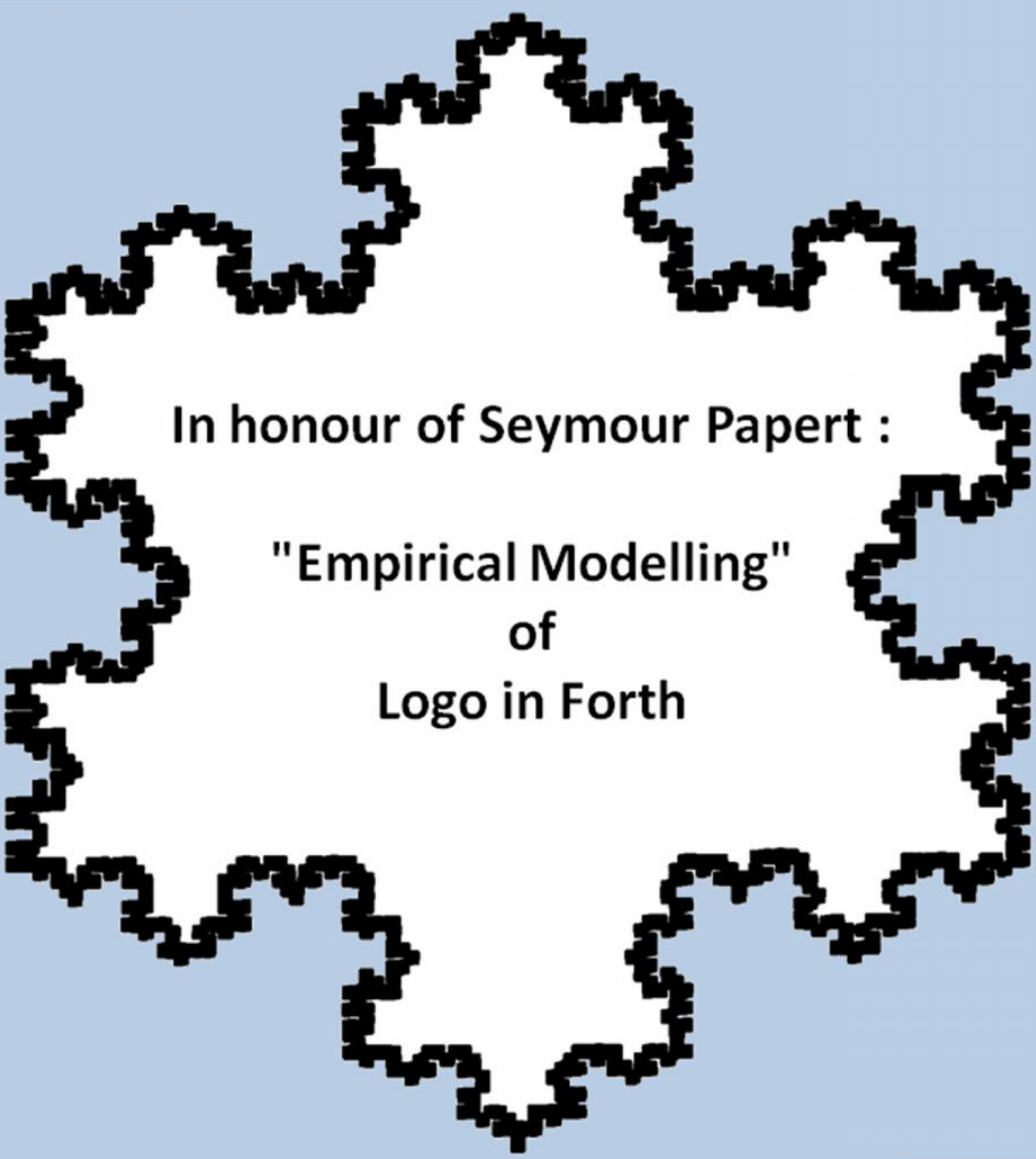
Institut für Philosophie

Universität Potsdam



In honour of Seymour Papert :

"Empirical Modelling" of Logo in Forth

Prof. Dr. Hans-Joachim Petsche

Institut für Philosophie

Universität Potsdam

Some key points of my doctoral thesis about
"Mathematics as Driving Force of Scientific and Social Progress":

- 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**.

And last but not least:

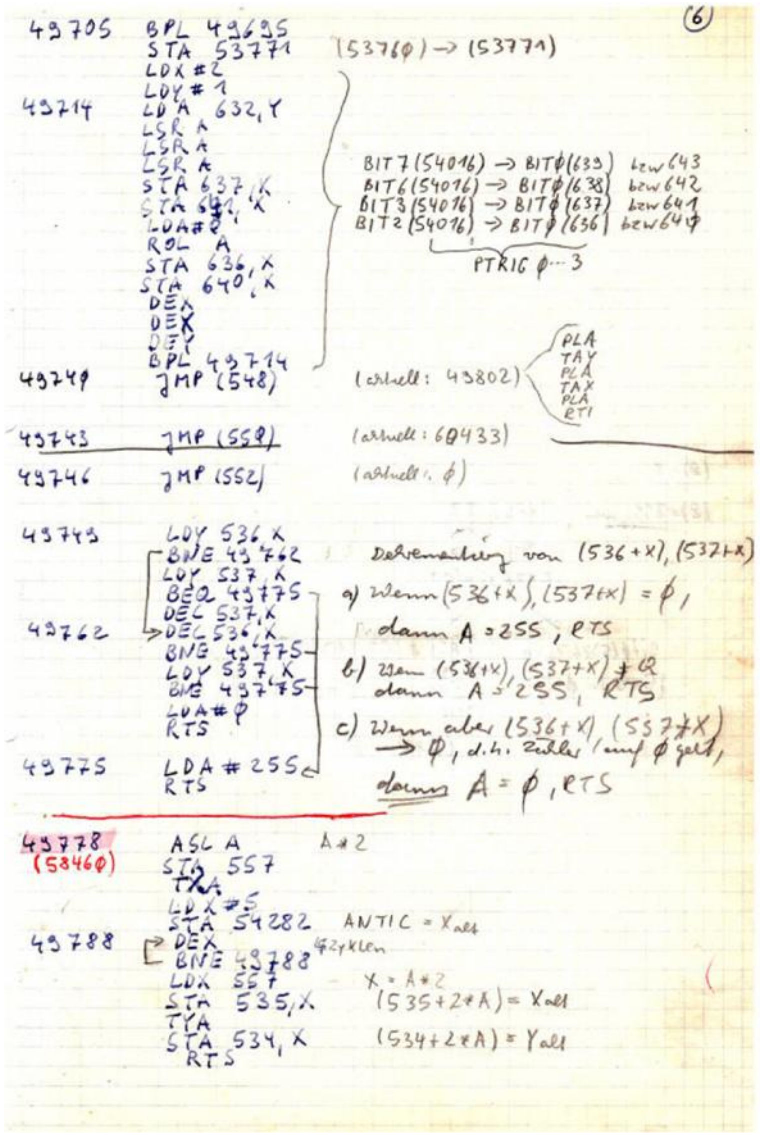
- The emergence of computers (of the third generation) as **dialog tools** became the hour of birth of **experimental mathematics**.

"Empirical Modelling" of Logo in Forth

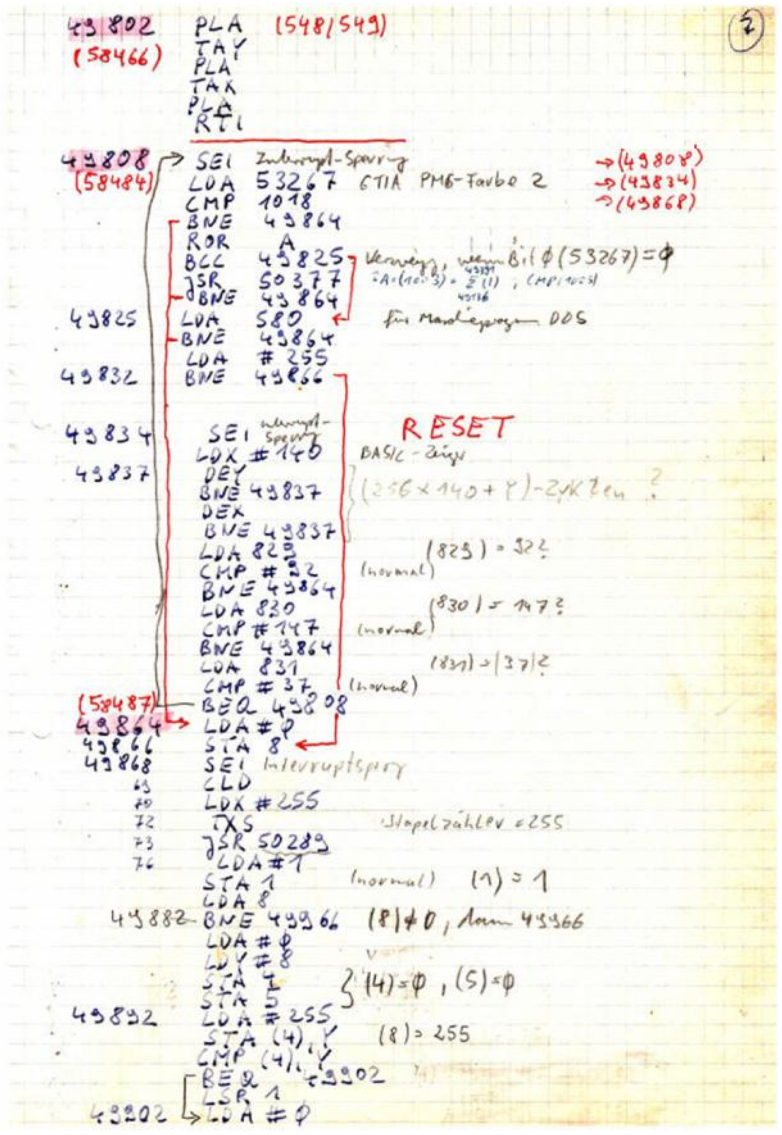
Institut für Philosophie Universität Potsdam

KC 85/1 from robotron (GDR) and
Atari 800 XL from Atari, Inc. (USA)

"Empirical Modelling" of Logo in Forth



```
                                                            (6)
49705   BPL 49695
        STA 53771        (5376φ) -> (53771)
        LDX #2
        LDY #1
49714   LDA 632,Y
        LSR A
        LSR A
        LSR A            BIT7(54016) -> BITφ(639) bzw 643
        STA 637,X        BIT6(54016) -> BITφ(638) bzw 642
        STA 641,X        BIT3(54016) -> BITφ(637) bzw 641
        LDA#φ            BIT2(54016) -> BITφ(636) bzw 64φ
        ROL A                 PTRIG φ…3
        STA 636,X
        STA 640,X
        DEX
        DEX
        DEY
        BPL 49714                            PLA
49740   JMP (548)        (aktuell: 49802)    TAY
                                             PLA
                                             TAX
                                             PLA
                                             RTI
49743   JMP (55φ)        (aktuell: 6φ433)
49746   JMP (552)        (aktuell: φ)

49749   LDY 536,X
        BNE 49762        Dekrementierung von (536+X), (537+X)
        LDY 537,X
        BEQ 49775        a) Wenn (536+X), (537+X) = φ,
        DEC 537,X           dann A = 255, RTS
49762   DEC 536,X
        BNE 49775        b) wenn (536+X), (537+X) ≠ φ
        LDY 537,X           dann A = 255, RTS
        BNE 49775
        LDA #φ           c) Wenn aber (536+X), (537+X)
        RTS                 -> φ, d.h. Zähler auf φ geht,
49775   LDA #255            dann A = φ, RTS
        RTS
--------------------------------------------
49778   ASL A            A*2
(5846φ) STA 557
        TXA
        LDX #5
        STA 54282        ANTIC = X_alt
49788   DEX              42 Zyklen
        BNE 49788
        LDX 557          X = A*2
        STA 535,X        (535+2*A) = X_alt
        TYA
        STA 534,X        (534+2*A) = Y_alt
        RTS
```



```
                                                            (7)
49802   PLA   (548/549)
(58466) TAY
        PLA
        TAX
        PLA
        RTI
--------------------------------------------
49808   SEI        Interrupt-Sperrung                 -> (49808)
(58484) LDA 53267  GTIA PMG-Farbe 2                   -> (49834)
        CMP 1018                                      -> (49868)
        BNE 49864
        ROR A
        BCC 49825  Verzweigung, wenn Bit φ (53267) = φ
        JSR 50377  = A = (1018) = E(1) ; CMP(1023)
        BNE 49864
49825   LDA 580    für Maschinenprogramm DOS
        BNE 49864
        LDA #255
49832   BNE 49866

49834   SEI  Interrupt-Sperrung        RESET
        LDX #140                  BASIC-Zeiger
49837   DEY
        BNE 49837                 (256×140+Y)-Zyklen ?
        DEX
        BNE 49837
        LDA 829                   (829) = 32 ?
        CMP #32        (normal)
        BNE 49864
        LDA 830                   (830) = 147 ?
        CMP #147       (normal)
        BNE 49864
        LDA 831                   (831) = 37 ?
        CMP #37        (normal)
(58487) BEQ 49808
49864   LDA #φ
49866   STA 8
49868   SEI  Interruptsperre
69      CLD
70      LDX #255
72      TXS                       Stapelzähler = 255
73      JSR 50289
76      LDA #1
        STA 1          (normal)   (1) = 1
        LDA 8
49882   BNE 49966      (8) ≠ 0, dann 49966
        LDA #φ
        LDY #8
        STA 4          (4) = φ , (5) = φ
        STA 5
49892   LDA #255
        STA (4),Y      (8) = 255
        CMP (4),Y
        BEQ 49902
        LSR 1
49902   LDA #φ
```

2 of 128 pages of the handwritten disassembling code of the Atari 800XL operating system

Fascination of Forth:

- Forth was a programming language as well as an operating system.
- Forth was able to compile programs as well as to work in an interpreter modus.
- Forth was a list-processing language like LISP.
- Forth programming was the creation of words by words.
- Forth has the ability of reflection, that means that Forth can examine, introspect, and modify its own structure and behaviour at runtime.
- Forth let you develop your own private language.

But:

- Forth uses RPN (reverse Polish notation).
- Forth has no graphic commands.
- Forth does not allow recursive procedures.
- Forth needs a consequent bottom up design of programming.
- The final programs are often hard to understand (in a "write-only" manner).

Fascination of Logo:

- Logo is (like Forth) a relative of Lisp.
- Programming is the creation of words by words.
- Unlike Forth, Logo has an infix notation.
- Logo allows top down programming.
- Logo enables recursive programming.
- Logo has a turtle graphic.

So what to do if you can't get Logo for your PC and have no information about its inner architecture?

This should be a case of "Empirical Modelling".

The problem to be solved was:

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

Empirical Modelling: a broader view of computing

computation = making sense of phenomena
and information processing (human computing)

observation and experiment

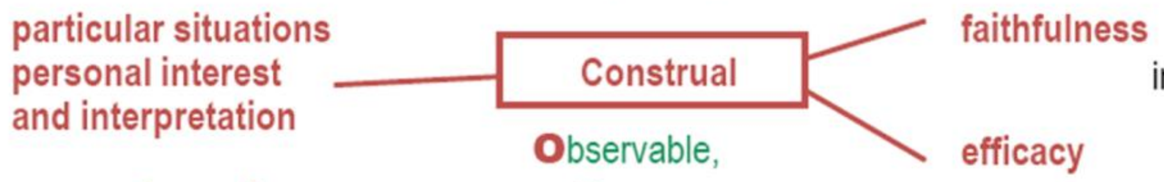
semantics and construction
are experience-oriented

Model or artefact
construction

domain of interest
conflation of design,
development, use

personal engagement
with the world:

interpretation shaped
by free interaction



informal, intuitive, exploratory
imagination and memory

Observable,
Dependency,
Agency
in definitive scripts
in appropriate notations
driven by interaction

personal experience
and expression,
perception, observation,
dependency and agency,
sensory stimuli

context is personal, provisional

state-as-experienced is primary
behaviour is derived

process in open environment

(created by Steve Russ)

Institut für Philosophie

Universität Potsdam

“Empirical Modelling” of Logo in Forth

Why do I call it "Empirical Modelling"?

The artefact 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 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 replacement of its referent (but with a different inner structure).

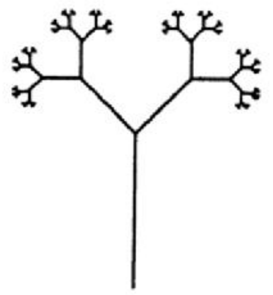
"Empirical Modelling" of Logo in Forth

Institut für Philosophie Universität Potsdam

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

```
(* 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 PRODUCT_RULE :FCT]   IF :Z = "+  [OP SUM_RULE   :FCT]
  IF :Z = "o  [OP CHAIN_RULE   :FCT]       IF :Z = "ex [OP POWER_RULE  :FCT]
END

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

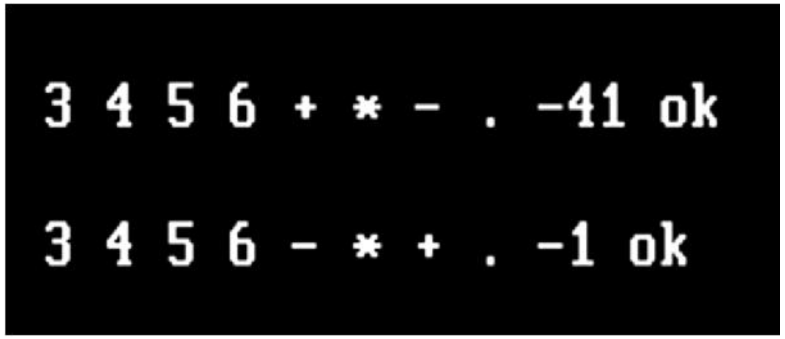
```
TO BAUM :X
IF :X < 1 [STOP]
FD :X LT 45 BAUM :X/2 RT 90
BAUM :X/2 LT 45 BK :X
END
```



BAUM 64

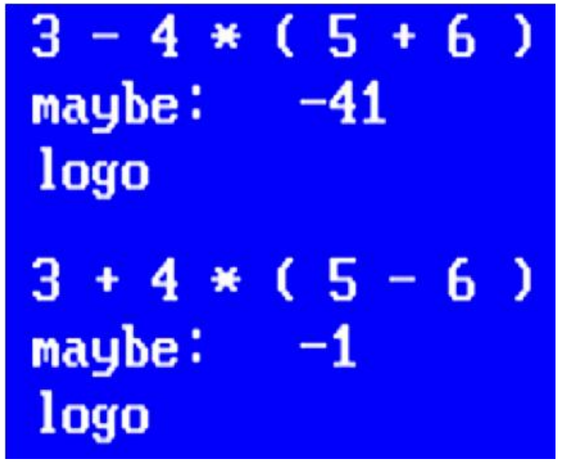
Prof. Dr. Hans-Joachim Petsche

Replacing the RPN of Forth with an infix notation of Flokc



```
3 4 5 6 + * - . -41 ok

3 4 5 6 - * + . -1 ok
```

Forth

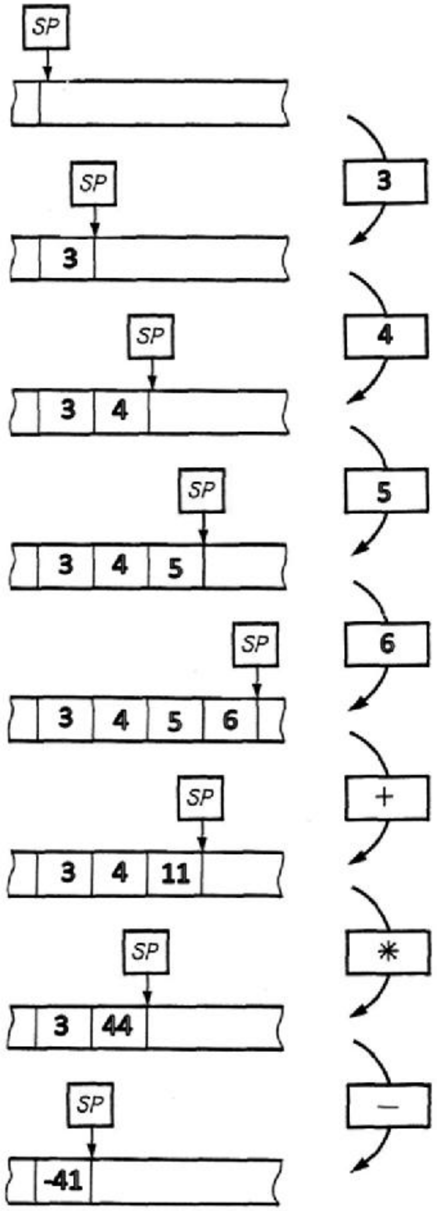


```
3 - 4 * ( 5 + 6 )
maybe:   -41
logo
3 + 4 * ( 5 - 6 )
maybe:   -1
logo
```

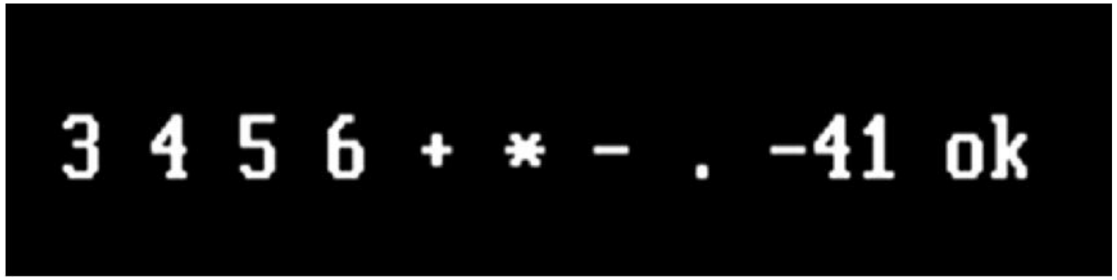
Flokc

"Empirical Modelling" of Logo in Forth

Institut für Philosophie Universität Potsdam



RPN (reverse Polish notation) of Forth

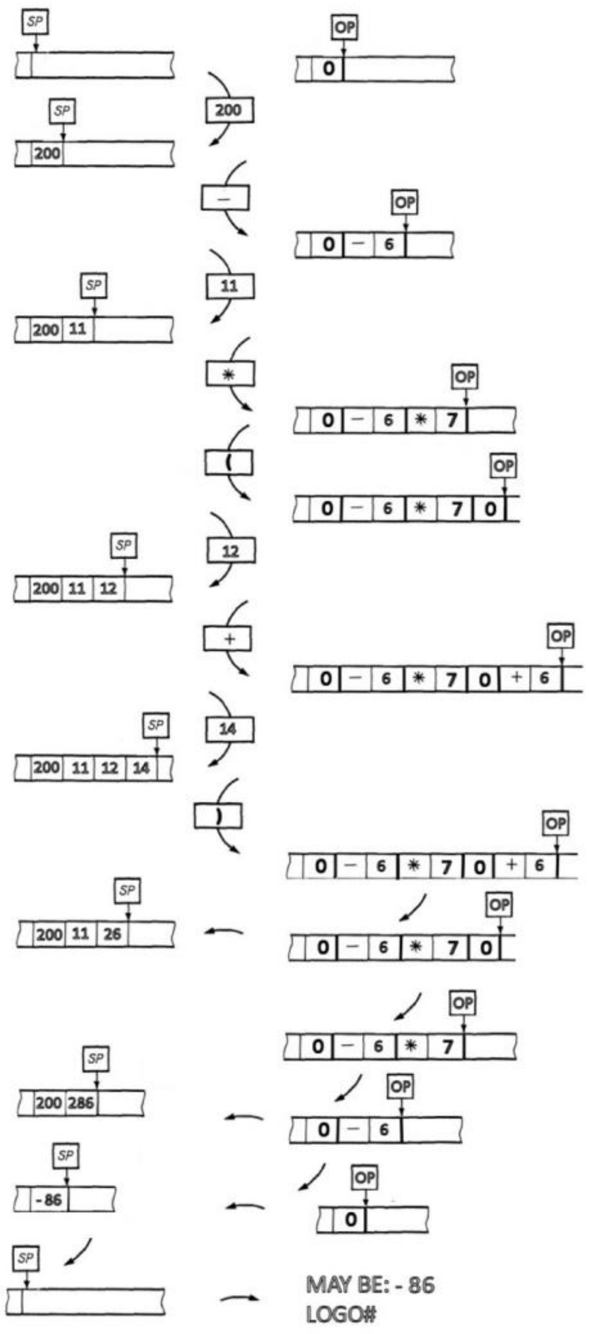


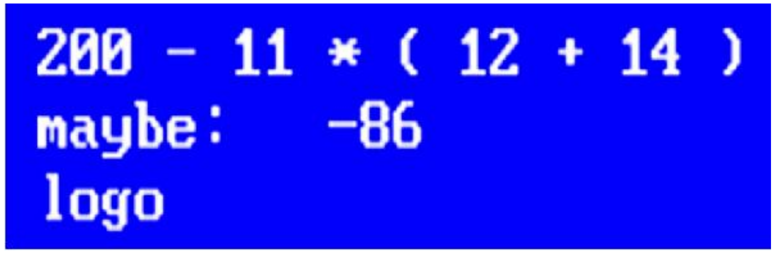
```
3 4 5 6 + * - . -41 ok
```

"Empirical Modelling" of Logo in Forth

Institut für Philosophie

Implementation of infix notation in Forth





```
200 - 11 * ( 12 + 14 )
maybe:   -86
logo
```

"Empirical Modelling" of Logo in Forth

Institut für Philosophie Universität Potsdam

TOP-DOWN-FORTH·

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

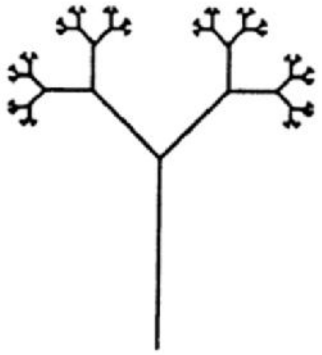
```
:   ?TOP CUH §
         IF TER? IM § IN ! BL WORD HERE CUH § (FIND)
            IF PUT ENDIF
         ENDIF ;
: TOP-LOAD IN-TOP LOAD TOP-TEST :
: VARIABLE ?IN VARIABLE ?TOP ;
: CONSTANT ?IN CONSTANT ?TOP ;
: CREATE ?IN CREATE SMUDGE ?TOP SMUDGE ;
: : ?IN (COMPILE) : SMUDGE ?TOP SMUDGE ; IMMEDIATE
: 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 )
```

"Empirical Modelling" of Logo in Forth

Institut für Philosophie
Universität Potsdam

Logo

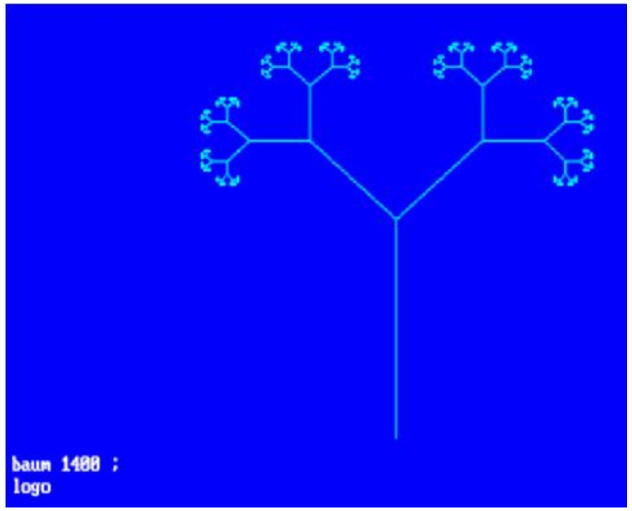
```
TO BAUM :X
IF :X < 1 [STOP]
FD :X LT 45 BAUM :X/2 RT 90
BAUM :X/2 LT 45 BK :X
END
```



BAUM 64

FLOKC

```
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
```



```
baum 1400 ;
logo
```

"Empirical Modelling" of Logo in Forth

(* 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 PRODUCT_RULE :FCT]
  IF :Z = "+  [OP SUM_RULE   :FCT]
  IF :Z = "o  [OP CHAIN_RULE   :FCT]
  IF :Z = "ex [OP POWER_RULE  :FCT]
END

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

(* 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 *)

I>  cos [ ? -sin ]      I> sin [ ? cos ]
I> -sin [ ? -cos ]      I> -cos [ ? sin ]
I>  exp [ ? exp  ]      I>  x  [ ? 1   ]
I>  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
```

Prof. Dr. Hans-Joachim Petsche

„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.“

Empirical Modelling is a way of construal making.

But what is modelling?

In general, computer modelling is used to solve three types of problems:

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

In general, computer modelling is used to solve three types of problems:

(1) **the analysis problem**: The **behaviour** of a 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.

In general, computer modelling is used to solve three types of problems:

(1) **the analysis problem**: The **behaviour** of a 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 a 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.

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.

An "object-to-think-with" can not only model an **empirically given** but also an **empirically wanted object**. Construal as an "object-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:
It 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.

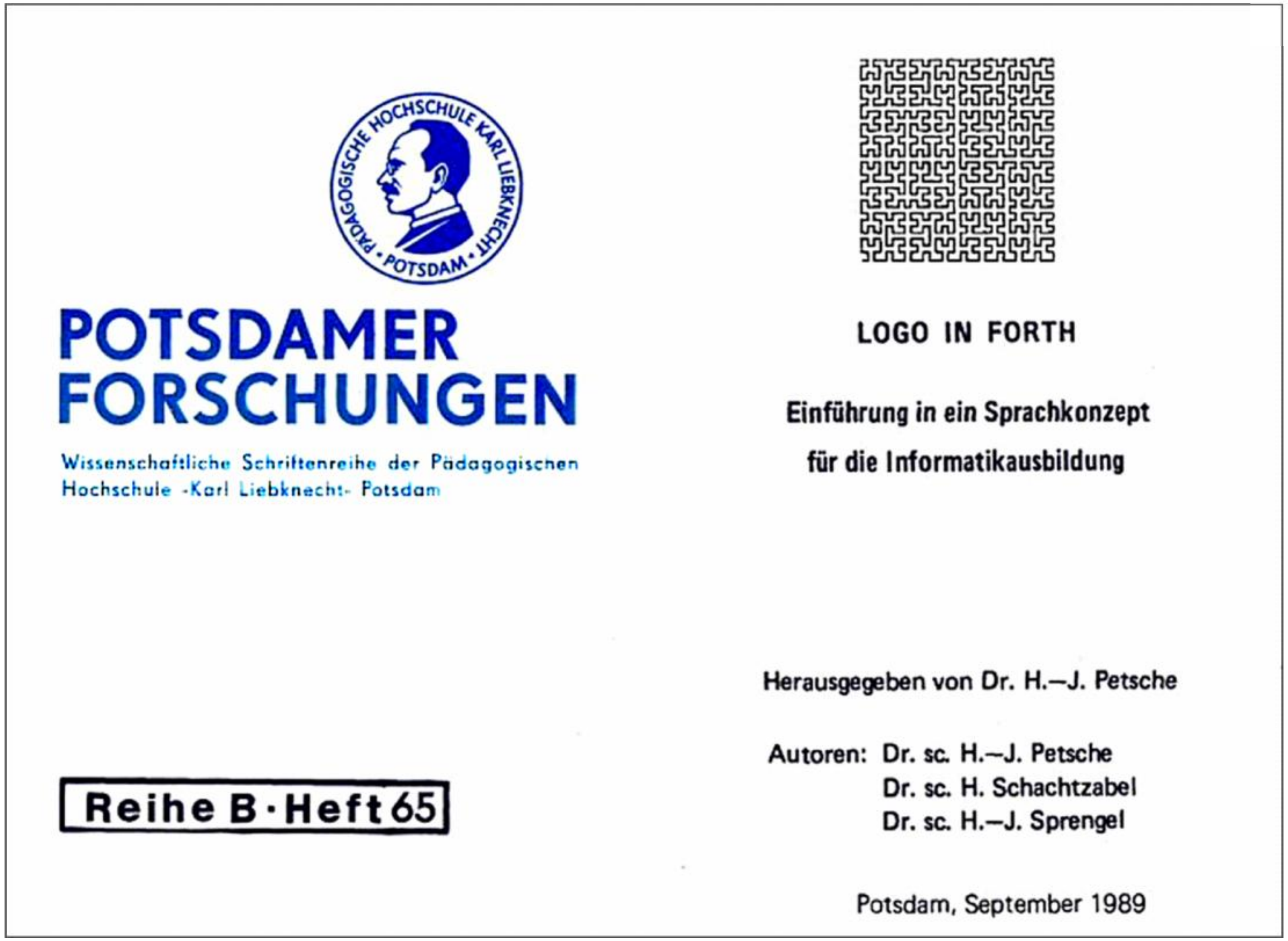
Modelling Logo in Forth is in my opinion an example for synthesising the behaviour of a desired object (a programming language) by observing the behaviour of the original object and by playing with agencies and dependencies.

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

"Empirical Modelling" of Logo in Forth

Institut für Philosophie

Universität Potsdam



PÄDAGOGISCHE HOCHSCHULE KARL LIEBKNECHT · POTSDAM ·

POTSDAMER FORSCHUNGEN

Wissenschaftliche Schriftenreihe der Pädagogischen Hochschule -Karl Liebknecht- Potsdam

Reihe B · Heft 65

LOGO IN FORTH

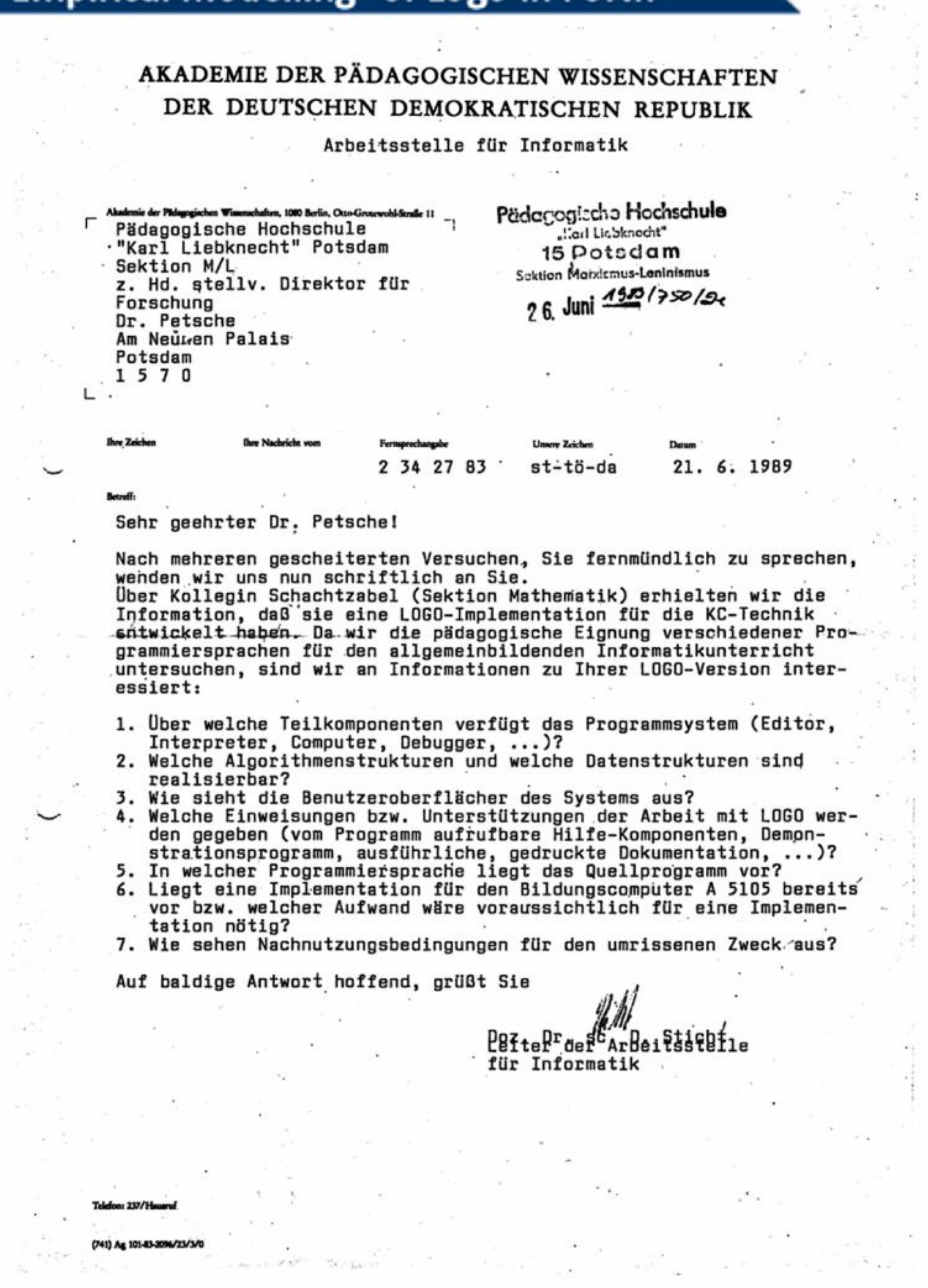
Einführung in ein Sprachkonzept für die Informatikausbildung

Herausgegeben von Dr. H.–J. Petsche

Autoren: Dr. sc. H.–J. Petsche
Dr. sc. H. Schachtzabel
Dr. sc. H.–J. Sprengel

Potsdam, September 1989

"Empirical Modelling" of Logo in Forth



AKADEMIE DER PÄDAGOGISCHEN WISSENSCHAFTEN
DER DEUTSCHEN DEMOKRATISCHEN REPUBLIK

Arbeitsstelle für Informatik

Akademie der Pädagogischen Wissenschaften, 1080 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 1989/750/9[illegible]

Ihre Zeichen	Ihre Nachricht vom	Fernsprechangabe	Unsere Zeichen	Datum
		2 34 27 83	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

[illegible]

Doz. Dr. sc. D. Sticht
Leiter der Arbeitsstelle
für Informatik

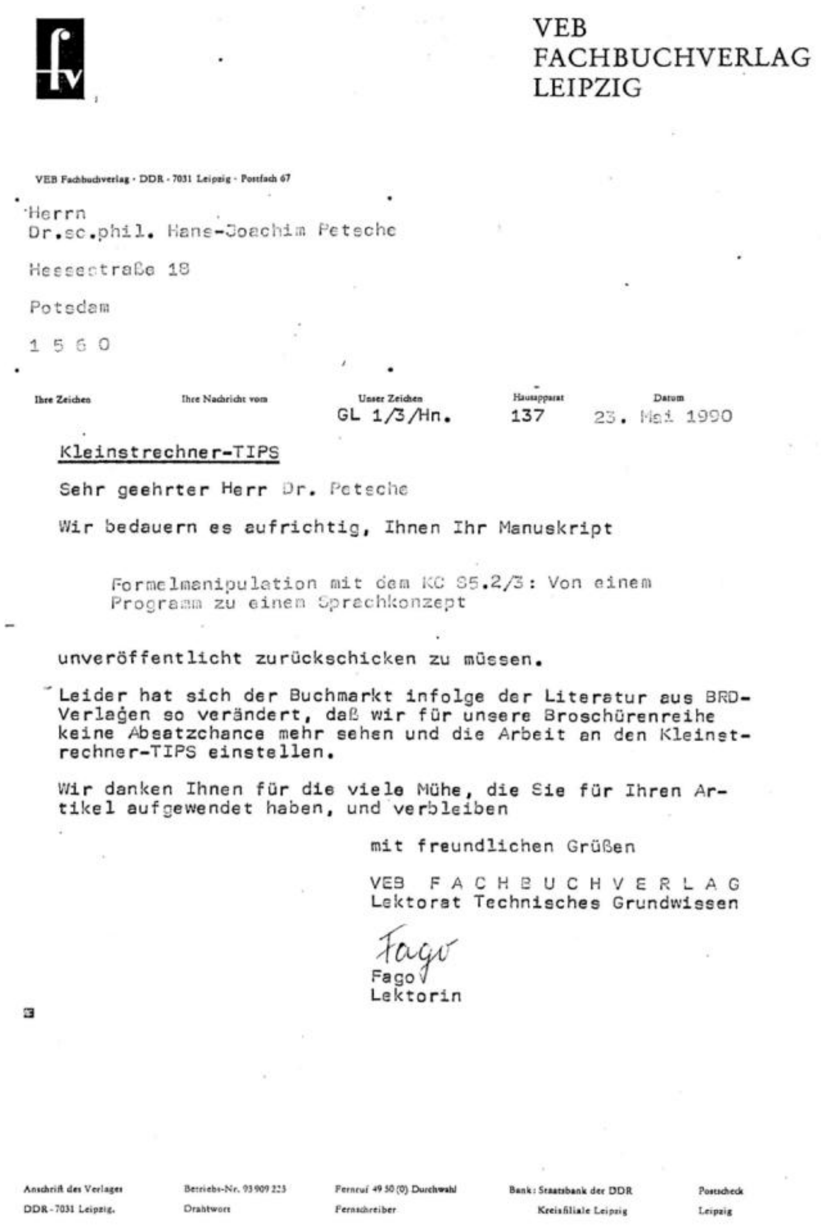
Telefon: 237/Hausruf

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

Institut für Philosophie
Universität Potsdam

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.

"Empirical Modelling" of Logo in Forth



VEB
FACHBUCHVERLAG
LEIPZIG

VEB Fachbuchverlag · DDR - 7031 Leipzig · Postfach 67

Herrn
Dr.sc.phil. Hans-Joachim Petsche

Hessestraße 18

Potsdam

1 5 6 0

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

<u>Kleinstrechner-TIPS</u>

Sehr geehrter Herr Dr. Petsche

Wir bedauern es aufrichtig, Ihnen Ihr Manuskript

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

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
Fago
Lektorin

Anschrift des Verlages	Betriebs-Nr. 93 909 223	Fernruf 49 50 (0) Durchwahl	Bank: Staatsbank der DDR	Postscheck
DDR - 7031 Leipzig.	Drahtwort	Fernschreiber	Kreisfiliale Leipzig	Leipzig

Institut für Philosophie
Universität Potsdam

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.

Supplement

Which programming language would I as a philosopher consider to be the best for children?

This seemed to be an easy question.

Institut für Philosophie Universität Potsdam

Supplement

Which programming language would I as a philosopher consider to be the best for children?

This seemed to be an easy question.

Martin Heidegger: “Language is the House of Being”.

What could that mean?

Supplement

Martin Heidegger: “Language is the House of Being”.

What could that mean?

- Language is not the construct of Being and also not the universe of Being. And language is not only a house of Being.

Supplement

Martin Heidegger: “Language is the House of Being”.

What could that mean?

- 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.

Supplement

Martin Heidegger: “Language is the House of Being”.

What could that mean?

- 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.

Institut für Philosophie

Supplement

Martin Heidegger: “Language is the House of Being”.

What could that mean?

- 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.
- SCRATCH for example is too LEGO-like, too superficially oriented on making (and constructing) and not on thinking, rethinking and problematising.

Supplement

Martin Heidegger: "Language is the House of Being".

What could that mean?

- 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.
- 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 it is not all?. So it seems to me, that a LOGO-like (LISP oriented) language could be a good compromise.

Institut für Philosophie Universität Potsdam

Supplement

Martin Heidegger: “Language is the House of Being”.

What could that mean?

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.

Its only the next step to find a good representation.

Faraday discovered such problems and found excellent representations!

Supplement

Martin Heidegger: “Language is the House of Being”.

What could that mean?

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.

Its 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)

The Picasso Construal

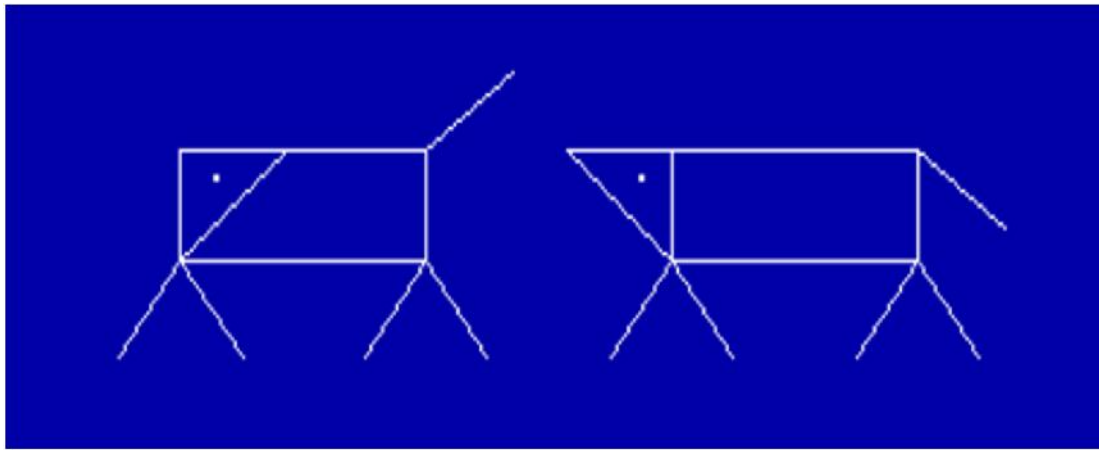
During his Cubist period, they ask Picasso if he can could express "affection" and "astonishment" with only a few brushstrokes.

He could indeed!

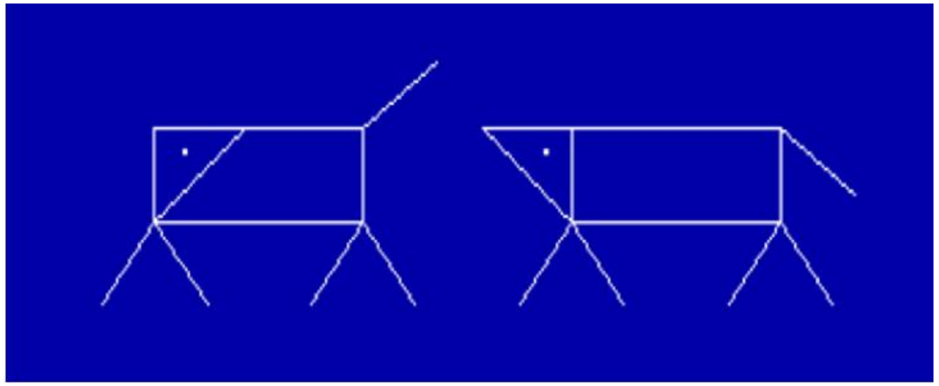
The Picasso Construal

During his Cubist period, they ask Picasso if he could express "affection" and "astonishment" with only a few brushstrokes.

He could indeed!

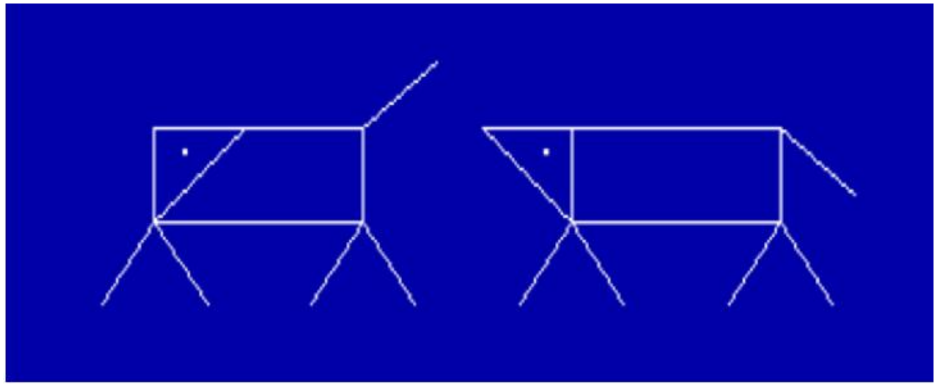


The Picasso Construal



- Picasso modelled human behaviour through the behaviour of dogs, which for their part were modelled by some points, triangles, and rectangles. That was ingenious.

The Picasso Construal

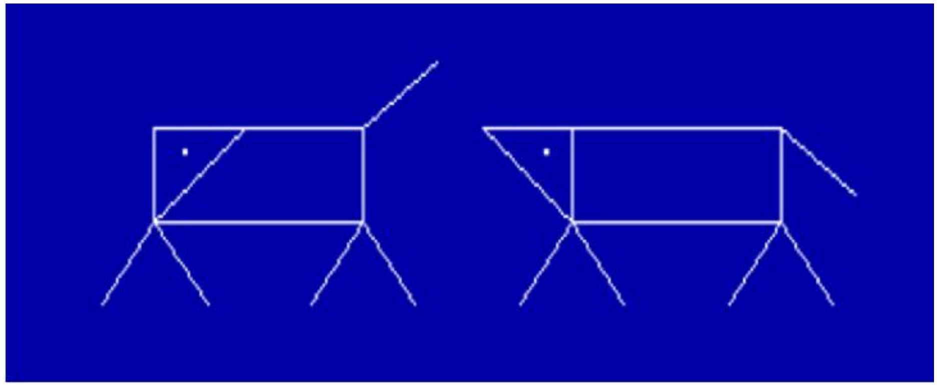


- Picasso modelled human behaviour through the behaviour of dogs, which for their part were modelled by some points, triangles, and rectangles. That was ingenious.
- And he gave us a fantastic problem for empirical modelling: Which other kinds of human behaviour could be expressed through modelling of dogs by points, triangles, and rectangles. And what will occur if we add some kind of motion?

Institut für Philosophie

Universität Potsdam

The Picasso Construal



- Picasso modelled human behaviour through the behaviour of dogs which for their part were modelled by some 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 by points, triangles, and rectangles. And what will occur if we add some kind of motion?
- These construals, made by a purist program, generate visual abstractions of human behaviour which gives us good things to think with.

So we need more good problems to make construals on computers which give us more good things to think with.

"Empirical Modelling" of Logo in Forth

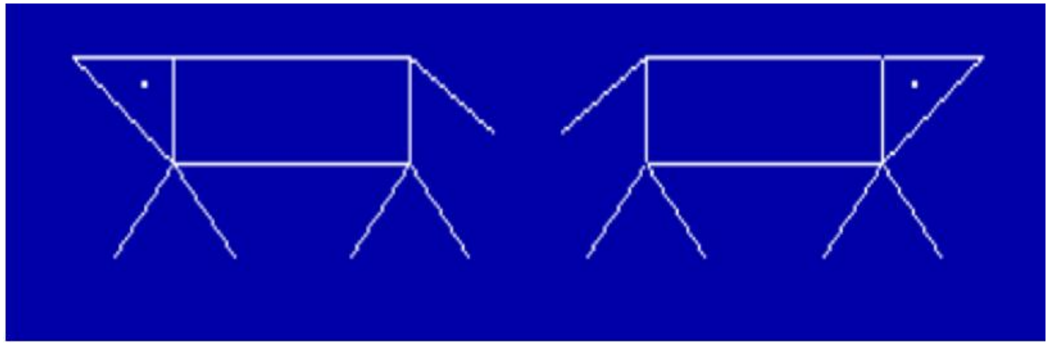
Institut für Philosophie

Thank you for your

Attention!

Prof. Dr. Hans-Joachim Petsche

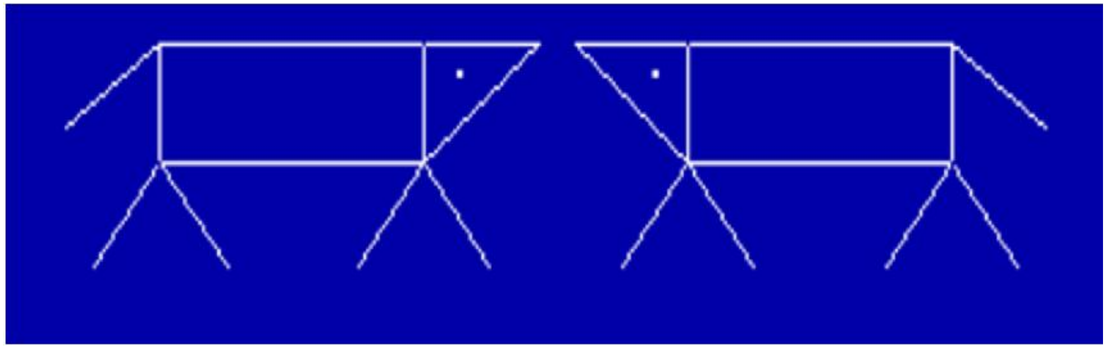
The Picasso Construal



"Disinterest"

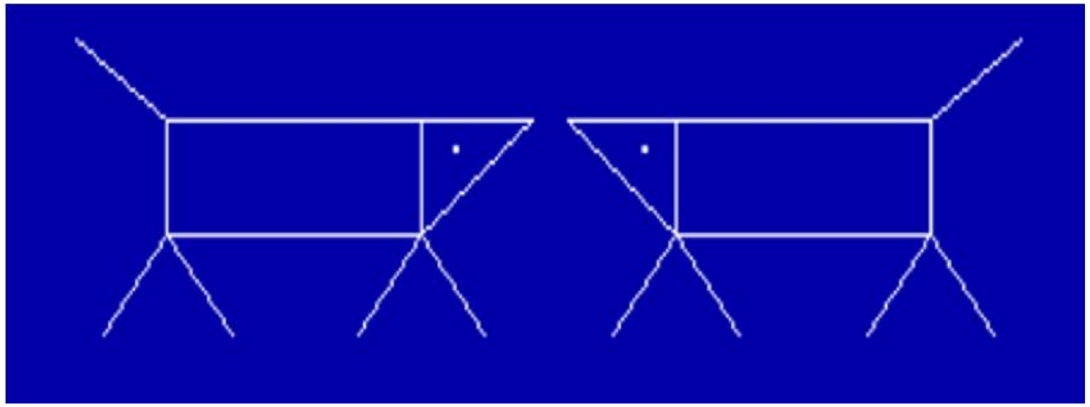
Institut für Philosophie

The Picasso Construal



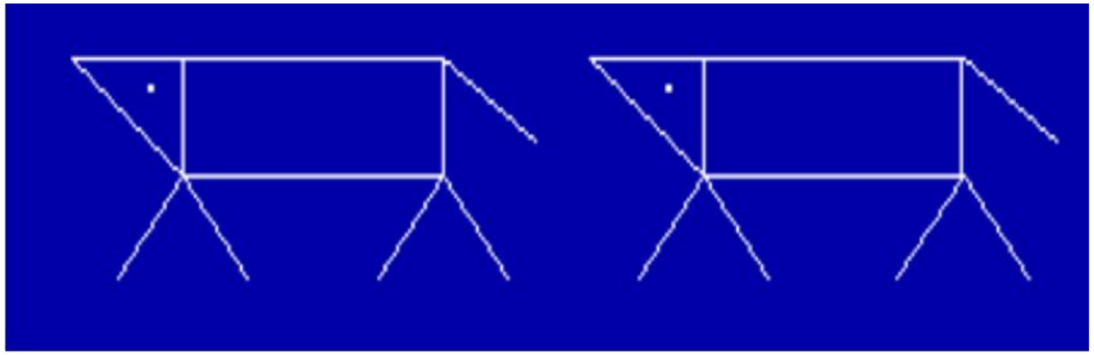
"Inspect"

The Picasso Construal



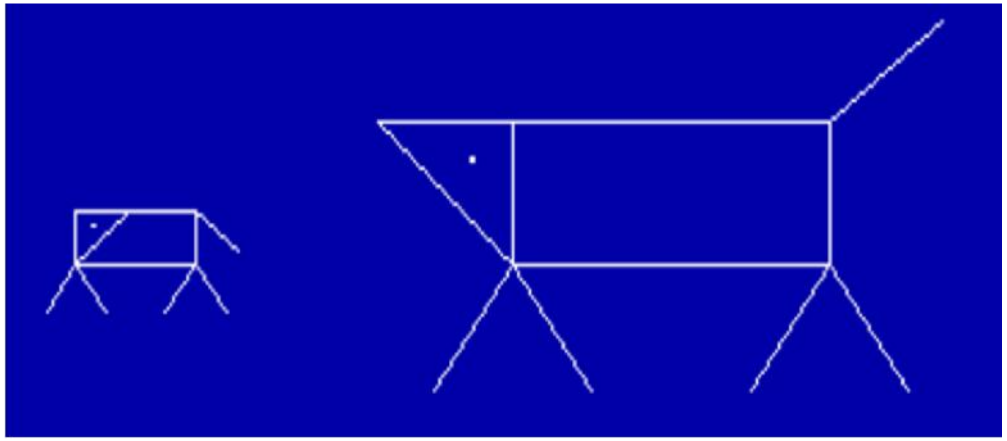
"Joy"

The Picasso Construal



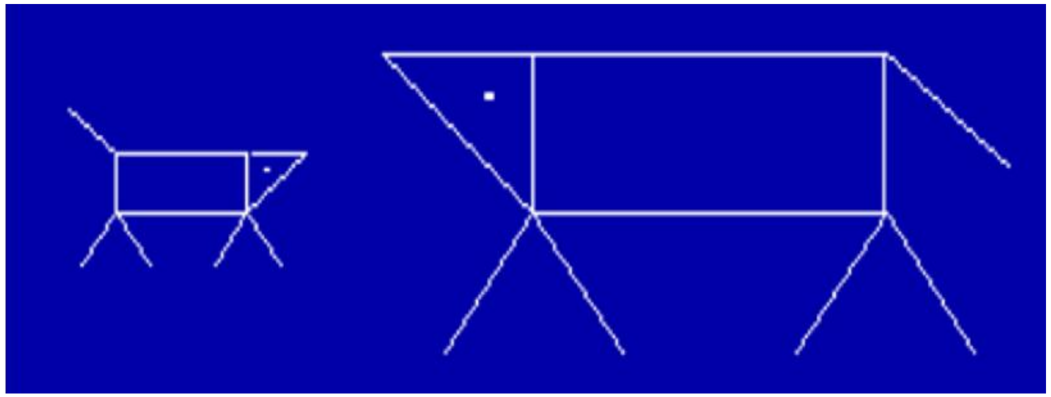
"Unanimity"

The Picasso Construal



"Fear"

The Picasso Construal



"Courage"

Institut für Philosophie

Universität Potsdam

Thank you for your

Attention!

"Empirical Modelling" of Logo in Forth

(* 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
```

(* Symbolic Differentiation of a Function in FLOKC *)

```
GLO F LOK FCT LIS FKT

TO ABL FKT ; (* Start program *)
   CR MAKE FCT " FKT ;  Abl FCT ;
END

TO Abl FCT ; (* Differentiation *)
   IF  LISTP FCT      =0 THEN ? 0            STOP ENDIF
   IF ( COUNT FCT ) = 1    THEN EXEC FCT      STOP ENDIF
   IF ( COUNT FCT ) = 3 =0 THEN LP FCT ? BAD! STOP ENDIF
   "( EXEC ( ITEM 2 FCT ) )"
END

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 o ;    (* Chain rule *)
   "( Abl FIRST FCT ; ? o Op  LAST FCT ;  )"
   ? * Abl LAST FCT ;
END

TO * ;    (* Product rule *)
   Op FIRST FCT ;  ? * Abl LAST FCT ;
   ? + Abl FIRST FCT ; ? * Op LAST FCT ;
END

TO + ;    (* Sum rule *)
   Abl FIRST FCT ; ? + Abl  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" * ID ex "  PR ( F - 1 )
END

TO Op F ; (* Output *)
   IF LISTP F THEN IF ( COUNT F ) > 1 THEN LP F STOP ENDIF
   ENDIF PR F
END
```

Prof. Dr. Hans-Joachim Petsche

"Empirical Modelling" of Logo in Forth

```
(* 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
```

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

GLO F LOK FCT LIS FKT

L> ID   << ID >>     L> EXP     << EXP >>     L> LN      << LN >>
L> SIN  << SIN >>    L> COS     << COS >>     L> -SIN    << -SIN >>
L> -COS << -COS >>   L> o       << >>         L> ex      << >>

TO Chain_rule ;
   P" ( " AbL FIRST FCT ; P" o " Op LAST FCT ; P" )
   P" * " AbL LAST FCT ;
END

TO Product_rule ;
   Op FIRST FCT ;  P"  * "  AbL LAST FCT ;
   P" + "  AbL FIRST FCT ; P"  * "  Op LAST FCT ;
END

TO Sum_rule ;
   AbL FIRST FCT ;  P"  + "  AbL  LAST FCT ;
END

TO Power_rule ; MAKE F LAST FCT ;
   IF LISTP F =0
      THEN IF F =0
              THEN P" 0 " STOP
              ELSE Op F ; P" * ID ex "  Op F - 1 ;
           ENDIF
      ELSE Op FCT ; P" BAD!" STOP
   ENDIF
END

TO ELEM ;
   IF F =   " ID   THEN P"   1 "    ENDIF
   IF F =   " SIN  THEN P"   COS "  ENDIF
   IF F =   " COS  THEN P"   -SIN " ENDIF
   IF F =   " -SIN THEN P"   -COS " ENDIF
   IF F =   " -COS THEN P"     SIN " ENDIF
   IF F =   " EXP  THEN P"   EXP "     ENDIF
   IF F =   " LN   THEN P"   ( ID ex -1 ) " ENDIF
END

TO ABL FKT ;  MAKE FCT " FKT ;  CR Abl FCT ;
END

TO Abl FCT ;-|
   IF LISTP FCT =0 THEN P" 0 " STOP ENDIF
   IF ( COUNT FCT ) = 1
      THEN MAKE F FIRST FCT ; ELEM STOP
      ELSE IF ( COUNT FCT ) = 2
           THEN LP FCT P" BAD!" STOP
      ENDIF
   ENDIF
   MAKE F ITEM 2 FCT ;
   P" ( "
   IF F = " * THEN PRODUKTREGEL ELSE
   IF F = " + THEN SUMMENREGEL  ELSE
   IF F = " o THEN KETTENREGEL  ELSE
   IF F = " ex THEN POTENZREGEL
   ENDIF ENDIF ENDIF ENDIF P" ) "
END

TO Op F ;
   IF LISTP F THEN LP F ELSE PR F ENDIF
END
```