverse Polish Notation (RPN), it has no graphic commands, it does not allow recursive procedures, it needs a consistent bottom up design of programming and the final programs are often hard to understand (in a "write-only" manner).

When I first read about Logo I was very fascinated by it. Like Forth, Logo is a relative of Lisp. But it has an infix notation, allows top down programming, enables recursive programming and includes turtle graphics: These are all advantages that Forth does not have.

So what to do if you can't get Logo for your computer and have no information on its inner architecture? This should be a case of "Empirical Modelling". The problem to be solved was:

How can you model the main observable results of the behaviour of Logo in terms of Forth?

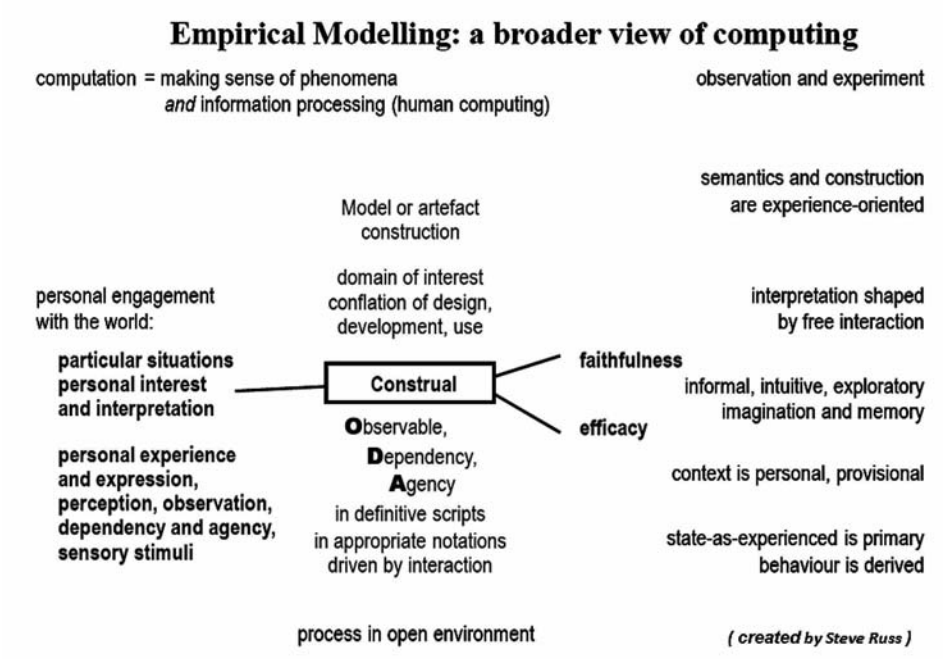


Figure 2. Empirical Modelling as a new approach of computing (Russ, 2009, Slide 10)

Why do I call it "Empirical Modelling"?

The artefact – the "Logo in Forth" (FLOKC) – that is built on the computer is itself a source of immediate experience which can be compared – through interaction – with experience of

its referent (LOGO). If we compare the characterization of Empirical Modelling by Steve Russ (see Figure 2) with our approach, we find only a few differences:

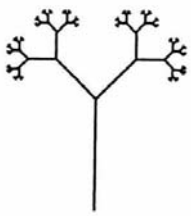
1. The empirical referent of our artefact is a computer language (LOGO) and no lift, no ant, no oxo, no poem or piece of music.

2. The construal is not build in definitive scripts in the environment of EDEN but in words as lists of words of Forth.

3. The construal can serve as a partial replacement of its referent (but with a different inner structure).

Two recursive programs – one for drawing a binary tree (Figure 3) and one for symbolic differentiation (Figure 4) – should serve as yardstick for measuring the success of rebuilding Logo in Forth.

```
TO BAUM :X
IF :X < 1 [STOP]
FD :X LT 45 BAUM :X/2 RT 90
BAUM :X/2 LT 45 BK :X
END
```



BAUM 64

Figure 3. Binary tree in Logo

```
    (* Symbolic Differentiation of a Function in LOGO *)

TO Chain_rule :FCT
   OP ( LIST KL ( LIST KL ABL FIRST :FCT "0 LAST :FCT )
   "* KL ABL LAST :FCT )
END

TO Product_rule :FCT
   OP ( LIST FIRST :FCT "* ABL ( LAST :FCT )
   "+ ABL FIRST :FCT "* LAST :FCT )
END

TO Sum_rule :FCT
   OP ( LIST ABL FIRST :FCT "+ ABL LAST :FCT )
END

TO Power_rule :FCT
   MAKE "M LAST :FCT
   IF :M  = 0 [ OP [0] ]
   MAKE "N ( LIST "ID "ex :M - 1 )
   OP ( LIST :M "* :N )
END

TO ELEM   :FCT
   MAKE "F  FIRST   :FCT
   IF :F =   "ID [OP "1]
   IF :F =   "SIN    [OP "COS ]
   IF :F =   "COS    [OP "-SIN]
   IF :F =   "-SIN   [OP "-COS]
   IF :F =   "-COS   [OP "SIN ]
   IF :F =   "EXP    [OP "EXP ]
   IF  F =   "LN     [OP ( LIST "ID "ex "-1 )
   OP "0 ]
END

TO ABL FCT ;
   IF NOT LISTP :FCT [MAKE "FCT KL :FCT]
   IF ( COUNT :FCT ) = 1 (OP ELEM :FCT STOP]
   IF ( COUNT :FCT ) = 2 (OP LIST "FCT "CORRECT? STOP]
   PR "BEGIN
   MAKE "Z FIRST BF :FCT
   IF :Z = "*  [OP PRODUKTREGEL :FCT]
   IF :Z = "+  [OP SUMMENREGEL  :FCT]
   IF :Z = "o  [OP KETTENREGEL  :FCT]
   IF :Z = "ex [OP POTENZREGEL  :FCT]
END

TO  KL   :A
    OP FPUT :A [ ]
END
```

Figure 4. Program for symbolic differentiation in Logo

1.2 How to rebuild Logo in Forth?

The first problem was to replace the RPN with an *infix notation.* (see Baranov, Nozdrunov 1988). For this purpose I had to separate data from operations by means of a second stack and find the agency which would do that. I found two kinds of agents: arithmetic and logic operators with their priorities which will push themselves on the operation stack and all kinds of brackets (including "Enter") which will start to collect the operators and end with the execution of the operations. This agency with a second stack gives Forth the desired Logo-like infix notation (see figure 5 and 6).

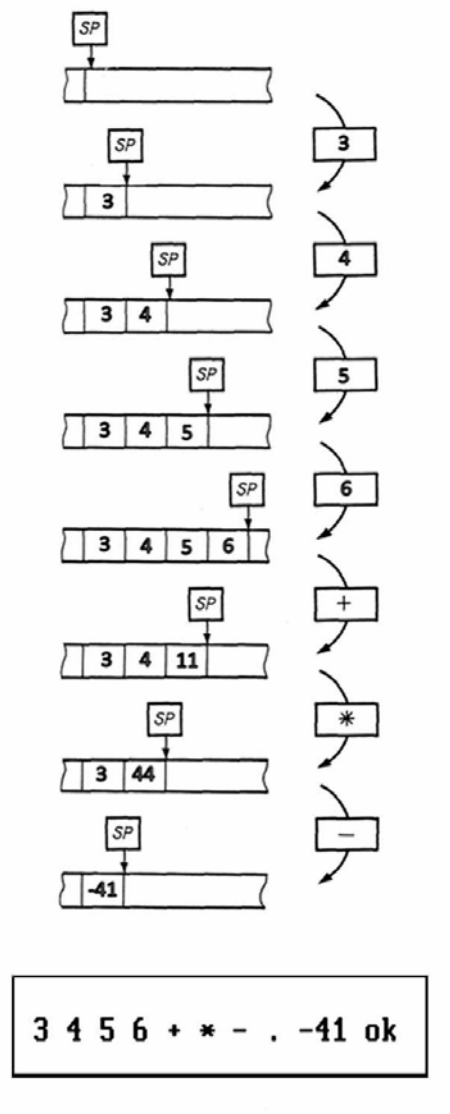


Figure 5. Changes of stackpointer in Forth (RPN)

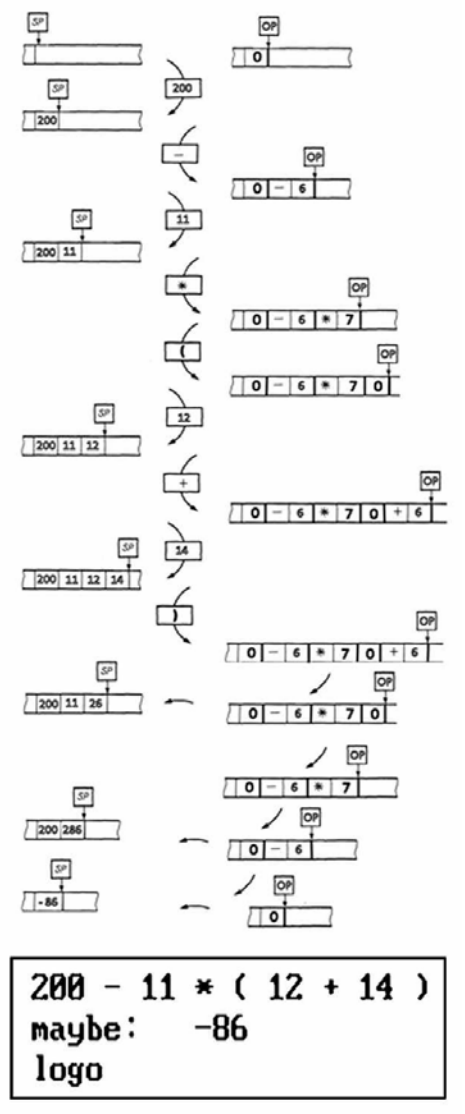


Figure 6. Changes of stack- and operationpointer in Forth-Logo (Infix notion)

Now I had to teach Forth the *top down programming*. For that we need a temporary vocabulary and a redefinition of the word "abort" as an agent ("abort" usually outputs an error message when we call an undefined word):

1. If a word in the vocabulary of the system is not found during the compilation of the corresponding text screen and is not interpreted as a number, the word "abort" now carries out an additional search run.
2. First, it is checked whether the given word represents the self-call of the word currently being defined. If this is the case, it is compiled into the vocabulary and the program returns to normal compilation mode (thus a first step is made in the direction of recursive programming). If this check is unsuccessful, it also checks whether the unknown word already exists in a temporary vocabulary. If this is the case, the word “??” is compiled into the word to be defined. The address of this word is entered into the temporary vocabulary under the name of the still undefined word. If the word is not found in the temporary vocabulary, it is entered into this vocabulary.
3. In addition, we need four new agents, which we find in the words "create" and ":" (which start compiling new words) as well as in "variable" and "constant" (which hold data). These new agents perform their usual functions but also have the additional ability to test if a new word is already in the temporary vocabulary.
   If this is the case, the original word, which has now been defined, is entered at all points at which this word was called and in which “??” was temporarily present. The execution of the substitution is marked in the temporary vocabulary.
4. If some words remain undefined after compiling, we get an error message and can correct this in a next loop.

The realization in Forth looks a bit crazy, if one is not familiar with programming in Forth. But now we have the possibility to program top down as well as bottom up (see Table1)!

Table 1

TOP-DOWN-FORTH-Extension

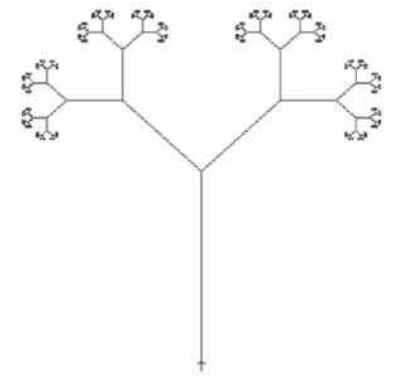
```
HERE FIRST § 1000 - DP ! LATEST CONSTANT TD-
' (ABORT) § CONSTANT AB 0 VARIABLE HL   0 VARIABLE TP
0 VARIABLE CUL  0 VARIABLE CUH 0 VARIABLE T0   0 VARIABLE IM

: TER? PAD T0 § > IF ." TOP-STACK-ERROR" 1 ERROR ENDIF ;
: ?? CR ." UNKNOWN WORD IN ADR: " R> CFA U. CR SP! DECIMAL QUIT ;
: VOR CUH CURRENT DUP § CUL ! ! HERE HL ! TP § DP ! ;
: RE  CUL § CURRENT ! HERE TP ! HL § DP ! ;
: EX WARNING !   '    (ABORT) ! ;
: CREA HL § HERE 40 CMOVE 0 BRANCH (( ' CREATE 2+ HERE - , )) ;
: REK SMUDGE LATEST PFA LFA DUP DUP § 0 ROT ! HERE LATEST (FIND)
     >R R IF DROP CFA , ENDIF   SWAP ! R> SMUDGE ;
: MERK TER? HERE CUH § DUP
           IF (FIND) ELSE SWAP DROP ENDIF
           0= IF VOR CREA SMUDGE RE
                 ELSE DROP CFA DUP § 2+ TP § DUP ROT ! SWAP !
              ENDIF
           4 TP HERE OVER § 0 OVER 2+ ! ! +! ' ?? CFA , ;
: TOP DUP 0= IF REK 0= IF MERK ENDIF
               DROP DROP DROP DROP R> R> R> R> R> R>
               DROP DROP DROP DROP DROP DROP ' INTERPRET >R ;S
            ENDIF R> DROP ;
: IN-TOP TER? TP § 0= IF TO § TP ! ENDIF
        0 CUH ! '   TOP CFA -1 EX ;
: PUT DROP 0 OVER CFA ! LATEST PFA CFA >R
     BEGIN R OVER § ! 2+ § DUP 0=
     UNTIL
     R> DROP DROP ;
: ?IN IN § IM ! ;
: NORM TD- TD- ! 0 TP ! AB 0 EX ;
: TOP-TEST 0 CUH §
          BEGIN DUP
            WHILE PFA DUP CFA §
             IF DUP NFA ID. 2 SPACES
                ." UNKNOWN WORD " CR
                SWAP 1+ SWAP
             ENDIF  LFA §
          REPEAT
          DROP 0= IF NORM ENDIF ;
: ?TOP CUH §
        IF TER? IM § IN ! BL WORD HERE CUH § (FIND)
           IF PUT ENDIF
        ENDIF ;
: TOP-LOAD IN-TOP LOAD TOP-TEST :
: VARIABLE ?IN VARIABLE ?TOP ;
: CONSTANT ?IN CONSTANT ?TOP ;
: CREATE ?IN CREATE SMUDGE ?TOP SMUDGE ;
: : ?IN (COMPILE) : SMUDGE ?TOP SMUDGE ; IMMEDIATE
: Top CR ." 1. TOP-STACK-Anf.  T0 ! " CR ." 2. Screen-Nr.  TOP-LOAD " CR ;
DP !  HERE 13 + CONSTANT TD
: TOP (( TD ' TD NFA )) LITERAL LITERAL ! Top ;
: -TOP ( ( LATEST PFA LFA TD- ) ) LITERAL LITERAL ! ( ( LATEST ) )
      LITERAL DUP 32 TOGGLE 8224 OVER 1+ ! 40992 SWAP 3 + ! ;
' TD CFA ' NORM 2+ ! ?  TD- TD !
                                          ( H.-J.Petsche, 3.4.1989 )
```

The last significant problem was to implement *recursive programming*. To enable this, we created a new type of 'agent' - recursive variables. This new type of variables – which does not exist in Logo – has an own stack for their values *depending* on the deepness of recursion. If you call such a variable it looks at its recursion pointer and selects the corresponding value.

So a recursive variable is an agent to manage its own recursive values.

Modelling the rest of the Logo functions caused no serious problems. A test shows that our program works in a Logo-like input-output manner.

If we finally compare Logo with our empirical model of Logo in Forth ("FLOKC"), we see both similarities and differences. The program for drawing a binary tree is almost identical in Logo and FLOKC (see Figure 7). Only in FLOKC we have to explicitly define a recursive (local) variable.



```
REK H

TO BAUM H;
IF H < 1 THEN STOP ENDIF
FD H LT 45 BAUM H / 2; RT 90
BAUM H / 2; LT 45 BK H
END
```

Figure 7. Flokc program of a binary tree

The programming of symbolic differentiation is almost word-for-word translatable from Logo to FLOKC. But we can also translate it in a manner more typical of FLOKC's basic idea. In addition to the use of list operators ("L>"), we can determine the operators and the elementary functions as agents that generate their own derivatives. This is shorter and makes the program even more transparent (see Table 2) ...

Table 2

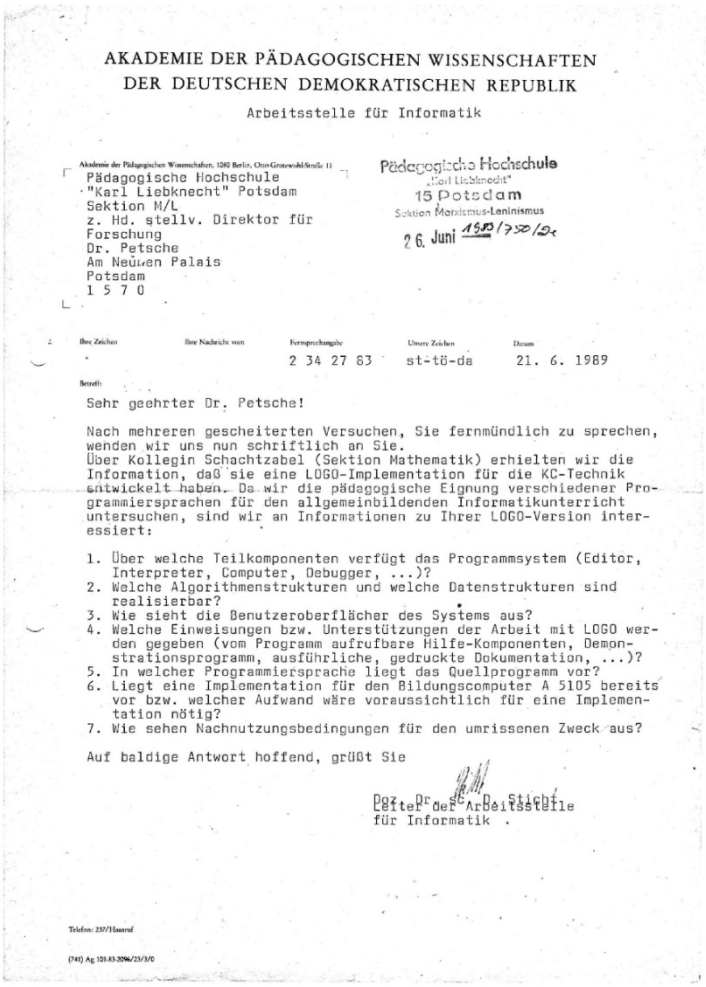
```
(* Symbolic Differentiation of a Function in FLOKC *)

lis function     rek fct      var f

to abl function ; (* start program *)
  cr make fct " function ;  diff fct ;
end

to diff fct ; (* differentiation *)
  if ( listp fct )  = 0 then ? 0                    stop endif
  if ( count fct ) = 1    then run fct   stop endif
  if not ( ( count fct ) = 3 ) then lp fct ? bad! stop endif
  "( run ( item 2 fct ) )"
end

to op f ; (* Output *)
  if listp f then if ( count  f ) > 1 then lp f stop endif
  endif  pr  f
end

(* elementary function *)

l>  cos [ ? -sin ]      l> sin [ ? cos ]
l> -sin [ ? -cos ]      l> -cos [ ? sin ]
l>  exp [ ? exp  ]      l>  x  [ ? 1   ]
l>  ln  [ p" x ex -1"   ]

to + ;   (* sum rule *)
  diff first fct ; ? + diff  last fct ;
end

to * ;   (* product rule *)
    op first fct ;  ? * diff last fct ;  ? + diff first fct ; ? * op last fct ;
end

to o ;   (* chain rule *)
  "(  diff first fct ; ? o op  last fct ;   )"  ? * diff last fct ;
end

to ex ;   (* power rule *)
  make f last fct ;
  if  listp  f  then lp fct ? bad! stop endif
  if  f  = 0    then  ? 0               stop endif
  pr f  p" * x ex "  pr ( f - 1 )
end
```

1.3 Finally, what can be said about the fate of "Logo in Forth"?

In April 1989, a volume of the journal "Potsdamer Forschungen" (Potsdam researches), which I authored together with a mathematician and computer scientist, was published entitled

"LOGO IN FORTH - Einführung in ein Sprachkonzept für die Informatikausbildung" (Logo in Forth. Introduction to a language concept for education in computer science. Petsche, Schachtzabel, Sprengel 1989).

In June 1989, the Academy of Pedagogical Sciences in the GDR wrote to me that they were interested in my program (see Figure 8).



AKADEMIE DER PÄDAGOGISCHEN WISSENSCHAFTEN
DER DEUTSCHEN DEMOKRATISCHEN REPUBLIK
Arbeitsstelle für Informatik

Akademie der Pädagogischen Wissenschaften, 1040 Berlin, Otto-Grotewohl-Straße 11
Pädagogische Hochschule
"Karl Liebknecht" Potsdam
Sektion M/L
z. Hd. stellv. Direktor für
Forschung
Dr. Petsche
Am Neuen Palais
Potsdam
1 5 7 0

Pädagogische Hochschule
„Karl Liebknecht"
15 Potsdam
Sektion Marxismus-Leninismus
26. Juni 1989/750/0x

Ihre Zeichen	Ihre Nachricht vom	Fernsprechangabe	Unsere Zeichen	Datum
-		2 34 27 63	st-tö-da	21. 6. 1989

Betreff:

Sehr geehrter Dr. Petsche!

Nach mehreren gescheiterten Versuchen, Sie fernmündlich zu sprechen, wenden wir uns nun schriftlich an Sie.
Über Kollegin Schachtzabel (Sektion Mathematik) erhielten wir die Information, daß sie eine LOGO-Implementation für die KC-Technik entwickelt haben. Da wir die pädagogische Eignung verschiedener Programmiersprachen für den allgemeinbildenden Informatikunterricht untersuchen, sind wir an Informationen zu Ihrer LOGO-Version interessiert:

1. Über welche Teilkomponenten verfügt das Programmsystem (Editor, Interpreter, Computer, Debugger, ...)?
2. Welche Algorithmenstrukturen und welche Datenstrukturen sind realisierbar?
3. Wie sieht die Benutzeroberflächer des Systems aus?
4. Welche Einweisungen bzw. Unterstützungen der Arbeit mit LOGO werden gegeben (vom Programm aufrufbare Hilfe-Komponenten, Demonstrationsprogramm, ausführliche, gedruckte Dokumentation, ...)?
5. In welcher Programmiersprache liegt das Quellprogramm vor?
6. Liegt eine Implementation für den Bildungscomputer A 5105 bereits vor bzw. welcher Aufwand wäre voraussichtlich für eine Implementation nötig?
7. Wie sehen Nachnutzungsbedingungen für den umrissenen Zweck aus?

Auf baldige Antwort hoffend, grüßt Sie

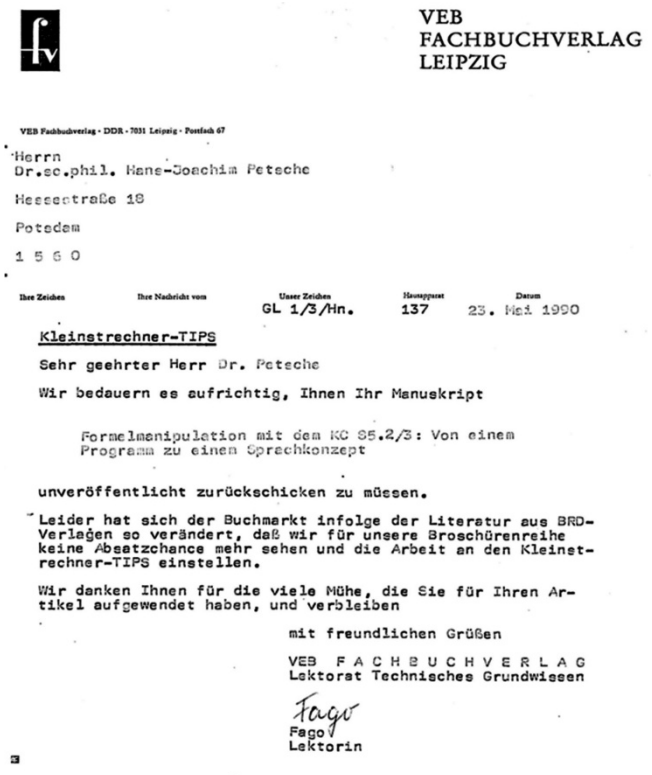
Dr. sc. D. Stiehl
Leiter der Arbeitsstelle
für Informatik

Telefon: 237/Hausruf

(740) Ag 101-83-2096/23/3/0

Figure 8. The "Academy of Pedagogical Sciences in the GDR" wrote me a letter on June 21, 1989, saying that they were interested in my logo implementation.

I also got a green light for the publication of three papers on "Top down in Forth" and "Logo in Forth". Then in November 1989 came the "Wende" (the “Turnaround”) and already at the beginning of 1990 no one in the GDR was interested in Logo or Forth anymore. With the accession of the GDR to the Federal Republic of Germany the project came to an end. Now there were other priorities (see Figure 9).



VEB
FACHBUCHVERLAG
LEIPZIG

VEB Fachbuchverlag · DDR - 7031 Leipzig · Postfach 67

Herrn
Dr.sc.phil. Hans-Joachim Petsche

Hesseestraße 18

Potsdam

1 5 6 0

Ihre Zeichen	Ihre Nachricht vom	Unser Zeichen	Hauptpunkt	Datum
		GL 1/3/Hn.	137	23. Mai 1990

Kleinstrechner-TIPS

Sehr geehrter Herr Dr. Petsche

Wir bedauern es aufrichtig, Ihnen Ihr Manuskript

Formelmanipulation mit dem KC 85.2/3: Von einem Programm zu einem Sprechkonzept

unveröffentlicht zurückschicken zu müssen.

Leider hat sich der Buchmarkt infolge der Literatur aus BRD-Verlagen so verändert, daß wir für unsere Broschürenreihe keine Absatzchance mehr sehen und die Arbeit an den Kleinstrechner-TIPS einstellen.

Wir danken Ihnen für die viele Mühe, die Sie für Ihren Artikel aufgewendet haben, und verbleiben

mit freundlichen Grüßen

VEB FACHBUCHVERLAG
Lektorat Technisches Grundwissen

Fago
Lektorin

Figure 9. A letter from the Fachbuchverlag Leipzig, dated May 23, 1990, informing me about the Journal's discontinuation, because after the reunification of Germany no market would exist for it. My submitted article would therefore not be published.

2 Some additional remarks about modelling

"Making construals is a new digital skill that aims to bridge the gap between computing specialists and non-specialists. Its focus is on using the computer as an instrument to make connections in experience - an activity that complements computational thinking." (Beynon et al., 2015)

In general, mathematical modelling is used to solve three types of problems (see Petsche 1988):

1. *The analysis problem:* The *behaviour* of an external system with a defined (usually only implicit or blurred) structure under (mostly blurred) conditions should be determined by computer applications.

2. *The synthesis problem:* From a (mostly blurred given) set of objectively possible systems of a specific type, the one that determines a (mostly blurred) *preselected behaviour* as far as possible (under predominantly blurred criteria) must be determined by the use of computer applications.

3. *The recognition problem:* From the *behaviour* of an external system *signalled* by structured data sets, the specific structure of this system must be determined by computer applications as precisely as possible within the framework of the given requirements and in the context of a recognition hypothesis about the affiliation of this system to a certain set of objectively possible systems of a particular type.

In my opinion Empirical Modelling generally belongs to the field of recognition problems. Empirical modelling is a special way to solve recognition problems: It plays with models as construals, as "objects-to-think-with" about empirically given objects. It does not build a simulation model but it generates a deeper individual human insight into the behaviour of the referent.

An "objects-to-think-with" can not only model an empirically given but also an empirical wanted object. Construal as an "objects-to-think-with-about" could serve to think about given as well as about desired objects. *Construals could be placed in the field of recognition as well as in the field of synthesis.*

This broader look on empirical modelling has its reasons: As can be shown, that the synthesis (2) and the recognition problem (3) have the same mathematical structure and the modelling of these problems is essentially identical. *"Recognition"* and *"Making"* have the same (mathematical) structure.

Modelling Logo in Forth is in my opinion an example for synthesising the behaviour of a desired object (a programming language) by observing the behaviour of the original object and by playing with agencies and dependencies.

I think that other examples of empirical modelling desired behaviour of technical systems would be desirable. The just-in-time-modelling with definitive scripts in an experimental computer environment will sharpen the technological sense of modellers.

3 The Picasso Construal: Papert meets Picasso

3.1 The Picasso problem

So FLOKC is dead. It was dead long before the Logo Tree Project started. Making it fit for now – implementing it in a windows forth environment, adding an comfortable hypertext editor, adding floating point instructions, adding multimedia possibilities – would not be so hard. But there are enough versions of LOGO on the market.

Steve Russ, one of the organzisers of the Construit 2017 conference asked me a month before the conference started, which programming language would I as a philosopher and teacher educator consider to be the best for children? This seemed to be an easy question. But it wasn't. Well known is Heideggers bon mot, that "Language is the House of Being".
Language is not the construct of Being and also not the universe of Being.

And language is not only a house of Being.

Language is build by man and by Being.

Language should work for logicians as well as for geometricians (how Poincaré would say), for analysers, synthesists, constructivists and intuitionists.

Every child will develop his own access to the world. And I think that in this context SCRATCH for example is too LEGO-like, too superficially oriented on making (and constructing) and not on thinking, rethinking and problematising. Edutainment can be good but not always is. So it seems to me, that a LOGO-like (LISP oriented) language could be a good compromise.

And a final "but": The language is not the decisive point. If in the house of Being there is no Being, language will not give us anything.
We need good problems, which are worth talking about, that are worthy of modelling. It is only the next step to find a good representation. Faraday discovered such problems and found excellent representations!

"Discovery consists of seeing what everybody has seen and thinking what nobody has thought." (Albert Szent-György, cited by Irving J. Good)

To use the computer for thought, you need not only a good programming environment, but above all a paradigmatic problem. First you must have the problem then you have to get its powerful representation.

There is a small example that occurred to me some weeks ago. Let me call it *the Picasso Construal.* (I have programmed it in my old FLOKC environment, which was not exactly simple, because it runs under DOS)

During his Cubist period, Picasso was asked if he could express "affection" and "astonishment" with only a few brushstrokes (see Krumbholz 1969, p. 192). And he could indeed! (see Figure 10)

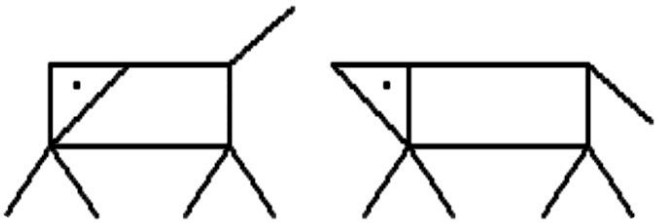


Figure 10. Picasso's visual interpretation of "affection" and "astonishment"

Picasso modelled human behaviour through the behaviour of dogs which for their part were modelled with a few points, triangles, and rectangles. *That was ingenious*.

And he gave us a fantastic problem for empirical modelling:
Which other kinds of human behaviour could be expressed through modelling of dogs with points, triangles, and rectangles. And what will occur if we add some kind of motion?

3.2 From turtle graphics to doggie graphics

In LOGO, we can use the commands of turtle graphics to create a doggie graphics environment. An old LOGO (written in FORTH) running under DOS is already good enough for this. Compilation of the following commands can be carried out by older children:

head_left / head_right
tail_up / tail_down
go_left / go_right
grow / shrink

color / steps

First you can do this for one dog - and a bit harder - for two independent dogs:

dog1 / dog2 / both

head_left1 / head_left2

…

go_left1 / go_left2 /

...

go_both / grow_both

...

Then you can use these new commands in interpreter mode. And now the younger children can also experiment with the modelling of emotions in the doggie environment (for example see Figure 11 – 14).

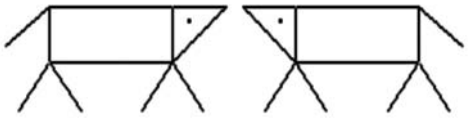


Figure 11. Inspect

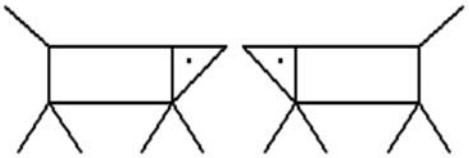


Figure 12. Joy

Figure 13. Disinterest

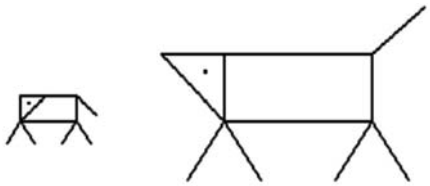


Figure 14. Fear

And by modelling the motion of the Picasso dogs the children can also “tell” little visual stories of emotional behaviour (they can make little "films" in a very easy way) and analyze what's going on ...[1]

So we need more good problems to make construals on computers which give us more good "things to think with" using a kind of empirical modelling which is suitable for children.

[1] See a little example on YouTube (Petsche, 2017).

References

Baranov, S. N., & Nozdrunov, N. R. (1988). *Yazyk Fort i yego realizatsii*. Leningrad: "Mashinostroyeniye".

Beynon, M. (1986). *The LSD notation for communicating systems. University of Warwick. Department of Computer Science*. (Department of Computer Science Research Report). (Unpublished) CS-RR-087. Permanent WRAP url: http://wrap.warwick.ac.uk/60783

Beynon, M., et al. (2015). *Making construals as a new digital skill: dissolving the program – and the programmer – interface*. Proceedings of the 2015 International Conference on Interactive Technologies and Games, 22-23 October 2015, Nottingham, UK, pp9-16.

Blekhman, I. I., Myshkis, A. D., & Panovko, Ya. G. (1976). *Prikladnaya matematika: predmet, logika, osobennosti podkhodov.* Kiyev: "Naukova dumka".

Krumbholz, E. (1969). Neue Fingerzeige. Anekdoten. Halle: Mitteldeutscher Verlag.

Moiseyev, N. N. (1979). *Matematika stavit eksperiment*. Moskva: "Nauka".

Petsche, H.-J. (1985). *Zur Bestimmung der Rolle der Mathematik als Triebkraft der wissenschaftlich-technischen Revolution – einige inhaltliche und methodologische Aspekte einer philosophischen Analyse*. 1985. Diss. B. Potsdam: Päd. Hochschule.

Petsche, H.-J. (1988). *Zu einigen weltanschaulich-philosophischen Aspekten der Mathematikanwendung bei der Synthese erkennungsfähiger technischer Systeme (Identifikatoren).* In: Potsdamer Forschungen. Reihe A; 87. Potsdam: Päd. Hochschule, pp. 120-146.

Petsche, H.-J. (2017, August 8). *The Picasso Construal - a Logo in Forth example*. Retrieved from https://www.youtube.com/watch?v=zxj25IrN044

Petsche, H.-J. (Ed.), Schachtzabel. H., & Sprengel, H.-J. (1989). *LOGO in FORTH - Einführung in ein Sprachkonzept für die Informatikausbildung.* (Potsdamer Forschungen. Naturwissenschaftliche Reihe, Heft 65). Potsdam: Päd. Hochschule.

Russ, S. (2009). *Human Computing.* (Slides). Retrieved from: http://slideplayer.com/slide/7557293/