Vierte Dimension

2/2012

Das Forth–Magazin

für Wissenschaft und Technik, für kommerzielle EDV, für MSR-Technik, für den interessierten Hobbyisten

In dieser Ausgabe:

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

tematik GmbH
Technische Informatik

Feldstrasse 143
D–22880 Wedel
Fon 04103 – 808989 – 0
Fax 04103 – 808989 – 9
mail@tematik.de
www.tematik.de

Gegründet 1985 als Partnerinstitut der FH–Wedel beschäftigten wir uns in den letzten Jahren vorwiegend mit Industrieelektronik und Präzisionsmeßtechnik und bauen z. Z. eine eigene Produktpalette auf.

Know–How Schwerpunkte liegen in den Bereichen Industriewaagen SWA & SWW, Differential–Dosierwaagen, DMS–Messverstärker, 68000 und 68HC11 Prozessoren, Sigma–Delta A/D. Wir programmieren in Pascal, C und Forth auf SwiftX86k und seit kurzem mit Holon11 und MPE IRTC für Amtel AVR.

LEGO RCX–Verleih

Seit unserem Gewinn (VD 1/2001 S.30) verfügt unsere Schule über so ausreichend viele RCX–Komponenten, dass ich meine privat eingebrachten Dinge nun Anderen, vorzugsweise Mitgliedern der Forth–Gesellschaft e. V., zur Verfügung stellen kann.

Angeboten wird: Ein komplettes LEGO–RCX–Set, so wie es für ca. 230,– € im Handel zu erwerben ist.

Inhalt:

1 RCX, 1 Sendeturm, 2 Motoren, 4 Sensoren und ca. 1.000 LEGO Steine.

Anfragen bitte an
Martin.Bitter@t-online.de

Letztlich enthält das Ganze auch nicht mehr als einen Mikrocontroller der Familie H8/300 von Hitachi, ein paar Treiber und etwas Peripherie. Zudem: dieses Teil ist „narrensicher“!

RetroForth
Linux · Windows · Native
Generic · L4Ka::Pistachio · Dex4u
Public Domain
`http://www.retroforth.org`
`http://retro.tunes.org`

Diese Anzeige wird gesponsort von:
EDV–Beratung Schmiedl, Am Bräuweiher 4, 93499 Zandt

Hier könnte Ihre Anzeige stehen!

Wenn Sie ein Förderer der Forth–Gesellschaft e.V. sind oder werden möchten, sprechen Sie mit dem Forth–Büro über die Konditionen einer festen Anzeige.

`Secretary@forth-ev.de`

KIMA Echtzeitsysteme GmbH

Tel.: 02461/690–380
Fax: 02461/690–387 oder –100
Karl–Heinz–Beckurts–Str. 13
52428 Jülich

Automatisierungstechnik: Fortgeschrittene Steuerungen für die Verfahrenstechnik, Schaltanlagenbau, Projektierung, Sensorik, Maschinenüberwachungen. Echtzeitrechnersysteme: für Werkzeug– und Sondermaschinen, Fuzzy Logic.

FORTecH Software GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

Bergstraße 10	D–18057 Rostock
Tel.: +49 381 496800-0	Fax: +49 381 496800-29

PC–basierte Forth–Entwicklungswerkzeuge, comFORTH für Windows und eingebettete und verteilte Systeme. Softwareentwicklung für Windows und Mikrocontroller mit Forth, C/C++, Delphi und Basic. Entwicklung von Gerätetreibern und Kommunikationssoftware für Windows 3.1, Windows95 und WindowsNT. Beratung zu Software–/Systementwurf. Mehr als 15 Jahre Erfahrung.

Hier könnte Ihre Anzeige stehen!

Wenn Sie ein Förderer der Forth–Gesellschaft e.V. sind oder werden möchten, sprechen Sie mit dem Forth–Büro über die Konditionen einer festen Anzeige.

`Secretary@forth-ev.de`

Ingenieurbüro
Klaus Kohl–Schöpe

Tel.: 07044/908789
Buchenweg 11
D–71299 Wimsheim

FORTH–Software (volksFORTH, KKFORTH und viele PDVersionen). FORTH–Hardware (z.B. Super8) und Literaturservice. Professionelle Entwicklung für Steuerungs– und Meßtechnik.

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 32 01 24
68273 Mannheim
Tel: ++49(0)6239 9201-85, Fax: -86
E–Mail: `Secretary@forth-ev.de`
`Direktorium@forth-ev.de`
Bankverbindung: Postbank Hamburg
BLZ 200 100 20
Kto 563 211 208
IBAN: DE60 2001 0020 0563 2112 08
BIC: PBNKDEFF

Redaktion & Layout

Bernd Paysan, Ulrich Hoffmann
E–Mail: `4d@forth-ev.de`

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

Januar, April, Juli, Oktober jeweils in der dritten Woche

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00€ + Porto u. Verpackung

Liebe Leser,

Die Forth–Gesellschaft war auf dem LinuxTag — dieses Jahr hatten wir genügend Helfer: Erich Wälde, Uli Hoffmann, David Kühling, Carsten Strotmann, Gido Baumann, Friedel Amend, ich selbst und sogar Claus Vogt war kurzzeitig Standpersonal. Unser neuer Roboter war zwar eine Attraktion, aber die gruselig süßen Roboter am Stand daneben waren noch attraktiver. Ok, die waren auch Selbstzweck, während unser Roboter ja lediglich Werbung für die Sprache Forth machen soll, und dass man mit knapp mehr als 2 kB Programmcode so einen Roboter auch steuern kann. Mit etwas komplizierterer Trigonometrie, aber davon weiter hinten im Heft mehr.

Das andere Ergebnis des LinuxTags: die Sommerkollektion der FG ist da: neue T–Shirts. Heinz Schnitters T–Shirts sind längst verteilt und haben sich zum Teil schon aufgelöst (so behaupten zumindest Anekdoten). Bei Heinz hat noch der nach Erdbeer duftende Plastik–Swap aus Hongkong Modell gestanden, bei mir ein Drachen–T–Shirt aus Shanghai. Natürlich musste noch ein zweiter Kopf 'ran, sonst kann er ja nicht swappen. Und Drachen, so habe ich mir sagen lassen, sind eigentlich golden oder feuerrot, jedenfalls chinesische. Naja, dieser ist blau. Und vom Siebdruck sind wir gleich auf Direct–on–Garment–Druck umgestiegen, also Tintenstrahldrucker mit Textilfarben. Das ermöglicht dann auch „unmögliche" Motive selbst auf schwarzen T–Shirts. Oder was auch immer man bedrucken will, der Textildrucker hat einen großen Katalog (für den 25.5.: Handtücher).

Natürlich stellt sich einem beim Besuch des LinuxTages die Frage, wie lang es so eine Präsenzmesse noch gibt, und welchen Sinn sie heute noch macht, schließlich kann man sich im Internet auch gut über Linux/freie Software und die aktuellen Entwicklungen informieren, und zwar nicht jährlich, sondern wöchentlich/täglich (z.B. auf `lwn.net`). Es sind schon prominentere Messekonzepte wie die Systems untergegangen. Unser derzeitiges Ausstellerkonzept ist sehr hardware- und controllerlastig; beim nächsten Mal sollten wir auch etwas mehr größere Systeme vorführen. Der Roboter mit Kinect–Kamera als interaktiver Mühle– oder Go–Spieler wird einen Teil hergeben, das Android–Gforth könnte man auf einem Tablet präsentieren — Android ist die Linux–Variante, die tatsächlich beim Consumer angekommen ist.

Ansonsten: Beim Überlegen, wie man Gforth mit einem smarten COMPILE, ausstattet, hat sich eine Diskussion über die verschiedenen Semantics des ANS Forth–Standards ergeben, die zeigt, dass da noch etwas konzeptioneller Klärungsbedarf vorhanden ist.

Erich Wälde hat nicht nur den LinuxTag–Stand bemannt, sondern auch fleißig geschrieben, über State–Maschinen und mehr. Und Johannes Teichs Wave Engine spielt im dritten Teil schon sehr virtuos.

Ach ja: Wir haben die Versionsverwaltung auf Fossil umgestellt. Das soll es dem wechselnden Redakteur ermöglichen, mit einer kompakten Template anzufangen, und nicht erst die ganze Versionsgeschichte seit 2006 herunterladen zu müssen. Und man kann endlich mit pdflatex TEXen.

Und so übergebe ich dann den Stab an Martin Bitter...

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
`http://fossil.forth-ev.de/vd-2012-02`

Die Forth–Gesellschaft e. V. wird durch ihr Direktorium vertreten:

Ulrich Hoffmann Kontakt: `Direktorium@Forth-ev.de`
Bernd Paysan
Ewald Rieger

Forth — ein etwas ausgefallenes Programmiersystem

Sehr geehrter Herr Behringer,

ich bin der Wikipedia–Co–Autor, der sich erdreistet hat, Forth als „etwas ausgefallenes Programmiersystem“ zu bezeichnen. Für einen FORTH–Enthusiasten mag das zwar „gotteslästerlich“ klingen, aber es entspricht wohl objektiv den Tatsachen, wenn man davon ausgeht, dass geschätzte 0,01% der weltweiten Programmierer FORTH nutzen.

Wir verwendeten für Mikrocontroller–Programmierung schon zu MS–DOS–Zeiten einen FORTH–Abkömmling, der FIFTH genannt wurde und der von Prof. Dr. Mayer–Lindenberg von der TU Harburg entwickelt worden ist. FIFTH war ein sehr leistungsfähiges Host/Target–Entwicklungssystem mit einem integrierten, sehr originellen Editor, der trotz DOS-Betriebssystem sehr gut zu handhaben war. FIFTH wich leider in einigen Punkten — unnötigerweise — vom FORTH–Standard ab und war schlecht dokumentiert.

Inzwischen verwenden wir SwiftX (von FORTH Inc.) für die Mikrocontroller–Familie MSP430 von Texas Instruments und sind damit sehr zufrieden. SwiftX ist hervorragend dokumentiert und hat einen guten Support. Für eingebettete Steuerungen gibt es meiner Meinung nach nichts besseres als SwiftX bzw. FORTH.

Mit freundlichen Grüßen

Glasmacher

Die Redaktion weiß zwar auch nicht, wie viele Forth–Programmierer es gibt, aber insgesamt gibt es ca. 15 Millionen Programmierer — 0,01% davon wären dann nur 1500, das erscheint dann doch etwas wenig.

VT100-Terminal-Emulation für iMac

Jan Kromhout schrieb auf der amForth-Mailing-Liste `Amforth-devel@lists.sourceforge.net`:

```
Von: Jan Kromhout <krom1109@hotmail.com>
Betreff: [Amforth] VT100 terminal IMAC
Datum: 2. Mai 2012 22:21:30 MESZ
An: Everything around amforth

Hallo,
ich suche einen Freeware-VT100-Terminal-
Emulator für den IMAC.
Könnt Ihr mir ein paar Vorschläge machen?

Grüße,

Jan kromhout
```

Das ist eine Frage, die häufig gestellt wird, und sicher auch unsere Leser mit Macs interessiert.

Ulrich Hoffmann (u. a.) antwortete:

```
Hallo Jan,

es gibt mehrere Möglichkeiten der Terminal-
Unterstützung in Mac OS X.

Zunächst mal kannst Du Kommandozeilenprogramme
verwenden, wie das gute alte Kermit [1] (das
benutz ich) oder screen [2] oder, wie bereits
schon erwähnt, minicom [3].

Und dann gibt es auch noch GUI-Programme für
OS X. Klassisch ist ZTERM [4], aber Du kannst
auch unter einer Reihe verschiedener Cocoa-
basierter Anwendungen wählen:
Cornflake [5], CoolTerm [6] or goSerial [7].

Alle haben mehr oder weniger den gleichen
Funktionsumfang. Schau Sie Dir an und wähle
selbst. Vielleicht willst Du uns dann wissen
lassen, was Dir am besten passt.

Du kannst auch noch auf Carsten Strotmanns
Terminal Emulatoren für Forth Systeme [8]
schauen.

Viele Grüße,
Ulli

Links:

[1] http://www.kermitproject.org/ck90.html
[2] http://etherealmind.com/serial-console-on-osx/
[3] http://turin.nss.udel.edu/programming/
[4] http://homepage.mac.com/dalverson/zterm/
[5] http://tomgerhardt.com/Cornflake/
[6] http://freeware.the-meiers.org/
[7] http://www.furrysoft.de/?page=goserial
[8] http://strotmann.de/roller/cas/entry/terminal_
    emulatoren_f%C3%BCr_forth_systeme
```

uho

Adventures 11: Finite State Machine

Erich Wälde

In einem nicht–forthigen Zusammenhang sind mir neulich endliche Automaten *(finite state machines) über den Weg gelaufen. Da erinnerte ich mich daran, dass ich noch ein Dokument hatte, welches ich irgendwann einmal lesen wollte:* Julian V. Noble — Finite State Machines in Forth *[2]. Das traf sich bestens, denn so hatte ich einen Grund, den darin vorgestellten Code auf amforth [1] zu portieren.*

Eingabefunktion für Dezimalzahlen

Eine Dezimalzahl soll folgenden Regeln in der Darstellung genügen:

1. Nur die Zeichen `0123456789.-` sind erlaubt
2. Das erste Zeichen darf eine Ziffer, das negative Vorzeichen `-` sowie der Dezimalpunkt `.` sein
3. Nach dem ersten Zeichen ist kein Vorzeichen `-` mehr erlaubt
4. Nach dem ersten Dezimalpunkt `.` sind keine weiteren Dezimalpunkte erlaubt

Also sind beispielsweise folgende Zahlen gültig: `0.123 .123 1.23 -1.23 123 -.123`.

Die Programmieraufgabe besteht nun darin, die Eingabe solcher Dezimalzahlen zu ermöglichen. Allerdings soll z. B. nach dem ersten Dezimalpunkt kein weiterer mehr angenommen werden. Die Eingaberoutine ist zu jeder Zeit über die weiterhin erlaubten Zeichen im Bild. Das Drücken der Eingabetaste beendet die Routine. Selbstverständlich kann dieses Problem auch prozedural gelöst werden (siehe Anhang Listings). Allerdings sieht der Kern der Angelegenheit, das Wort `legal?` doch einigermaßen unübersichtlich aus.

```
include tempbuffer.fs \ reset append show
variable prev.minus?
variable prev.dpoint?

: digit?   [char] 0  [char] 9  within ;
: dpoint?  [char] . = ;
: minus?   [char] - = ;
: first.minus?
  minus?  prev.minus? @ not  and ;
: first.dpoint?
  dpoint?  prev.dpoint? @ not  and ;
: false true not ;

: legal? ( c -- f )
  dup digit? if
    \ no '-' allowed after first digit
    drop true dup p.minus? !
  else dup
    first.minus? if
      drop true dup p.minus? !
    else
      first.dpoint? if
        true dup p.dpoint? !
      else
        false
      then
    then
  then
;
```

Es handelt sich um eine geschachtelte `if` — `else` — `then`–Konstruktion, welche die komplette Logik enthält. Dabei sollte man beachten, dass es sich hier um ein ziemlich einfaches Problem handelt. Bei einem komplexeren Problem wird das schnell sehr unübersichtlich.

Das Wort `legal?` wird in der Eingabeschleife für jedes Zeichen aufgerufen.

```
: getafix
  reset
  false p.minus? !
  false p.dpoint? !
  begin
    key
    dup emit
    dup $0D <> \ exit on CR
  while
      dup legal? if append else drop then
  repeat
  drop \ CR
  cr show cr
;
```

Statt der direkten Ausgabe (`emit`) habe ich einen Puffer angelegt, in den die gültigen Zeichen hineinkopiert werden (`append`). Nach der Eingabe wird die akzeptierte Zeichenkette ausgegeben. So sieht man sowohl die Eingabe als auch, was davon übrig geblieben ist.

```
> getafix
---234900x9er.2-304
-2349009.2304
 ok
```

Endlicher Automat

Diese Art Probleme können sehr schön mit einem endlichen Automaten (*finite state machine*) gelöst werden. Die Zustände des Automaten können als Gedächtnis benutzt werden: Im Zustand `0` wurde noch kein Zeichen verarbeitet. Ist das erste Zeichen gültig, dann wird dadurch der Übergang in die Zustände `1` (keine Minuszeichen erlaubt) oder `2` (zusätzlich keine Dezimalpunkte erlaubt) ausgelöst. In der nachfolgenden Tabelle steht ein `X` in den Feldern, in denen das eingegebene Zeichen nicht erlaubt ist und folglich verworfen wird. Ein `E` steht für die Ausgabe des Zeichens — ursprünglich ein `emit`, in diesen Beispielen durch ein `append` ersetzt. Hinter den Aktionen `X`

und E sind die gewünschten nachfolgenden Zustände angegeben. Die Eingabe eines Dezimalpunkts (Spalte DP?) führt immer zu einem Übergang in den Zustand 2.

```
\ Input: other?     digit?     minus?     dpoint?
\ State| Does Trans Does Trans Does Trans Does Trans
\ 0    |  X   -> 0   E   -> 1   E   -> 1   E   -> 2
\ 1    |  X   -> 1   E   -> 1   X   -> 1   E   -> 2
\ 2    |  X   -> 2   E   -> 2   X   -> 2   X   -> 2
\
\ E == echo or append
\ X == drop
```

Tabelle 1: Zustände und Übergänge des endlichen Automaten

Der Zustand 0 ist der initiale Zustand. Wird eine Ziffer oder ein (negatives) Vorzeichen eingegeben, dann wechselt der Automat in den Zustand 1. Der Zustand 1 sagt sinngemäß: *Es wurde schon ein Zeichen eingegeben, ein weiteres Vorzeichen ist jetzt nicht mehr erlaubt.*

Wird ein Dezimalpunkt eingegeben, dann wechselt der Automat auf jeden Fall in den Zustand 2. Dieser heißt sinngemäß: *Es ist kein weiterer Dezimalpunkt erlaubt.*

Man kann in der Tabelle sehen, dass in allen Zuständen ein ungültiges Zeichen schlicht verworfen wird (Spalte other? enthält nur X), der Zustand ändert sich nicht. Ebenso kann man sehen, dass ein Vorzeichen nur in Zustand 0 erlaubt ist, also als erstes Zeichen (Spalte minus?). Der Zustand 0 wird immer nach dem ersten gültigen Zeichen verlassen.

Im Folgenden benutze ich die Abkürzung FSM, um den endlichen Automaten zu benennen.

Version 1

Wir brauchen für die erste Version der FSM das Wort case, welches seinerseits postpone verlangt. Die Tests der Zeichen sind gleich geblieben.

```
include lib/ans94/postpone.frt
include case.fs
include tempbuffer.fs

variable mystate
: digit?   [char] 0  [char] 9  within ;
: dpoint?  [char] . = ;
: minus?   [char] - = ;
```

Das Wissen, welches zuvor in dem Wort legal? versammelt war, wird in drei verschiedene Worte aufgeteilt. Jeder Zustand bekommt ein eigenes Wort, welches sämtliche Aktionen und Übergänge dieses einen Zustands kennt.

```
: (0) ( c -- )
  dup digit? over minus? or
  if         append   1 mystate !
  else
    dup dpoint?
    if       append   2 mystate !
    else
      drop
    then
  then
;
: (1) ( c -- )
  dup digit?
  if         append ( 1 mystate ! )
  else
    dup minus?
    if       drop   ( 1 mystate ! )
    else
      dup dpoint?
      if     append   2 mystate !
      else
drop
      then
    then
  then
;
: (2) ( c -- )
  dup digit?
  if         append ( 2 mystate ! )
  else
             drop
  then
;
```

Superübersichtlich ist das vielleicht auch noch nicht, aber mit etwas Einrücken kann man zumindest die Aktionen und die gewünschten Übergänge sehen. Der Vergleich mit der Tabelle fällt jetzt leichter. Diese Worte werden in einen case–Block gesteckt, welcher dann die FSM bildet.

```
: <fixed.point#> ( char state -- )
  case
    0 of   (0) endof
    1 of   (1) endof
    2 of   (2) endof
  default:
  endcase
;
```

In der Schleife von getafix wird für jedes eingegebene Zeichen der momentane Zustand auf den Stapel gelegt und dann die FSM aufgerufen.

```
: getafix
  reset
  0 mystate !
  begin
    key
    dup emit
    dup $0D <> \ exit on CR
  while
      mystate @  <fixed.point#>
  repeat
  drop \ CR
  cr show cr
;
```

J. Noble nennt diese Version der FSM *brute force fsm*. Zwar funktioniert das Programm, aber schön ist diese Lösung noch nicht.

```
> getafix
0234.234
0234.234
 ok
```

```
> getafix
-324.234
-324.234
 ok
> getafix
0-.234.234-234
0.234234234
 ok
```

Version 2

Der oben gezeigte `case`–Block enthält eine Zeile für jeden Zustand der FSM. Für die Übereinstimmung mit der Tabelle müssen die Worte `(0)` etc. einzeln inspiziert werden. Eine Definition, welche die Tabelle in lesbarer Form im Quelltext enthält, wäre einfacher zu verstehen.

Um das zu realisieren, schreiben wir zunächst eine Funktion, die zu einem beliebigen Zeichen die Nummer der zugehörigen Tabellenspalte (`other? digit? minus? dpoint?`) berechnet. Die genaue Reihenfolge der Spalten wird in diesem Wort festgelegt.

```
: cat->col# ( n -- n' )
  \     other?   0
  dup   digit?   1 and
  over  minus?   2 and +
  swap  dpoint?  3 and +
;

> char a cat->col# .
0  ok
> char 3 cat->col# .
1  ok
> char - cat->col# .
2  ok
> char . cat->col# .
3  ok
```

Die neue FSM wird durch ein Definitions–Wort generiert. Zuerst wird ein Eintrag im Wörterbuch für die zu generierende FSM angelegt. Die Anzahl der Spalten in der Tabelle wird an dieser Stelle vom Programmierer übergeben und als Parameter gespeichert. Dann wird der Übersetzung–Modus eingeschaltet — damit wird der Rest der Definition bis zum schließenden Semikolon schlicht kompiliert und dem angefangenen Parameterfeld hinzugefügt. In jedem Feld der Tabelle befindet sich dann ein `xt`, eine Einsprungadresse für das Wort, welches die für diese bestimmte Kombination aus Zeichen und Zustand vorgesehene Arbeit erledigt und ggf. einen Zustandswechsel herbeiführt.

```
: fsm: ( width -- )
  create
    ,             \ store width in flash
    ]             \ switch compiler on to
                  \ consume state table
  does> ( char col# -- )
                  \ -- char col# pfa
    swap over     \ -- char pfa col# pfa
    @i            \ -- char pfa C width
    mystate @ *   \ -- char pfa C W*state
    +             \ -- char pfa C+W*state
    1+            \ pfa[0] is width, move 1 up
    \ cells \ no cells, we are in flash
    +             \ -- char pfa[C+W*state]
    @i            \ -- char xt
    execute       \ -- ( xt consumes char! )
;
```

Einschränkung: Diese sehr simple Lösung, `]` im Kompilier–Zweig des neuen Wortes einzusetzen, funktioniert nur bei *indirect–threaded* Forths — also nicht in gforth.

Der Laufzeitanteil der Definition berechnet aus Spaltennummer, Breite der Tabelle und Zustand der FSM das zuständige Feld in der Tabelle, besorgt das dort abgelegte `xt` und führt es aus.

Die Definition der Zustandstabelle wird jetzt tatsächlich im Quelltext sichtbar. Man muss sich nur klarmachen, dass lediglich drei verschiedene Aktionen/Übergänge stattfinden: Annehmen des Zeichens und Übergang nach Zustand `1` oder Zustand `2` einerseits, sowie Verwerfen des Zeichens ohne einen Übergang andererseits.

```
: (>1) append  1 mystate ! ;
: (>2) append  2 mystate ! ;

4 wide fsm: <fixed.point#> ( action# -- )
\   0     1     2     3      \ column#
\   other 0-9   -     .      \ state
\ ----------------------------------
    drop  (>1)  (>1)  (>2)   \ 0
    drop  (>1)  drop  (>2)   \ 1
    drop  (>2)  drop  drop   \ 2
;
```

Das Wort `getafix` ändert sich nicht. Das Programm tut immer noch das Gleiche, aber `case` benötigen wir nicht mehr, und die Darstellung ist besser zu lesen. Das Wort `wide` tut nichts und dient ausschließlich der besseren Lesbarkeit.

Version 3

Die Worte `(>1)` und `(>2)` zeigen zwar den gewünschten Übergang an, aber nicht unbedingt die auszuführende Aktion. Das lässt sich verbessern, wenn man in jedem Feld der Tabelle zwei Einträge hinterlegt: (a.) das `xt` der gewünschten Aktion und (b.) die Nummer des gewünschten, nächsten Zustands, etwa so:

```
0 constant >0
1 constant >1
2 constant >2

4 wide fsm: <fixed.point#> ( char action# -- )
\ 0        1          2          3          \ state
\ other    0-9        -          .          \ state
  drop >0  append >1  append >1  append >2  \ 0
  drop >1  append >1  drop   >1  append >2  \ 1
  drop >2  append >2  drop   >2  drop   >2  \ 2
;
```

Die Definition der Konstanten ist nötig, weil der Kompiler die Worte einliest. Man könnte natürlich auch `drop [ 0 , ]` statt `drop >0` schreiben. Andererseits ermöglichen Konstanten die Bezeichnung der Zustände mit

sprechenden Namen. Die zugehörige Definition von `fsm:` sieht dann so aus:

```
: fsm: ( width -- )
  create
    ,             \ store width in flash
    ]             \ switch compiler on to
                  \ consume state table
  does> ( char col# -- )
                  \ -- char col# pfs
    swap over     \ -- char pfa col# pfa
    @i            \ -- char pfa C  width
    mystate @ *   \ -- char pfa C W*state
    +             \ -- char pfa C+W*state
    1+            \ pfa[0] is width, move 1 up
    \ cells \ no, we are in flash
    2* \ we have 2 xt per field now.
    +
    \ update state first
    dup @i execute mystate !
    \ call action ( xt consumes char! )
    1-  @i execute
;
```

Die Reihenfolge von Aktion und Änderung des Zustands ist willkürlich und kann selbstverständlich auch geändert werden. Der Vorteil dieser Version besteht darin, dass die Worte der Aktionen (`drop` und `append`) im Gegensatz zu den alten Worten `(>1)` und `(>2)` nichts mehr über das Ändern der Zustände wissen müssen.

Version 4

Zwei weitere Verbesserungen sollen noch vorgenommen werden. (a.) Geschachtelte oder gar rekursive FSMs sind nicht möglich, solange die Variable `mystate` global für alle FSMs einer Sorte gilt. Die Lösung ist aber naheliegend: man allokiert eine Zelle im RAM und speichert die zugehörige Adresse im Parameterfeld der FSM. (b.) Es soll zuerst die Aktion ausgeführt und dann der Zustand geändert werden. Dabei soll der Datenstapel frei sein, so dass die Aktion auch etwas auf dem Datenstapel hinterlassen könnte. Die Adresse des Zeigers auf den Status (`pfa[0]`) und die Adresse der zu Zustand und Eingabe passenden Tabellenzelle (`pfa[col#,state]`) werden auf dem Return–Stapel zwischengeparkt. Leider wird der Laufzeitteil des Wortes `fsm:` davon nicht besser lesbar.

```
: fsm: ( width -- )
  create
    here , 2 allot \ p->state
    ,              \ store width in flash
    ]              \ switch compiler on to
                   \ consume state table
  does> ( char col# -- ? )
    \ -- char col# pfa
    dup >r         \ -- char col# pfa
                   \ r: pfa
    1+ @i          \ -- char col# width
    r@ @i @ *      \ -- char C W*state
    +              \ -- char C+W*state
    2*   \ we have 2 xts per field now.
    r@ +           \ -- char p->action
    1+   \ pfa[0] is p->state, move 1 up
    1+   \ pfa[1] is width, move 1 up
    dup >r         \ -- char p->action
                   \ r: pfa p->action
    \ call action ( xt consumes char! )
    @i execute     \ -- ( ? )
    \ update state
    r> 1+          \ -- ( ? ) p->update
    @i execute     \ -- ( ? )
    r> @i !        \ -- ( ? )
;
```

Der Rest des Programms bleibt aber unverändert. Alle Versionen finden sich in den Listings und können von der VD–Webseite heruntergeladen werden.

Ausblick

Der Aufsatz von J. Noble geht weiter mit einer *nicht deterministischen* Variante einer FSM, die FORTRAN Identifier prüft, sowie einer anderen FSM, die Gleitkommazahlen mit Exponenten entgegennimmt. Vielleicht gibt es daher eine Fortsetzung des Artikels. Möglicherweise könnte so eine FSM als recognizer zusammen mit dem `float` Modul zum Verarbeiten von Gleitkommazahlen dienen. Selbstverständlich sind alle aufgerufen, Probleme, die sich mit einer FSM lösen lassen, anhand der vorgestellten Lösungen umzusetzen.

So Sachen

1. Bei der Aktion habe ich natürlich prompt eine Unzulänglichkeit (Fehler wäre schon zu hoch gegriffen) in amforth gefunden, der verhinderte, dass die frisch angelegte FSM `<fixed.point#>` in der Wortliste auftauchte. Matthias Trute hat das sehr zügig repariert. Die gezeigten Programme funktionieren ab der Version 4.9.

2. Und nochmal zur Wiederholung: die hier gezeigten Lösungen funktionieren nur mit *indirect–threaded* Forths.

Referenzen

1. `http://amforth.sourceforge.net/`
2. J. V. Noble — Finite State Machines in Forth, `http://www.forth.org/literature/noble.html`

Listings

Prozedural programmierte Lösung

```
\ 2012-04-06 fsm1.fs

include tempbuffer.fs \ reset append show

variable prev.minus?
variable prev.dpoint?

: digit?  [char] 0  [char] 9 within ;
: dpoint?  [char] . = ;
: minus?   [char] - = ;
: first.minus?
  minus?  prev.minus? @ not  and ;
: first.dpoint?
  dpoint?  prev.dpoint? @ not  and ;
: false true not ;

: legal? ( c -- f )
  dup digit? if
    drop true dup prev.minus? !
  else dup
    first.minus? if
      drop true dup prev.minus? !
    else
      first.dpoint? if
        true dup prev.dpoint? !
      else
        false
      then
    then
  then
;

: getafix
  reset
  false prev.minus? !
  false prev.dpoint? !
  begin
    key
    dup emit
    dup $0D <> \ exit on CR == &13 == $0D
  while
      dup legal? if append else drop then
  repeat
  drop \ CR
  cr show cr
;
```

Puffer für die Zeichenkette

```
\ 2012-04-06  tempbuffer.fs

$40 constant #mybuf
variable mybuf #mybuf allot
variable >mybuf

: reset
  mybuf >mybuf !
  mybuf #mybuf 0 fill
;

: append ( c -- )
  >mybuf @ swap over !
  1+
  mybuf #mybuf + \ last addr of mybuf
  min >mybuf !
;
: show
  mybuf >mybuf @ over - 0 ?do
    dup i + @ emit
  loop
  drop
;
```

Automat Version 1

```
\ 2012-04-06 fsm2.fs

include lib/ans94/postpone.frt
include case.fs
include tempbuffer.fs

variable mystate
: digit?   [char] 0  [char] 9  within ;
: dpoint?  [char] . = ;
: minus?   [char] - = ;

: (0) ( c -- )
  dup digit? over minus? or
  if         append   1 mystate !
  else
    dup dpoint?
    if       append   2 mystate !
    else
      drop
    then
  then
;
: (1) ( c -- )
  dup digit?
  if         append ( 1 mystate ! )
  else
    dup minus?
    if       drop   ( 1 mystate ! )
    else
      dup dpoint?
      if     append   2 mystate !
      else
        drop
      then
    then
  then
;
: (2) ( c -- )
  dup digit?
  if         append ( 2 mystate ! )
  else
             drop
  then
;

: <fixed.point#> ( char state -- )
  case
    0 of   (0) endof
    1 of   (1) endof
    2 of   (2) endof
  default:
  endcase
```

```
;

: getafix
  reset
  0 mystate !
  begin
    key
    dup emit
    dup $0D <> \ exit on CR
  while
      mystate @  <fixed.point#>
  repeat
  drop \ CR
  cr show cr
;
```

case

```
\ shamelessly stolen from gforth
\ added default: clause

0 constant case
  ( C: -- case-sys )
  ( R: -- )
  immediate

: of
  ( C:  -- of-sys )
  ( R: x1 x2 -- |x1 )
  1+ >r
  postpone over postpone =
  postpone if postpone drop
  r>
; immediate

: endof
  ( C: case-sys1 of-sys -- case-sys2 )
  ( R:  -- )
  >r postpone else r>
; immediate

: default:
  postpone drop
; immediate

: endcase
  ( C: case-sys -- )
  ( R: x -- )
  \ postpone drop \ moved to default:
  0 ?do postpone then loop
; immediate
```

Automat Version 2

```
\ 2012-04-06 fsm3.fs

include tempbuffer.fs

variable mystate
: digit?   [char] 0  [char] 9 within ;
: dpoint?  [char] . = ;
: minus?   [char] - = ;

: wide     ;

: cat->col# ( n -- n' )
  \     other?   0
  dup   digit?   1 and
  over  minus?   2 and +
  swap  dpoint?  3 and +
;

: fsm: ( width -- )
  create
    ,            \ store width in flash
    ]            \ switch compiler on to
                 \ consume state table
  does> ( char col# -- )
                 \ -- char col# pfa
    swap over    \ -- char pfa col# pfa
    @i           \ -- char pfa C  width
    mystate @ *  \ -- char pfa C W*state
    +            \ -- char pfa C+W*state
    1+           \ pfa[0] is width, move 1 up
    \ cells \ no, we are in flash
    +            \ -- char pfa[C+W*state]
    @i           \ -- char xt
    execute      \ -- ( xt consumes char! )
;

: (>1) append  1 mystate ! ;
: (>2) append  2 mystate ! ;

4 wide fsm: <fixed.point#> ( action# -- )
\   0     1     2     3      \ column#
\   other 0-9   -     .      \ state
    drop  (>1)  (>1)  (>2)   \ 0
    drop  (>1)  drop  (>2)   \ 1
    drop  (>2)  drop  drop   \ 2
;

: getafix
  reset
  0 mystate !
  begin
    key
    dup emit
    dup $0D <> \ exit on CR
  while
    dup cat->col# <fixed.point#>
  repeat
  drop \ CR
  cr show cr
;
```

Automat Version 3

```
\ 2012-04-06  fsm4.fs

include tempbuffer.fs

variable mystate
: digit?   [char] 0  [char] 9  within ;
: dpoint?  [char] . = ;
: minus?   [char] - = ;
: wide     ;

: cat->col# ( n -- n' )
  dup   digit?   1 and
  over  minus?   2 and +
  swap  dpoint?  3 and +
;
```

```
: fsm: ( width -- )
  create
    ,            \ store width in flash
    ]            \ switch compiler on to
                 \ consume state table
  does> ( char col# -- )
                 \ -- char col# pfs
    swap over    \ -- char pfa col# pfa
    @i           \ -- char pfa C  width
    mystate @ *  \ -- char pfa C W*state
    +            \ -- char pfa C+W*state
    1+           \ pfa[0] is width, move 1 up
    \ cells \ no, we are in flash
    2* \ we have 2 xt per field now.
    +
    \ update state first
    dup @i execute mystate !
    \ call action ( xt consumes char! )
    1-  @i execute
;
0 constant >0
1 constant >1
2 constant >2

4 wide fsm: <fixed.point#> ( char action# -- )
\ 0      1          2          3              \ state
\ other  0-9        -          .
drop >0  append >1  append >1  append >2 \ 0
drop >1  append >1  drop   >1  append >2 \ 1
drop >2  append >2  drop   >2  drop   >2 \ 2
;
: getafix
  reset
  0 mystate !
  begin
    key
    dup emit
    dup $0D <> \ exit on CR
  while
      dup cat->col# <fixed.point#>
  repeat
  drop \ CR
  cr show cr
;
```

Automat Version 4

```
\ 2012-04-06  fsm5.fs

include tempbuffer.fs

: digit?   [char] 0  [char] 9  within ;
: dpoint?  [char] . = ;
: minus?   [char] - = ;
: wide     ;

: cat->col# ( n -- n' )
  dup   digit?   1 and
  over  minus?   2 and +
  swap  dpoint?  3 and +
;
: fsm: ( width -- )
  create
    here , 2 allot \ p->state
    ,            \ store width in flash
    ]            \ switch compiler on to
                 \ consume state table
  does> ( char col# -- ? )
    \ -- char col# pfa
    dup >r       \ -- char col# pfa
                 \ r: pfa
    1+ @i        \ -- char col# width
    r@ @i @ *    \ -- char C W*state
    +            \ -- char C+W*state
    2*   \ we have 2 xts per field now.
    r@ +         \ -- char p->action
    1+   \ pfa[0] is p->state, move 1 up
    1+   \ pfa[1] is width, move 1 up
    dup >r       \ -- char p->action
                 \ r: pfa p->action
    \ call action ( xt consumes char! )
    @i execute   \ -- ( ? )
    \ update state
    r> 1+        \ -- ( ? ) p->update
    @i execute   \ -- ( ? )
    r> @i !      \ -- ( ? )
;

0 constant >0
1 constant >1
2 constant >2

4 wide fsm: <fixed.point#> ( char action# -- )
\ 0      1          2          3
\ other  0-9        -          .              \ state
drop >0  append >1  append >1  append >2 \ 0
drop >1  append >1  drop   >1  append >2 \ 1
drop >2  append >2  drop   >2  drop   >2 \ 2
;

: getafix
  reset
  0  ['] <fixed.point#> 1+ @i  ! \ 0 mystate !
  begin
    key
    dup emit
    dup $0D <> \ exit on CR
  while
      dup cat->col# <fixed.point#>
  repeat
  drop \ CR
  cr show cr
;
```

Forth–Gesellschaft e.V.
Ordentliche Mitgliederversammlung
11.03.2012

Erich Wälde

Moderation Carsten Strotmann

Protokollant Erich Wälde

Teilnehmer
Direktorium: Ewald Rieger Ulli Hoffmann Bernd Paysan
insgesamt 18 stimmberechtigte Mitglieder (Anzahl der ausgegebenen Stimmkarten)

Sitzungsdatum 11.03.2012

Sitzungsbeginn 09:30 h

Sitzungsende 11:50 h

Sitzungsort Kloster Beukenhof, Biezenmortel, Noord–Brabant, NL

1. Begrüßung

Ewald Rieger begrüßt im Namen des Direktoriums die anwesenden Mitglieder.

2. Wahl des Schriftführers

Das Protokoll übernimmt Erich Wälde.

3. Wahl des Versammlungsleiters

Zum Sitzungsleiter wird Carsten Strotmann gewählt. Der Sitzungsleiter stellt fest, dass die Versammlung fristgerecht einberufen wurde. Es wurden 18 von 114 Stimmkarten ausgegeben. Damit sind mehr als 10 % der Mitglieder anwesend und die Versammlung ist beschlussfähig.

4. Ergänzungen zur Tagesordnung

Mehrere Punkte für Verschiedenes

5. Bericht des Direktoriums

a. Bericht der Verwaltung (Ewald Rieger)

Mitgliederentwicklung Im Jahr 2011 gab es 3 Neuzugänge und 6 Austritte. Der Verein hatte zum Jahresende 2011 114 Mitglieder. In 2012 sind bereits 3 Austritte zu verzeichnen. Auffällig ist, dass ein Teil der Austretenden bereits nach wenigen Jahren den Verein wieder verlässt.

Satzungsänderung Die zur Eintragung anstehende Satzungsänderung betreffend die Auflösung des Vereins wurde am 18.3.2011 (einen Tag nach der letzten Jahresversammlung) vom zuständigen Registergericht vorgenommen.

Finanzen Ewald Rieger erläuterte im Detail die Einnahmen und Ausgaben, getrennt nach Verein und Zweckbetrieb (Vierte Dimension). Es ergibt sich ein Vermögen des Vereins von 12938.42 € zum Jahresende. Dieser Betrag ist um ca. 2800 € niedriger als im Vorjahr. Das ist der vom zuständigen Finanzamt gewünschte und planmäßige Abbau des Vermögens.

Für das Jahr 2012 sind Einnahmen von etwa 4400 € zu erwarten und Ausgaben in Höhe von 7950 €geplant (das beinhaltet eine zweckgebundene Spende von 1200 € für die Durchführung der Euroforth). Damit ist zum Jahresende 2012 ein Vermögen von etwa 9400 € zu erwarten.

Man kann noch ein weiteres Jahr recht bequem mit den derzeitigen Ausgaben wirtschaften. Dann allerdings muss wieder eine Kostendeckung erreicht werden. Eine Möglichkeit wäre die Anhebung der für Rentner reduzierten Beiträge, die lediglich das Heft abdecken, nicht jedoch den Betrieb des Webservers und andere Kosten.

Bericht des Kassenprüfers

Friederich Prinz hat am 09.03.2012 die Kassenprüfung durchgeführt. Er berichtet der Versammlung, dass alle Geldbewegungen plausibel, nachprüfbar, belegt sowie sachlich und rechnerisch korrekt sind. Er lobt die sorgfältige Buchhaltung ausdrücklich und empfiehlt uneingeschränkt die Entlastung des Direktoriums.

b. Rund um das Forth Magazin (Ulli Hoffmann)

Vor einem Jahr wurde beschlossen, die Aufgabe des Chefredakteurs auf mehrere Personen zu verteilen. Das hat sehr gut funktioniert: Ulli Hoffmann, Carsten Strotmann, Bernd Paysan und Erich Wälde haben je ein Heft zusammengestellt. Die Zusammenarbeit über die zentrale Versionsverwaltung subversion und per email hat einwandfrei funktioniert. Die Hefte kommen allerdings nur durch die Mitwirkung der Autoren und Helfer — allen voran Fred Behringer als Korrekturleser — zustande. Ulli Hoffmann dankte an dieser Stelle ausdrücklich allen Mitwirkenden und den Autoren.

Auf die Frage, ob das Heft eine ISBN/ISSN habe, lautet die Antwort: nein, haben wir nicht. Allerdings wird

ein Exemplar an das deutsche Zentralarchiv in Hannover geschickt.

Es gibt Überlegungen/Anfragen, ob man das Magazin nicht auch im epub–Format veröffentlichen und dann u.U. über den Apple Shop oder Amazon zu einem geringen Entgelt vertreiben könnte. Das wäre möglicherweise auch eine neue Art der Werbung für den Verein.

Das Heft 2012-01 ist annähernd fertig.

Ein neues AVR– oder Mikrokontroller–Sonderheft, welches in Fortbildungen verteilt werden kann, steht ebenfalls auf der Aufgabenliste.

Für die Redaktionsarbeit werden weiterhin Freiwillige gesucht. Es ist nicht wichtig, dass jemand das Heft selbst mit Text füllt, sondern die Artikel zusammenfügt, den Autoren gut zuredet und die Korrekturen vornimmt.

c. Internet–Präsenz

Der Umzug vom kränkelnden root–server auf einen günstigeren V–server (virtuelle Instanz statt eigener hardware) hat sich als sehr vorteilhaft herausgestellt. Alles läuft ruhig und unauffällig.

d. Außendarstellung und Projekte

Die größten Aktivitäten in diesem Bereich kommen derzeit von den Fort*h*bildern, die netterweise auch über ihre Aktivitäten berichten.

Unklar ist, ob wir eine Beteiligung am Linuxtag in Berlin (23.-26.05.2012) auf die Beine bekommen. Am nächsten Wochenende finden die Chemnitzer Linuxtage statt, an denen Carsten Strotmann und Erich Wälde wieder eine Fort*h*bildung veranstalten.

6. Entlastung des Direktoriums

Anton Ertl stellt den Antrag, das Direktorium zu entlasten.

Der Antrag wird einstimmig angenommen, das Direktorium ist damit entlastet.

7. Wahl des Direktoriums

Die bislang amtierenden Mitglieder des Direktoriums sind bereit, für eine weitere Amtszeit zu kandidieren.

Das Direktorium (Ewald Rieger, Ulli Hoffmann, Bernd Paysan) wird einstimmig für eine weitere Amtszeit gewählt. Alle Gewählten nehmen die Wahl an.

8. Projekte

Dieser Punkt fand keine separate Beachtung.

An dieser Stelle wurde die Sitzung unterbrochen. Der Tradition folgend hatte der Drachenrat am Vorabend getagt. Friedel Amend übergab den Drachen an den neuen Drachenhüter Erich Wälde. Die erste Amtshandlung des neuen Drachenhüters war die ebenso traditionelle Aufstellung zum Gruppenfoto.

9. Verschiedenes

Mailing Liste

Erich Wälde schlägt die Einrichtung einer geschlossenen Mailing Liste nur für Vereinsmitglieder vor. Die Hoffnung ist, dass dadurch die Vereinsmitglieder öfter vom Verein hören (und nicht nur über das Heft) und sich evtl. auch zum Mitmachen hinreißen lassen. Der Vorschlag stößt auf Zustimmung.

Linuxtag 2012 Berlin

Die Teilnahme am Linuxtag 2011 hatten wir aufgrund von Helfermangel abgesagt. Die Teilnahme dieses Jahr ist auch noch nicht klar.

Zum einen fehlt es noch an Freiwilligen für die Standbesatzung. Es braucht vier Personen an allen Tagen.

Zum anderen ist für einen Hingucker noch Arbeit zu leisten. In selbstloser Weise hat Friedel Amend eine neue, viel schickere und schnellere Version des Trizeps gebaut. Bernd Paysan übernimmt es, den Roboter mit dem Microcore board anzusteuern. Dennoch wäre eine Ansteuerung via Arduino oder MSP430 ebenfalls denkbar und wünschenswert.

Der Roboter hat wechselbare Spielbretter (Mühle, Go, Solitär) und eine Parkposition, die es erlaubt, dass eine Person gegen den Roboter spielt. Der Roboter *sieht* das Spielfeld über eine Kinect (playstation?).

Eine Live-CD (Knoppix) mit installierten Entwicklungsumgebungen und Forths wird ebenfalls angeregt.

Martin Bitter hat einen kleinen Stirlingmotor, der über einer Thermoskanne mit heißem Wasser läuft. Mit Temperatur– und Drehzahlsensoren könnte eine Art Prüfstand realisiert werden.

Ein Forth–Stammtisch im Gasthaus zur Dicken Wirtin wird ebenfalls wieder organisiert.

Mikrokontrollerverleih

Im Mikrokontrollerverleih gibt es neue Boards:

- ein **GA144** eval board von Green Arrays
- ein **seaforth** eval board (Spende von Stephen Pelc)
- sowie drei **MSP430-FRAM** Wolverine boards

An die Ausleiher ergeht die Bitte, aufzuschreiben, was sie herausgefunden haben, v.a. Probleme und deren Lösung oder Einschränkungen der Systeme. Extra Bonus für Aufzeichnungen, aus denen sich ein VD–Artikel zimmern lässt.

Es ergeht die Anregung, auch Arduinos in den Verleih aufzunehmen.

Es gibt die Anfrage, ob es auch ein Forth auf Android gibt. Bernd Paysan: gforth an sich tät schon. openGLES tut auch (für GUIs). Aber als einfach installierbare Äpp ist es noch nicht eingepackt.

Vierte Dimension 2012/13

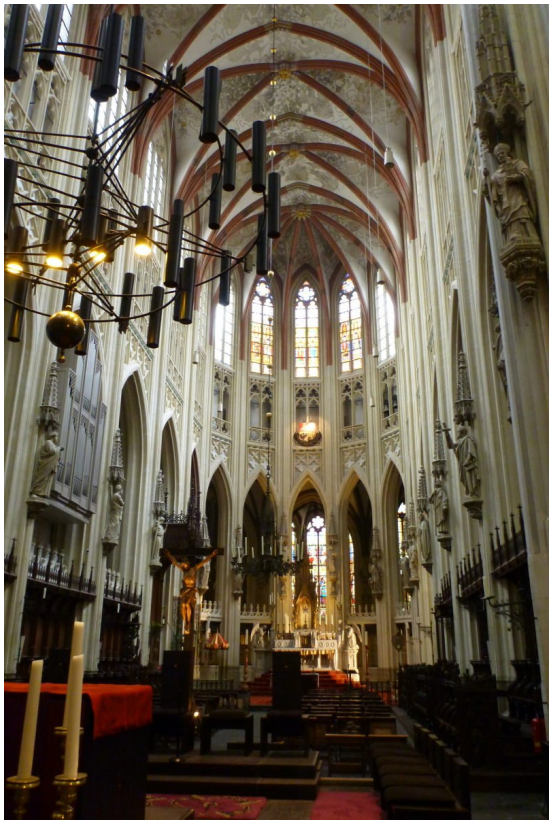
Redaktion aktuelles Heft: Ulli Hoffmann, dann Bernd Paysan, Martin Bitter & Carsten Strotmann zusammen, sowie Erich Wälde. Freiwillige sind willkommen, unter Anleitung mitzuhelfen oder auch ein Heft zu übernehmen.

Vintage Computer Festival Europe

Auf dem Vintage Computer Festival Europe in München wird der Forth Benchmark Wettbewerb fortgesetzt. Das Komitee dort bittet darum, wieder Preise zu stiften. Wird vom Direktorium erledigt.

Tagung 2013

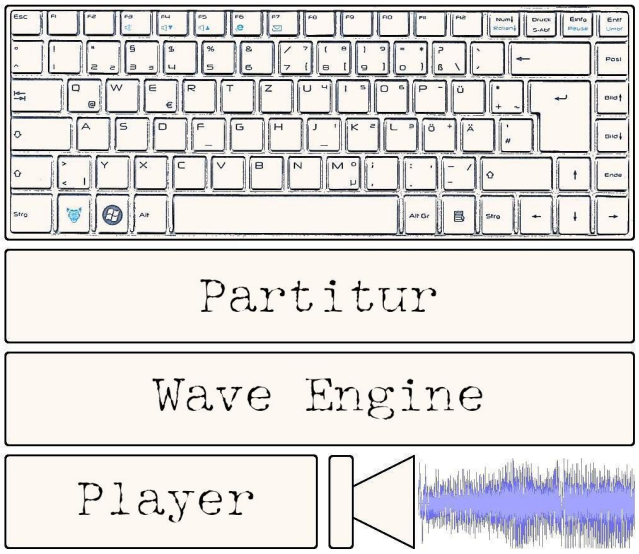
Für den Tagungsort 2013 konnte in der Versammlung kein Organisator gefunden werden. Gerüchteweise wäre Heinz Schnitter bereit, etwas zu organisieren. Eine Tagung raum– und zeitgleich mit einer anderen Veranstaltung ist noch nicht spruchreif (Süddeutschland). Als Notlösung käme das Kloster Roggenburg in Frage. Außerdem wird darum gebeten, die Jahrestagung wieder eher in den April zu verlegen, weil der März gewöhnlich mit etlichen (Messe-)terminen gefüllt ist.



Wave Engine (3)

Hannes Teich

Mit der letzten Folge ist das Gesamtkonzept der „PC–Orgel“ so weit skizziert worden, dass wir uns nun einigen überschaubaren Teilaspekten zuwenden können.



Nochmal weicher Toneinsatz

Ich komme auf die Routine **adjust** zurück, die für die verschiedenen Dämpfungen der Töne zuständig ist. Hier nochmal der Code, zumal er in der letzten Folge eine schadhafte Zeile enthielt. Diese nun korrigierte, hier aber ausgeklammerte Zeile war nur für die Funktion *mellow* gedacht, nicht aber für *fading*, *sustain* und *staccato*.

```
: adjust  ( n - r)
          0 max 3000 min
          ( ?dup 0= IF 3000 THEN )
          draft_ IF 4 * THEN
          1e s>f 1e5 f/ 1e f+ f/ ;
```

Besagte Zeile sollte mit Null die *mellow*–Funktion abschalten. Für *fading*, *sustain* und *staccato* gilt: Je höher Parameter n, desto stärker die Bedämpfung. Für *mellow* bedeutet „starke Bedämpfung“ jedoch das Gegenteil, nämlich einen schnellen Tonanstieg, während eine Null den Ton niemals hoch kommen ließe.

Irgendwie hat mir diese Ausnahme nicht gefallen, darum habe ich die Sache umgedreht: Null schaltet die Funktion *mellow* ab und bewirkt damit einen harten Toneinsatz, hohe Werte machen das Einschwingen weich. Und wenn ich’s recht bedenke, ist auch *sustain* betroffen: Eine Null soll keinen endlosen Nachklang bedeuten, sondern gar keinen. Im Beispiel weiter unten ist das mit sus=450 schon berücksichtigt.

Eine Division bringt die gewünschte Charakteristik. Hier sind einige Beispiele, deren Wirkung auf *mellow* in Abb. 1 zu sehen ist:

```
: corr ( n1 - n2) 10 + 30000 swap / ;
     0  corr adjust  f. 0.97087  ok
    10  corr adjust  f. 0.98522  ok
   100  corr adjust  f. 0.99729  ok
   300  corr adjust  f. 0.99904  ok
  1000  corr adjust  f. 0.99971  ok
  3000  corr adjust  f. 0.99991  ok
```

Zur Erzeugung von Abb. 1 waren noch weitere Parameter im Spiel. Die *Partitur* sah aus wie folgt (wobei für mel=0 nacheinander mel=10 bis mel=3000 eingesetzt wurde):

```
 fix{ste=0;dra=0;}
 set{tem=195;cha=A;pit=440;mic=0;}
regA{par=(2000);fad=4;sus=450;
                 sta=200,2000;mel=0;}
   A{:1/4 _ 4G 4F 4E 4D 4C ~ _ }.
```

Die Kurven wurden mit der Anwendung Audacity gemalt.

Wir haben hier viererlei *Partiturzeilen*: **fix{...}** ist unveränderlich, **set{...}** könnte auch mehrmals auftauchen, **regA{...}** ist die *Registrierung* von *Manual* A, und in **A{...}** stecken die Musiknoten für *Manual* A. Der Punkt am Ende bedeutet Stopp.

Die Funktionen **stereo** und **draft** sind mit Null abgeschaltet; **tempo** meint Viertelnoten pro Minute; **channels=A** blendet die *Manuale* B, C und D aus; **pitch** ist der Kammerton mit 440 Hertz; **micro=0** bedeutet die übliche temperierte Stimmung (als Alternative zur reinen Stimmung mit kleineren Intervallen); ***partials*=(2000)** erzeugt Sinustöne (hier ohne Obertöne) mit vorgegebener Anfangslautstärke; ***fading*** bestimmt das Abklingen, ***sustain*** das Nachklingen, ***staccato*** die Lücken zwischen den Tönen, und ***mellow*** schließlich die Weichheit des Einschwingens.

Beim Abspielen der Wave–Datei erklingen fünf Töne: g’ – f’ – e’ – d’ – c’ mit der Dauer von vier Viertelnoten und einer Halbnote. Die Pause am Anfang ist eine Rücksichtnahme auf Abspielgeräte und –programme, die sich sonst verhaspeln könnten. Die Pause am Ende schafft für den Nachklang (*sustain*) Platz.

Die Amplitude und damit die subjektive Lautheit sind übrigens stark von *mellow* abhängig. Die Anwendung Audacity gleicht das auf Knopfdruck aus. In der *Partitur* kann – mit der nötigen Geduld – ein Ausgleich über die Funktion *intensity* vorgenommen werden.

Zur Schreibweise der Registerzeilen

Folgendes Beispiel **regA{...}** einer Registerzeile ist im Prinzip schon bekannt, aber es gibt noch einiges zu sagen. Die Regeln sind nicht ganz trivial; sie sollen halt das Abfassen der *Partitur* so weit wie möglich erleichtern. Der Reiz für den Programmierer liegt darin, diese Regeln zu bedienen.

```
regA{ *1 partials=
      (1:3000  2:1000  3: 500  4:   0
       5:   0  6:   0  7:   0  8:   0
       9:   0 10:   0 11:   0 12:   0
      13:   0 14:   0 15:   0 16:   0),
      (1:   0  2:   0  3:   0  4:   0
       5:   0  6:   0  7:   0  8:   0
       9:   0 10:   0 11:   0 12: 800
      13:   0 14:   0 15:   0 16:   0),
      (1:   0  2:   0  3:   0  4:   0
       5:   0  6:   0  7:   0  8:   0
       9:   0  10:  0 11:   0 12:   0
      13:   0 14:   0 15:   0 16:   0);
      fading=20,150,0;
      sustain=40,0,0;
      mellow=10,0,0;
      staccato=900,100;
      intensity=0; }
```

Und dasselbe in Kuzschreibweise:

```
regA{*1 pre=0;par=(3000 1000 500),(12:800);
        fad=20,150;sus=40;mel=10;sta=900,100;}
```

Es handelt sich offensichtlich um die *Registrierung* des *Manuals* A. (Zur Erläuterung des *habitus* ***1** komme ich noch.) Oben sieht man die dreimal 16 *Teiltöne* (*partials*) einzeln aufgeführt. In der kurzen Version sind alle Nullen weggelassen bzw. durch ***preset*=0** ersetzt (das nicht am Anfang stehen müsste, denn es wird als Erstes erkannt). Die drei *Teiltonreihen* werden jeweils in runde Klammern gesetzt, aber sie müssen nicht alle aufgeführt werden, wenn sie inhaltlich dem *preset* entsprechen. Ein Semikolon nach der ersten oder zweiten schließenden runden Klammer ist zulässig und beendet die *partials*. Die

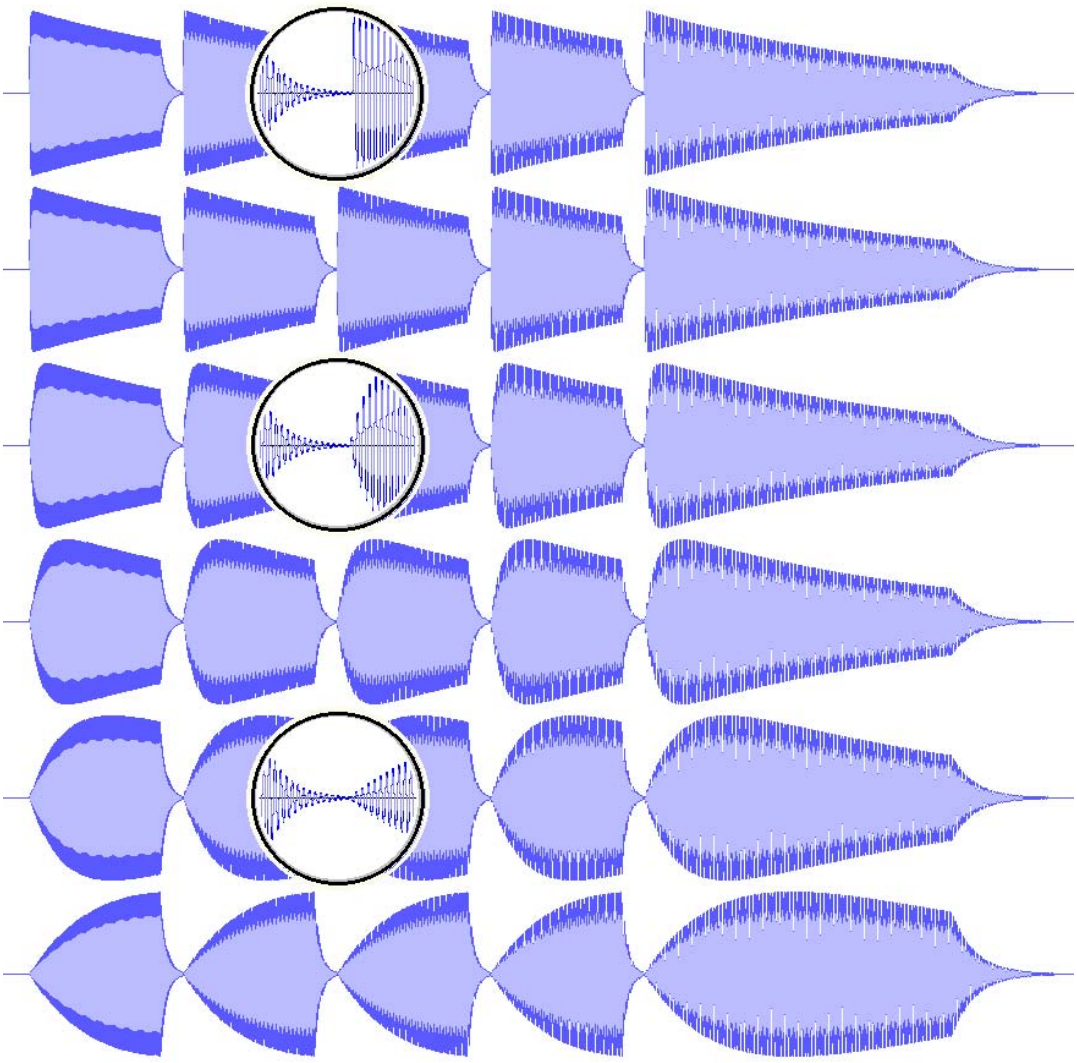


Abbildung 1: Die *mellow*–Funktion mit den Vorgabewerten 0, 10, 100, 300, 1000 und 3000. Man sieht den weichen Einsatz, das Abklingen der Töne, das Nachklingen am Ende sowie die Lücken zwischen den Tönen. Die drei Lupen spreizen nur horizontal.

Teiltonnummern können jeweils dort entfallen, wo die Positionen ohnehin eindeutig sind. Die Nummer 12 in der zweiten Klammer wird benötigt. Die Werte 3000, 1000, 500 etc. geben die Lautstärke der *Teiltöne* vor und gelten für alles, was auf *Manual* A gespielt wird.

Das Abklingen (*fading*) geschieht mit 20 recht moderat, mit 150 für die zweite *Teiltonreihe* (die für die Einschwingvorgänge vorgesehen ist) deutlich rascher. Der Nachklang (*sustain*) erübrigt sich für die zweite *Teiltonreihe*. Die dritte wird hier gar nicht verwendet. Der weiche Einsatz (*mellow*) betrifft nur die erste Reihe. Für die Lücken zwischen den Tönen (*staccato*) ist die Dauer mit 900 *[Wave–Datei–Frames]* angegeben, was etwa 1/50 Sekunde entspricht. Der Wert für die *staccato*–Dämpfung wird der Routine **adjust** übergeben wie oben erläutert. Mit *intensity* kann die Lautstärke des aktuellen *Manuals* nach oben oder unten justiert werden. Da sie hier neutral auf Null steht, kann sie in der Kurzform entfallen.

Und so liest Forth die Registerzeilen

Chuck Moore: *»Factoring into subroutines or threads is essential. A wall of straight–line code is ugly. As is code loaded with comments or conditions.«* – Das leuchtet mir zwar ein, aber im Moment bin ich glücklich, dass die folgende Routine das tut, was sie soll, nämlich die Registierung der *partials* interpretieren, wie oben beschrieben. Auch Negativ–Beispiele entbehren nicht eines gewissen didaktischen Wertes.

```
: par:  ( a u -- a' u' true)
            0 prow !
  BEGIN     read-multiline
        '(' extract-char?   not IF |120| THEN
            0 param !
    BEGIN   read-multiline
        ')' extract-char? not dup
      IF    param @ &17 <   not IF |121| THEN
      THEN
    WHILE   extract-unum    not IF |122| THEN
            >r over c@ ':' =
      IF    proceed
            r@ 1 17 within  not IF |123| THEN
            r> 1- param !
            extract-unum    not IF |122| THEN
            >r
      THEN  r@
            0 &5001 within  not IF |124| THEN
            r> reg-X prow @ &16 *
            param @ + cells + !  \ load reg-X
            param incr
    REPEAT  prow incr
            over c@ ';' = not
      IF    prow @ 3 <      not IF |125| THEN
            over c@ ',' =   not IF |126| THEN
            false
      ELSE  true
      THEN  >r proceed r>  UNTIL      true ;
```

Kommentare und Stackzustände habe ich aus Platzgründen weggelassen. Hinter den Nummern wie |120| verbergen sich detaillierte Fehlermeldungen, denn eine *Partitur* syntaktisch fehlerfrei zu schreiben ist eine Kunst. **read–multiline** erlaubt die Spreizung einer *Partiturzeile* über mehrere Textzeilen: Immer dann, wenn der String leer ist, wird eine neue Textzeile gelesen. Der Kommentar weist auf die Stelle, wo in das *X–Register* (**reg–X**) geschrieben wird. Das ist ein Vorab–Register, aus dem die echten Register (**reg–A**, **reg–B** etc.) gespeist werden. Das **true** am Ende sagt, dass es sich noch nicht um das Zeilenende handelt.

Die übrigen Elemente der Registerzeile sind deutlich einfacher. Hier ist die entsprechende Routine für den *fading*–Effekt:

```
: fad:  ( a u -- a' u' true)
      extract-unum
            IF reg-X fad1 + ! ELSE |220| THEN
      over c@ ';' = IF proceed true EXIT  THEN
      over c@ ',' = IF proceed ELSE |221| THEN
      extract-unum
            IF reg-X fad2 + ! ELSE |222| THEN
      over c@ ';' = IF proceed true EXIT  THEN
      over c@ ',' = IF proceed ELSE |223| THEN
      extract-unum
            IF reg-X fad3 + ! ELSE |224| THEN
      over c@ ';' = IF proceed ELSE |225| THEN
      true ;
```

Hier werden bis zu drei Parameter gelesen, aber ein Semikolon kann bereits nach dem ersten oder zweiten Parameter die Sache abschließen. **extract–unum** liest eine vorzeichenlose Zahl (andernfalls Fehlermeldung und Abbruch), **proceed** rückt im String eine Position weiter. Das **true** am Ende hatten wir schon.

Die Routinen für *sustain* und *mellow* sehen genau so aus, und die übrigen Routinen sind recht ähnlich. Stets handelt es sich darum, die gelesenen Werte im *X–Register* an der richtigen Stelle abzulegen. – Hier sind noch zwei weitere (**pre:** und **hab:**), auf die ich weiter unten zurückkomme:

```
: |364| err ." preset= unsigned number missing."
  quit ;
: |365| err ." preset= parameter lacks a close
semicolon." quit ;
: |366| err ." preset= invalid preset number
 (valid: 0...2)." quit ;
: pre:  ( a u -- a' u' true)          \ preset
      extract-unum
            IF reg-X pre1 + ! ELSE |364| THEN
      over c@ ';' = IF proceed ELSE |365| THEN
      true to preflg_
      reg-X pre1 + @
      lowest pre-X recwid xfill
  CASE              \ preset=0|1|2
    0 OF    \ clear pre-X completely
          0 pre-X recwid          xfill  ENDOF
    1 OF    \ clear partials only
          0 pre-X 48              xfill  ENDOF
```

```
    2 OF   \ clear all but partials
         0 pre-X 48 + recwid 48 - xfill  ENDOF
    ( place more presets here)
                          ( error)  |366|
  ENDCASE
      true ;
: |367| err ." habitus must be 1st item in
register line." quit ;
: |368| err ." identifier (*) must be immediat
ely followed by an unsigned number." quit ;
: hab:  ( a u -- a' u' f)            \ habitus
      item# @ IF |367| THEN
      extract-unum not IF |368| THEN
      to habit#_ true ;
```

Alle diese Routinen werden von Routine **regline** aus aufgerufen.

Eine Registerzeile enthält meist nur eine Untermenge aller Funktionen. Mit ***preset*=0** kann (via **pre–X**) das *X–Register* vorher nullgesetzt werden, aber oft möchte man die Einstellungen, die nicht neu gesetzt werden, erhalten. Deshalb ist ein *Transparent–Zeichen* erforderlich, das dafür sorgt, dass bei der Übertragung von **reg–X** nach **reg–A** keine ungewollten Löschungen geschehen. Für dieses Zeichen wurde **lowest (= $80000000)** gewählt. Damit wird das *X–Register* (**reg–X**) durch **xfill** zellenweise vorbesetzt. Die Routine **xmove** bewerkstelligt die Übertragung, ohne die **lowest**–Werte mit zu übertragen. Konstante **recwid** liefert die Anzahl der Zellen jedes Registers oder *Record*s (derzeit 112).

Zu *habitus* und *Record*s sage ich gleich noch etwas, aber hier ist erst einmal **regline**. (**regline** wird von Routine **interpreter** gestartet, die in Heft 3/2011 zu sehen war.)

```
: regline  ( a u -- a' u')
           lowest reg-X recwid xfill
           0 item# !
    BEGIN  exec-reg ( a' u' f) not
                   \ handle all items
    UNTIL          \ end of line detected
           Manuals @ %1000 and  \ Manual A
    IF     preflg_              \ preset flag?
      IF   pre-X reg-A recwid xmove
      THEN reg-X reg-A recwid xmove
           settle-A        \ load FP variables
    THEN
           Manuals @ %0100 and  \ Manual-B
    IF     preflg_              \ preset flag?
      IF   pre-X reg-B recwid xmove
      THEN reg-X reg-B recwid xmove
           settle-B        \ load FP variables
    THEN
           Manuals @ %0010 and  \ Manual-C
    IF     preflg_              \ preset flag?
      IF   pre-X reg-C recwid xmove
      THEN reg-X reg-C recwid xmove
           settle-C        \ load FP variables
    THEN
           Manuals @ %0001 and  \ Manual-D
    IF     preflg_              \ preset flag?
      IF   pre-X reg-D recwid xmove
      THEN reg-X reg-D recwid xmove
           settle-D        \ load FP variables
    THEN
           0 to preflg_
  habit#_ IF  habit#_ $1F and :pool { rec-N }
              lowest rec-N recwid xfill
              pre-X  rec-N recwid xmove
              reg-X  rec-N recwid xmove
           THEN ;
```

Am Anfang von **regline** wird das *X–Register* transparent gemacht; Variable **item#** zählt die Elemente der Registerzeile, denn die *habitus–Marke* muss, wenn überhaupt, am Anfang stehen; **exec–reg** durchsucht die Tabelle **regsym** mit allen zulässigen Element–Namen einer *Registerzeile*. (Die Tabelle wurde in der letzten Folge gezeigt.)

```
: |380| err ." unknown element name." quit ;
: exec-reg  ( a u -- a' u' f)
            read-multiline ( a u)
            regsym extract-token not
      IF    |380|
      THEN  PERFORM
            item# incr ;
```

Die Routine **err** schreit ***ERROR***, und schreibt die aktuelle Zeile samt Zeilennummer bis hin zum Missgeschick.

Die Routine **settle–A** hat die Aufgabe, die vom *Generator* benötigten Floatpoint–Werte zu generieren, denn die Register enthalten nur Zahlen mit einfacher Präzision (*cell–sized*). Immer dann, wenn **reg–A** verändert wird, ist **settle–A** fällig.

```
: aide  ( n - r)  2e s>f 500e f/ f** ;
: settle-A  ( --)  cr ." settle-A "
             \ fading 3 p-rows
       reg-A fad1 + @ adjust 0 :fade-A* f!
       reg-A fad2 + @ adjust 1 :fade-A* f!
       reg-A fad3 + @ adjust 2 :fade-A* f!
             \ sustain 3 p-rows
       reg-A sus1 + @ corr adjust 0 :sust-A* f!
       reg-A sus2 + @ corr adjust 1 :sust-A* f!
       reg-A sus3 + @ corr adjust 2 :sust-A* f!
             \ mellow 3 p-rows
       reg-A mel1 + @ corr adjust 0 :mell-A* f!
       reg-A mel2 + @ corr adjust 1 :mell-A* f!
       reg-A mel3 + @ corr adjust 2 :mell-A* f!
             \ staccato
       reg-A sta1 + @ adjust    stac-A* f!
       reg-A sta1 + cell+ @ to gaplim-A_
             \ intensity & volume
       reg-A int1 + @ aide intens-A* f!
       reg-A vol1 + @ aide volume* f!
             ( more to come) ;
```

Zurück zu **regline**: Da die beteiligten *Manuale* 16 Kombinationen zulassen (**A**, **AB**, **ACD** etc.), sind sie in der Variablen ***Manuals*** in dualer Form repräsentiert ($1000, $1100, $1011 etc.). Wenn ***preset*** gesetzt ist, ergießt sich

das *Preset–Register* **pre–X** nach **reg–A**. Danach wird **reg–X** nach **reg–A** kopiert. Beide Male sorgt **xmove** dafür, dass *Transparent–Zeichen* **lowest** nicht mit übertragen werden.

Die Sache mit dem *habitus* und den *Record*s

Am Ende der Routine **regline** wird in ein *Record* geschrieben, als Teil von **:pool**, in dem 32 *Record*s zu je 112 Zellen Platz finden. Die geschwungene Klammer installiert übrigens eine „lokale Variable“ **rec–N** vom Typ *value*, nach Gforth–Art, und **rec–N** versieht das durch **habit#_** gewählte *Record* mit einem flüchtigen Namen. Dann geschehen drei Aktionen: *Record* **rec–N** wird transparent gemacht, dann mit *Preset–Daten* versehen und schließlich aus dem *X–Register* (**reg–X**) mit aktuellen Daten gespeist.

So ist der *Records–Pool* (:pool) definiert:

```
create pool| reclen 32 * allot   \ 32 Records
: :pool  ( # -- addr)  reclen * pool| + ;
```

By the way, diese Form einer *Registerzeile* ist nicht typisch:

```
regA{*1 pre=0;par=(3000 1000 500),(12:800);
fad=20,150;sus=40;mel=10;sta=900,100;}
```

Und zwar deshalb: Entweder ich registriere eines oder mehrere *Manuale* direkt, z. B. mit **regABCD{...}**, oder ich verwende einen *Habitus* (z. B. *1), um die *Registrierung* in einem *Record* zu speichern, von wo sie in einer *Notenzeile* durch eine Marke (*1) geholt werden kann. In diesem Fall reicht die Form **reg{...}** aus. Beides zusammen ist möglich, aber eher unpraktisch.

Ich erwähnte, dass in einer *Registerzeile* das Kommando ***preset*** als Erstes gelesen würde, auch ohne am Anfang stehend. Dazu ist kein Scanner nötig, denn die Reihenfolge der Datenübertragung erledigt das nebenbei. Mit einem erkannten *Preset–Kommando* wird Routine **pre:** gerufen, aber die Übertragung in echte *Register* oder *Records* geschieht erst am Ende der Registerzeile (nämlich nach `UNTIL` in **regline**).

Routine **regline** ruft, wie geschildert, ggf. **settle–A** auf, sobald **reg X** verändert wird. Das ist aber auch der Fall, wenn in einer *Notenzeile* **A{...}** ein *habitus* (z. B. *1) auftaucht. Dieser bewirkt nach Durchlaufen des FIFO–Pufferspeichers den Aufuf der Routine **behave–A**, die ihrerseits **settle–A** verwendet.

```
: behave-A  ( u --)
            dup old-habit-A_ <>
      IF    dup :pool reg-A recwid xmove
            settle-A
      THEN  to old-habit-A_ ;
```

So, und ehe ich mit der Abgabe des Textes wieder zu spät komme, wie bei *Heft 4/2011*, beschließe ich für diesmal meine Ausführungen. Bis zum Erscheinen von *Heft 2/2012* gibt es sicherlich auch neue Tonbeispiele mit neuen Effekten.

Beachten Sie bitte den Dateibereich der Website der Forth–Gesellschaft unter
`http://www.forth-ev.de/filemgmt/viewcat.php?cid=54`
sowie die Site des Autors unter
`http://www.stocket.de/WE`

Abbildung 2: Der Autor (einst) bei ersten Playback-Versuchen

Semantics

Bernd Paysan

Der ANSI–Standard definiert vier verschiedene „Semantics," wobei Interpretations–, Compilations– und Run–Time–Semantics ziemlich klar umrissen sind, die Execution–Semantics aber eher nicht — aber genau die wird als Basis für die Definition der meisten Wörter verwendet. Ich versuche, da ein wenig Licht ins Dunkle zu bringen, und schlage vor, das System etwas zu ändern.

Ausgangspunkt der Diskussion

Der Ausgangspunkt der Diskussion war, Gforth mit einem smarten COMPILE, zu versehen, so wie das Stephen Pelcs VFX inzwischen auch macht. Die Idee ist älter, der Cross–Compiler, der bei Gforth dabei ist, hat schon seit 20 Jahren ein vergleichbares System. Jedes Wort dort („Ghost" genannt) hat eine ganze Reihe Funktionen, die die verschiedenen Semantics abbilden. Das Konzept nennen wir „Monotoken," weil es nur ein einziges Token gibt, das Namen, Interpretations– und Compilations–Semantik und was sonst noch alles gebraucht wird, vereint.

Die unterschiedlichen Teile davon sollten in eine virtual function table (VT) im objektorientierten Stil gelangen, mit Ausnahme dessen, was man für EXECUTE braucht — EXECUTE ist performance–kritisch und soll daher so behandelt werden wie bisher. Am Ende gab es dann eine VT mit sechs Funktionen: COMPILE„ INT–XT, COMP–XT, TICK–XT, RUN–PRELUDE und EXTRA–CODE. Die Prelude ist ein Konzept von Manfred Mahlow, welches sich mit so einer VT etwas eleganter integrieren lässt, und EXTRA–CODE braucht man z.B. für DO-DOES.

Die Wörter INT–XT und COMP–XT findet Anton Ertl „komisch," und ich finde TICK–XT komisch. Wozu sind sie überhaupt gut? Zur Kompatibilität mit dem aktuellen Gforth, weil ich auf Flags im Header wie IMMEDIATE und COMPILE–ONLY verzichten will. Eigentlich würde ich INT–XT und COMP–XT durch EXECUTE und COMPILE, ersetzen, das geht aber so nicht — sagt Anton, das würde das COMPILE, kaputt machen (natürlich nicht in der Default–Variante, sondern in besonderen Fällen). Und TICK–XT würde ich gerne weglassen, denn das dient nur dazu, einen Fehler zu produzieren, wenn man etwas tickt, was man nicht ticken darf (also COMPILE–ONLY). Etwas, was man aber immer noch legitimerweise COMPILE,n könnte.

Nähere Betrachtung des Standards

So, was ist nun Sache? Der Standard hat die verschiedenen Semantics wie folgt definiert:

Interpretation Das Verhalten eines Wortes, wie es vom Text–Interpreter im Interpretations–State ausgeführt wird

Compilation Das Verhalten eines Wortes, wie es vom Text–Interpreter im Compilations–State ausgeführt wird — zur Compilationszeit

Run–Time Das Verhalten dessen, was da compiliert wird, zur Laufzeit des Wortes, in das hineincompiliert wurde.

Execution Das Verhalten eines xts (execution tokens), wenn man es mit EXECUTE ausführt — bzw. mit COMPILE, in ein Wort ablegt, dann wird das entsprechende Verhalten zur Laufzeit ausgeführt

Die meisten Wörter im Standard sind über ihre Execution Semantics definiert. Das impliziert zunächst mal, dass es ein mit ' ermittelbares xt gibt, welches mit EXECUTE ausgeführt und COMPILE, compiliert werden kann, aber hier ist der Standard nicht konsistent. Es gibt Wörter, die haben zwar eine Execution Semantics, aber keine Interpretation Semantics. Das sind Wörter, die am Returnstack werkeln, aber auch LEAVE und COMPILE,. LEAVE kann am Returnstack werkeln (traditionelle Implementierung — Ausstiegsadresse wird als dritter Wert von DO auf den Return–Stack gepusht), muss aber nicht — in Gforth z.B. wird die Ausstiegsadresse direkt angesprungen, sie muss nicht auf dem Returnstack liegen. LEAVE ist hier wie andere Kontrollstrukturen auch implementiert, es erzeugt zur Compile–Zeit einen Sprung. Dann sollte es auch wie eine solche beschrieben werden, also mit Compilations– und Run–Time–Semantics.

Was ist aber mit COMPILE,? Das kann man doch eigentlich auch im Interpretations–State ausführen, solange es nur eine aktuelle Definition gibt. Mitch Bradley hat dazu folgendes nachgeschoben, was aus seinen Überlegungen für die Einführung von COMPILE, hervorgeht, aber nicht Eingang in den Rationale–Teil des Standards gefunden hat:

COMPILE, ist das fehlende Stück, mit dem ein User sich einen eigenen Text–Interpreter schreiben kann. Der sollte dann wie folgt aussehen:

```
bl word find dup IF
    state @ IF
        \ compilation semantics
        0< IF  compile,  ELSE  execute  THEN
    ELSE
        execute \ interpretation semantics
    THEN
ELSE
    ... handle things like literals ...
THEN
```

Die Abfrage wird oft mit

```
0< state @ and IF compile, ELSE execute THEN
```

auf einen Einzeiler verkürzt, zur Verdeutlichung lasse ich das so, wie MITCH es geschrieben hat.

COMPILE, wird in diesem Szenario nur ausgeführt, wenn man im Compilations–State ist. Und zwar nur auf nicht–immediate–Wörter. Es ist das Äquivalent zu EXECUTE für den Compiler, so steht's auch im Standard. Wenn der User dieses Konstrukt verwenden kann, um seinen eigenen Compiler zu schreiben, dann *ist* das der Compiler, plus minus Feinheiten. Das FIND darf übrigens state–smart sein, also je nach STATE verschiedene xts zurückliefern — für die Interpretation– oder Compilation–Semantics.

Das verschiebt die Probleme von State–Smartness nur, weshalb ANTON und mir FIND auch überhaupt nicht gefällt. Nicht nur, weil es einen Counted String nimmt, sondern weil es ein xt und eine Flag zurückliefert, die Flag wichtig ist (EXECUTE oder COMPILE,?) — aber nur, wenn man compiliert, und zu allem Übel das xt von STATE abhängen kann.

So, jetzt haben wir bei ' gelernt, dass xt+EXECUTE äquivalent zur Interpretation–Semantics sind. Beim Ergebnis von FIND gilt das nicht — das kann auch eine Compilation Semantics sein. Man sollte dabei auch tunlichst vermeiden, STATE zwischen FIND und EXECUTE zu ändern — denn im Interpreter passiert das ja auch nicht. Die Execution Semantics eines xts kann also dreierlei sein:

Mit EXECUTE die Interpretation Semantics, wenn das xt von :NONAME, ' oder FIND im Interpretations–State kommt

Mit EXECUTE die Compilation Semantics, wenn das xt von FIND im Compilations–State kommt und das Wort immediate ist

Mit COMPILE, die Run–Time Semantics, wenn das xt von :NONAME, ' oder FIND im Compilations–State kommt und das Wort nicht immediate ist

Das ist zumindest der wohldefinierte Teil.

Postpone

Während COMPILE, die Execution–Semantics compiliert, compiliert POSTPONE die Compilations–Semantik. Die Motivation zu POSTPONE statt COMPILE und [COMPILE] war ja, dass man als Nutzer nicht unbedingt bei jedem Wort wissen kann, ob das jetzt immediate ist. Also, POSTPONE sollte so definierbar sein:

```
: postpone ( „name“ -- )
  bl word find dup ?notfound \ throw error
  0< IF  postpone literal postpone compile,
  ELSE  compile,  THEN ; immediate
```

So ist es aber nicht spezifiziert. Die Compilations–Semantik von Wörtern wie IF oder S“ sind ja eindeutig beschrieben. Um das zu implementieren, muss das System ein „compilation token“ haben, wie das COMP–XT oben oder Gforths aktuelles NAME>COMP, das aus einem Name–Token das Compilations–Token bildet (das kann es, weil in Name–Token ja die immediate–Flag drin ist). Das CVS–Gforth (mit seinem Recognizer) macht für Forth–Wörter dann bei Postpone im Endeffekt folgendes

```
: nt, ( nt -- )  name>comp execute ;
\G compiliert ein Name-Token
: postpone ( „name“ -- )
  parse-name find-name dup ?notfound
  postpone literal postpone nt,
; immediate compile-only
```

Hier können wir das, aber die klassische Implementierung kennt kein Name–Token. Sie kennt auch kein compile–only, sondern hat (wie fig–Forth) ein ?COMP, das den State abfragt. Wie State–Smart–Wörter ein „Misfeature“, wie Anton es nennt. Das bedeutet aber, dass folgendes nicht geht:

```
: my-if postpone if ; \ nicht immediate!
: foo [ my-if ] .“ yes!“ then ;
```

Wenn wir POSTPONE so definieren wie am Anfang des Abschnitts, dann haben wir noch eine weitere Execution Semantics:

Mit COMPILE, die Compilation Semantics zur Laufzeit, wenn das xt von FIND im Compilations–State kommt und das Wort immediate ist.

Konsequenzen

Die Erkenntnis ist: Das geht so nicht. Der Standard ist nicht konsistent, die Änderung, die am wenigsten Auswirkungen hat, wäre, die Beschreibung der Wörter, die jetzt unter „Execution Semantics“ eingeordnet ist, als Interpretation+Run–Time Semantics zusammenzufassen (Compilation Semantics ist dann, wenn nicht anders erwähnt, das Anhängen der Run–Time Semantics an das gerade definierte Wort).

Das andere, was so nicht geht, ist mein Header–Vorschlag von oben. Das mit der Flag bei Immediate–Wörtern ist zu tief im Forth drin, als dass man es signifikant anders implementieren könnte. Wenn man keine state–smarten Wörter haben will (wollen wir nicht) und wenn man kein FIND haben will, das abhängig von STATE verschiedene xts zurückliefert (wollen wir auch nicht), dann bleibt eigentlich nur die Implementierung von VFX übrig: immediate (und compile–only) wird nach wie vor im Header geflagt (d.h. immediate–Wörter unterscheiden sich von solchen mit besonderer Compilations–Semantik), und es gibt nur zwei sichtbare „Methoden“ für ein Wort: EXECUTE und COMPILE,. EXTRA–CODE wird ja nur intern verwendet, und RUN–PRELUDE muss man sich auch nochmal überlegen — der vom User geschriebene Text–Interpreter kennt das nicht, und kann damit nichts anfangen.

Bleibt die offene Frage, ob COMPILE, dadurch „kaputt gemacht“ wird, wie ANTON es formuliert — weil die mit den neuen Methoden erstellten Wörter mit spezieller

Compilation Semantics (wie TO, IS, S“ und so) ja über COMPILE, laufen müssen (sonst geht der vom User gebaute Text–Interpreter nicht), damit aber die Spezifikation von COMPILE, im Standard verletzen (weil es ja etwas anderes tut als die Execution Semantics compilieren). Also wird man auch hier die Spezifikation ändern müssen, um COMPILE,s Absicht gerecht zu werden „das Äquivalent zu EXECUTE für den Compiler zu sein.“ (Rationale für COMPILE,).

Vorschläge

Andrew Haley macht den Vorschlag, zurück zu den Wurzeln zu gehen, also keine Extrawürste braten, was spezielle Compilations–Semantik betrifft, und Implementierungsdetails wie die Code–Generierung eines Native–Code–Compilers für den Benutzer unsichtbar zu machen. Die Erkenntnis, dass ein moderner Compiler ohnehin einen Tokenizing–Schritt braucht, legt nahe, dass das, was der klassische ITC–Compiler macht, gar nicht so falsch ist (denn der legt einfach nur die Tokens alias xt der Reihe nach in dem Speicher ab). Das heißt dann aber auch, Guru–Code state–smart zu machen, und den Gurus zu überlassen, die damit zurechtkommen, und mit den Limitierungen zu leben. Damit gibt man den Versuch auf, von der Implementierung zu abstrahieren — moderne Techniken unterscheiden sich nur „im Hintergrund“ von traditionellen.

Das COMPILE, in diesem System erledigt dann nur den Schritt der Tokenisierungs–Phase. xts hätten dann (in einem System wie oben beschrieben) ein NATIVE–COMPILE„ das den Schritt vom Token zum Native Code macht, das aber für den Benutzer uninteressant ist — diese Übersetzung wird für den Nutzer transparent erledigt. Man könnte wieder vorschreiben, welches Wort im Standard immediate ist und welches nicht, weil sich das nicht mehr zu ändern braucht. Vielleicht muss man etwas genauer hingucken, weil LEAVE und EXIT heute oft immediate sind — LEAVE compiliert einen Sprung hinter LOOP, statt auf den Returnstack zuzugreifen, und EXIT räumt die lokalen Variablen auf.

Stephen Pelc dagegen meint, lieber nach vorne zu gucken, und mit neuen Implementierungen von Interpretations– und Compilations–Semantik wie das intelligente COMPILE, zu experimentieren und zu sehen, wie sich das bewährt und ob es sich durchsetzt — und vielleicht sind in 20 Jahren IMMEDIATE, STATE und POSTPONE Geschichte.

Gforth hat inzwischen ein „Recognizer“–Konzept bekommen, also eine Faktorisierung des Interpreters, mit denen man die Behandlung von Literals und anderen Sachen einhängen kann. Mit dem Recognizer erübrigt es sich für den Nutzer, den Interpreter neu schreiben zu müssen, wenn er nur Teile davon ändern will — er hängt einfach einen Recognizer ein. Kritische Wörter wie S“ oder TO kann man hier als Recognizer implementieren, die „ganz normale String–Literale“ (mit Anführungszeichen vor und hinter dem String) implementieren, oder ein „>“–Prefix (ohne Space) vor einem Value, Local oder deferred Word wird als Zuweisung interpretiert. Der Recognizer behandelt alles, inklusive POSTPONE.

Das wird bestimmt ein Workshop–Thema auf der EuroForth, und noch mehr Diskussion anregen.

Fazit

ANS Forth hat den heroischen Ansatz gewagt, von einem Implementierungsstandard zu einem Kommunikationsstandard zu gehen, und das an vielen Stellen auch erfolgreich durchgezogen. Dazu gehört auch, dass Wörter im Standard nicht mit „immediate“ oder „compile–only“ geflagt sind (wie in Forth–83 und davor), sondern eben die verschiedenen Semantics eingeführt wurden. Allerdings ist ANS Forth hier den Weg nicht ganz zu Ende gegangen, was man schon daran merkt, dass einige Wörter explizit als „immediate“ bezeichnet werden (die Kommentar–Wörter (und \ z.B.), statt ihnen einfach gleichlautende Interpretations– und Compilations–Semantik zu geben.

Das Konzept mit STATE und Flags im Header ist eines aus den 70ern, und Chuck Moore hat das auch nicht so richtig gefallen, weshalb er verschiedene Ansätze gemacht hat: Von State–dumb–Wörtern wie CHAR und [CHAR], die niemand so recht mag (eigentlich ist nur ’ und [’] übrig geblieben), über getrennte Wortlisten in cmForth bis zu ColorForth, bei dem die Behandlung explizit über den Farb–Code vorgegeben wird — man aber immer noch wissen muss, ob ein Wort nun eine Kontrollstruktur ist, die zur Compilationszeit ausgeführt wird, oder ob es einfach nur compiliert werden muss.

Bootmanager und FAT–Reparatur: Elfter Fort(h)schritt (hide&activate–mbr&ebr) Teil A

Fred Behringer

Was das Schreiben eines VD–Artikels betrifft, hat man zunächst einmal eine nur ungefähre Vision. Im Laufe der Ausarbeitung muss man dann aber schmerzvoll feststellen, dass man sich mit der verbleibenden Zeit heillos übernommen hat. So auch hier. Was mache ich also? Ich unterteile einfach: In Teil A behandle ich „nur“ primäre Partitionen (und eine eventuelle erweiterte Partition im Ganzen) und für Teil B (das mir dann wieder weitere 3 Monate Zeit lässt) werde ich das „Durchhangeln“ durch die erweiterte Partition in Angriff nehmen.

Meine Suche bei Google brachte unter dem Begriff „Logisches Laufwerk verstecken“ bei Powerforen.de als Antwort auf die dort von jemandem gestellte Frage „Kann ich ein logisches Laufwerk in einer erweiterten Partition verstecken? (thx@all)“ u.a. folgende drei Reaktionen: „Du brauchst ein Programm, um Partitionen verstecken zu können, z. B. PartitionMagic.“ (Mein Kommentar dazu: hm. . .)

„Oder Windows 2000 bzw. XP (Im Datenträgermanager einfach den Laufwerksbuchstaben entfernen). Oder LINUX. Gibt auch Bootmanager, die das beherrschen. . . “ (Mein Kommentar dazu: hm, hm. . .)

„Aber mit Partition Magic kann man nur primäre Partitionen verstecken oder vielleicht die gesamte erweiterte. Aber die Möglichkeit, ein logisches Laufwerk zu verstecken, gibt es bis Version 7.0 jedenfalls nicht.“ (Mein Kommentar dazu: sic!. . . Aber was interessieren mich die diversen Versionen von Partition–Magic, wenn ich nichts weiter machen will, als einfach nur den erweiterten Interrupt 13h zum Beschreiben des MBRs aufzurufen?)

Uff! Im vorliegenden Artikel zeige ich, wie einfach es geht — wenn man Forth einsetzt und sich nicht an das klammert, was in Foren zu lesen ist, sondern wenn man sich wirklich die Mühe macht, Unterlagen zu studieren. Ich verwende das Erarbeitete aus den bisherigen Artikeln meiner Serie und baue mir so eine Art von Mini–Partition–Manager–System (nach dem Lego–Baukasten–Prinzip) auf. (Wenn ich mit dem Wissen von heute nochmal 20 sein dürfte, würde ich anfangen, einen Bootmanager mit allem Pipapo (in Forth!) zu bauen. Das geht ja nun leider nicht mehr und ich tue gut daran, mich zu beschränken. So ist mir UEFI und so sind mir die GPT–Bestrebungen (GUID Partition Table) zwar bewusst, aber ich gehe ihnen, hier jedenfalls, tapfer aus dem Wege.)

Mit Hilfe des Mini–Partition–Manager–Systems lasse ich mir bei sämtlichen Laufwerken (primär oder logisch) anzeigen, ob es versteckt (hidden) oder nicht versteckt ist, ob aktiv (bootbar) oder nicht aktiv, ob in einer primären oder in der erweiterten Partition befindlich. Gleichzeitig werden Möglichkeiten bereitgestellt, all das (jedes gesondert) durch Eingabe von „n verstecken“ oder dergleichen zu ändern. Dazu brauche ich keine Version xyz von irgendeinem Profi– oder Semiprofi–System. Dazu muss ich nur wissen, welches Byte im MBR zu ändern ist! (Die Partitionsnummer n wird mit allen ihren Kollegen am Bildschirm angezeigt und entspricht der Reihe nach den (z.B. von DOS zugeordneten) Laufwerkbuchstaben.)

Ein Schreck in der Abendstunde,

der dazu beigetragen hat, den vorliegenden Artikel (fürs weitere schnelle Experimentieren in der Fix–und–fertig–Kurzversion) zu verfassen, war folgender. Ich hatte unter DOS (geht prima!) einen als Festplatte eingerichteten USB–Stick nach dem Motto–Hairu–Verfahren von Panasonic (device=USBASPI.SYS /v /w /u device=Di1000dd.SYS) eingebunden. Dann hatte ich per XFDISK (geht auch bestens) nach WIN–XP geschaltet. XP kann natürlich auf DOS zugreifen! Dann war ich dazu übergegangen, ein logisches Laufwerk des Sticks (den hatte ich mit einer primären Partition und mehreren logischen Laufwerken versehen) zu löschen. Und plötzlich war auch das Laufwerk L: weg. Einfach weg! Weder über den XP–Explorer noch bei Rückkehr zu DOS irgendwie noch auffindbar. In der Folge der Laufwerk–Buchstaben wurde aber eine Lücke hinterlassen, sowohl von XP wie auch von DOS aus gesehen. Man vergleiche das mit dem oben aus den Powerforen.de Berichteten. Was war geschehen? Eigentlich nichts Schlimmes! Im ebr (im extended boot record) des Laufwerks L:, nicht etwa im mbr (im master boot record der Bootplatte), war an der Offset–Stelle 1be+4 der Wert 06 (FAT16) von mir zunächst unbemerkt und total unbeabsichtigt nach 16 (FAT16–versteckt) gerutscht. Aber wie soll man denn auch auf solch einen Gedanken kommen, wenn man sich nur mit Ungefähr–Berichten beschäftigt hat? Da nützen auch keine noch so intelligenten Programm–Pakete etwas! Mit dem im Vorliegenden zu entwickelnden Menü, das ich zusammen mit (der 16–Bit–Version von) Turbo–Forth (sucht man sich bei taygeta.com) zu einem klitzekleinen .com–Programm abspeichern kann, lässt sich dieser spezielle Punkt (und noch viel mehr) mit einem einzigen schnellen Tastendruck klären. (Natürlich kann ich das betreffende Prograämmchen unter DOS bis ME aufrufen — und ebenso natürlich ab XP natürlich nicht mehr! Uff! Unter XP kommt die Meldung „Ein Programm hat versucht, auf die Festplatte direkt zuzugreifen. Das ist in diesem System nicht zulässig.“ — Na ja!)

Vorteile, Nachteile und andere Erkenntnisse

Das wohl zunächst unter Linux entwickelte Programm GParted wird vielerorts gerühmt. Für den an sich guten Prozessor AMD–K6–2 ist es aber nicht brauchbar, da da „der Befehl cmov fehlt“. (Na ja! Als Ausweich–Lösung habe ich mir hierfür das fast so gute Programm QtParted zurechtgelegt.)

Linux hat die FS–Kennung 83 im Offset 1be+4. Damit kann dort das „Verstecken“ mit Hilfe einer 1 im vorderen Halbbyte nicht funktionieren.

Bei FAT32, und auch bei NTFS, aber auch bei HPFS aus 0S/2, funktioniert das Verstecken genau so gut wie bei FAT16 — auch in logischen Laufwerken!

Eine DOS–Partition kann nicht größer als 2 GB gemacht werden. (Von FreeDOS will ich hier nicht sprechen. DOS 6.20 ist für mich der Standard.) Der für DOS (FAT16) vorgesehene Bereich darf insgesamt (unter Einbeziehung einer erweiterten Partition) nur 8 GB groß werden.

Die erweiterte Partition darf ruhig größer werden (um z.B. auch Linux als logisches Laufwerk aufzunehmen), nur muss dann die FS–Kennung von dem von den „professionellen“ Partitionierungsprogrammen für Windows 95 vorgesehen Wert 0f auf den DOS–Wert 05 zurückgesetzt werden. Ich habe dafür in meinem Mini–Partitionierungs–System eine Einstell–Möglichkeit vorgesehen.

Die logischen Laufwerke dürfen unterschiedlich formatiert sein.

Ob in einer erweiterten Partition mit der Gesamt–FS–Kennung 05 auch logische Laufwerke auftreten dürfen, die anders formatiert sind (z.B. 07 im ebr für NTFS), habe ich hier nicht untersucht.

Hier nur Boot–Festplatte

„aktiv“ (Medium, von dem gebootet wird) heißt bei mir hier Offset (1be)=80, inaktiv (1be)=00. Mit den bisherigen Mitteln könnte man auch hier (1be)=81 schalten und damit eine zweite Festplatte oder einen USB–Stick erreichen. Mit den Mitteln von Teil 5 meiner Artikelserie lässt sich das leicht ergänzen. Es erfordert aber einen etwas größeren Aufwand. Ich verzichte hier darauf und verschiebe die Anstrengung auf Teil B.

Voraussetzungen

Ich arbeite hier mit Turbo–Forth in der 16–Bit–Version. Man findet es auf dem amerikanischen FTP–Server taygeta.com oder auf einem seiner Spiegel–Server. Über ZF mache ich mir im Vorliegenden keine Gedanken.

Turbo–Forth arbeitet mit den Escape–Sequenzen und benötigt daher den Treiber ansi.sys in der config.sys (z.B. für hervorgehobene OK–Anzeigen.). Wenn aus irgendeinem Grunde kein (von DOS her geläufiger) ansi.sys vorhanden ist, kann man auch ansi.com (siehe Google) verwenden. (Der Einbau von ansi.com kann auf keinen Fall schaden!)

Alle hier benötigten Forth–Worte aus früheren Artikeln der vorliegenden Serie werden in kompakter Form und ohne viel Kommentar auch in diesem Artikel wiederholt. (Das Schöne an Forth ist, dass man beliebig oft neudefinieren kann, ohne von Fehlerabbrüchen gestört zu werden.)

Inbetriebnahme

Das einzige im vorliegenden Artikel benötigte Forth–Wort, mit welchem alles erschlagen wird, ist show–hide&act. Es läuft alles interaktiv. Alle Eingabemöglichkeiten werden am Bildschirm angezeigt und alle Ausgaben werden dort hinreichend erläutert. Veränderungen spielen sich alle im Sektor–Puffer sectbuf (der Begriff darf sofort wieder vergessen werden!) ab. Die für die Handlungen im vorliegenden Artikel zulässigen Eingaben werden bei Aufruf von show–hide&act (jedesmal aktualisiert) angezeigt.

Es wird jedesmal die gesamte Partitions–Tabelle (mit ihren Hex–Werten in den vier 16–Byte–Partitions–Zeilen) ausgegeben. Rechts wird aber angezeigt, ob die betreffende Partition versteckt oder nicht, aktiviert oder nicht, erweitert oder primär, für DOS vorbereitet oder für Windows eingerichtet ist.

An der Festplatte (an der Boot–Festplatte) ändert sich so lange nichts, bis man (über die Tastatur) „anwenden“ (die übliche Denglish–Version von apply) eingibt. Dann allerdings wird der Sektor–Puffer (in seiner Gesamtheit) neu in den MBR (in den Master–Boot–Record) der Festplatte geschrieben — und ist in seiner bisherigen Form ein für alle Mal verloren, wenn man ihn nicht vorher gesichert hat!

Wenn man gleich zu Beginn show–hide&act eingibt, wird man sich über den Salat von Hexwerten wundern: Das sind Zufallsreste aus dem Turbo–Forth–System. Den eigentlichen MBR bekommt man auf den Bildschirm anfangs erst nach Eingabe von getmbr.

Vorsicht vor einer unüberlegten Eingabe von „anwenden“! Anwenden schreibt den gerade im Sektor–Puffer befindlichen Inhalt auf die Boot–HD in deren MBR! Und dieser Puffer–Inhalt ist anfangs (nach Aufruf von show–hide&act), alles andere als hilfreich!

Man speichere das zu show–hide&act gehörige Forth–Paketchen, mit den Forth–Zubringer–Worten, ab, z.B. als show–hide&act.fth, und rufe Turbo–Forth per forth include hide.fth auf. Natürlich kann man das „Gesamtpaket“, Turbo–Forth plus include hide.fth, auch per save–system zu einer .com–Datei verarbeiten.

Nicht alle Eingabe–Kombinationen sind sinnvoll. Erster Ansprech–Partner für Informationen ist Wikipedia. Guten Rat bekommt man von der MS–Knowledge–Base, auch wenn man dort die Dinge, die nicht zu MS gehören, weniger leicht findet. Die Interrupt–Sammlung von Ralf Brown leistet gute Dienste. Ich lasse Eingaben, die „sich beißen“, als ??? ausgeben.

FS–IDs mit 05 (sichtbar, erweitert) oder 15 (versteckt, erweitert) sind eindeutig gekennzeichnet. Ähnlich für 0f oder 1f. Es gibt aber viele FS–IDs, bei denen von der DOS–Organisation her gesehen nicht feststeht, was gemeint ist. Ein Beispiel ist die Kennung 83 für Linux–Partitionen (weder versteckt noch nicht versteckt). Ich gebe solche Fälle in der Partitions–Tabelle mit einer Reihe von Fragezeichen aus. Das würde (im vorliegenden Artikel) auch z.B. für eine zweite Festplatte (81 statt 80 im Offset 1be) gelten. Unter „primär“ (im Offest 1be+4) fasse ich alles zusammen, was nicht sonstwie (als FS–ID) eindeutig gekennzeichnet ist.

Zum Experimentieren darüber, ob bei der einen oder anderen Eingabe die Meldung ??? erscheint oder nicht, habe ich noch schnell das Wort hex! vorgesehen. Es läuft nach dem Schema: n cd hex! Dabei ist n die in das betreffende Byte des Sektor–Puffers zu speichernde Hexzahl, c die in der Partitionstabelle angezeigte Zeilennummer und d das Zeilen–Offset (Zählung von 0 bis 3). c und d sind hexadezimal gesehen die beiden Halbbytes im Verbund des Bytes cd.

Eingaben, die am momentanen Sektor–Puffer nichts ändern, wie beispielsweise „verstecken“, werden mit der Meldung „Schon versteckt“ (oder Entsprechendes) quittiert.

Und ein paar Absicherungen sind auch eingebaut: Wenn mehr als nur eine primäre Partition aktiv–geschaltet ist, erscheint eine Warnung. Wenn mehr als nur eine erweiterte Partition vorhanden ist (man überlege sich, wie man sowas überhaupt zustandebringen könnte — über hex! geht das), erscheint ebenfalls eine Warnung. Es wird nur gewarnt. Verboten wird sowas nicht. Man vergleiche das mit den diversen Fix–und–fertig–Programmen gparted, qtparted, partition–wizard, R.I.P., partition–table–doctor und wie sie alle heißen, mit ihren gutgemeinten, aber viel zu strengen Verboten und Beschränkungen!

Falls gar keine erweiterte Partition vorhanden ist, wird das auch angezeigt. Bei mehr als nur einer erweiterter Partition wird nicht zwischen „für DOS vorbeitet“ und „für Windows vorbereitet“ unterschieden. Bezüglich aktiviert und versteckt werden alle vier Möglichkeiten (auch aktiviert und sichtbar, deaktiviert und versteckt, deaktiviert und sichtbar) als „vernünftig“ betrachtet. (Was möglich ist, wird auch zugelassen!)

Im vorliegenden Teil A wird nur am MBR operiert. Die Verarbeitung von EBRs wird auf Teil B verschoben.

Mögliche Tastatur–Eingaben

`show-hide&act`	ruft das Gesamt–Menü auf
`verstecken`	setzt das von DOS her geläufige vordere Halbbyte auf 1
`sichtbar-machen`	nimmt diese 1 vom Halbbyte wieder weg
`aktivieren`	legt die Partition fürs Booten fest (Offset 1be=80)
`deaktivieren`	nimmt die Bootbarkeit wieder zurück (Offset 1be=00)
`erweitern-fuer-DOS`	macht (bei FAT16) die ersten 8 GB für DOS verwendbar
`erweitern-fuer-Win`	bereitet die erw. Part. für WIN vor. DOS ignoriert das.
`anwenden`	schreibt den momt Inhalt des Sektor–Puffers in den MBR
`hex!`	ermöglicht die Eingabe beliebiger Bytes in den Puffer

In Teil B findet man dann

das mich eigentlich Überraschende an der diesmaligen Übung in Forth, nämlich dass man nur ein einziges Halbbyte in der Abfolge von EBRs umzusetzen braucht — und schon das betreffende logische Laufwerk über das übliche Datei–Management nicht mehr findet. Am allerüberraschendsten war für mich dabei, dass selbst XP (das natürlich den vollen Zugriff auf die FAT–16–Laufwerke in der erweiterten Partition hat) dann ebenfalls nur noch eine Lücke im Laufwerk–Alphabet sieht und an das weggeschaltete logische Laufwerk (zumindest über den Explorer) nicht mehr herankommt. Und die Krone der Überraschung lag in der Erkenntnis, dass in den Internet–Foren viel von jenen Dingen aufbewahrt und immer noch diskutiert wird, die man schon längst mit einem einzigen Federstrich an einer zentralen Literatur–Stelle hätte zusammenfassend klarstellen können.

Die Vorbeitungen für das Heft 2/2012 der Vierten Dimension sind in vollem Gange und ich will mich nicht treiben lassen. Ich bringe also hier nur jenen Teil meines Vorhabens, der die primären Partitionen verarbeitet (Teil A). Die logischen Laufwerke einer erweiterten Partition sollen im Teil B hierzu ergänzt werden.

An sich brauche ich nur die Ausarbeitungen von Teil 5 meiner Artikelserie herzunehmen und sie anzupassen. Man weiß aber aus Erfahrung, dass solche „Anpassungen“ ihre Tücken haben, und tut gut daran, sich Zeit zu lassen. Ich hatte ja in Teil 5 ein paar Dinge umgekrempelt (Anwählbarkeit von mehr als einer Festplatte oder auch USB–Sticks). An sich ist es aber auch eine Herausforderung an den geneigten Leser, selbst mit Vorschlägen aufzuwarten.

Literatur

Google: Wikipedia.de
Google: Ralf Brown's Interrupt List.
Google: MS–Knowledge–Base
Google: DOS Batch files — Using ANSI sequences (ansi.com statt ansi.sys)
Google: gparted
Google: partition–wizard
Google: partition–table–doctor
KNOPPIX: Qtparted
Uni–Bremen: Mirror von `ftp://ftp.taygeta.com/pub/Forth/Compilers/native/dos/turbof.zip` 326 KB.
Wikipedia: Recovery Is Possible (R.I.P.)

Wegen einer eingedeutschten Fassung wesentlicher Teile von Turbo–Forth setze man sich mit dem Autor des vorliegenden Artikels in Verbindung.

Listings

```
\ ************************************************
\ *                                              *
\ * BOOTM-11.FTH                                 *
\ *                                              *
\ * Zutaten fuer FAT-Reparatur und Bootmanager unter *
\ * Turbo-FORTH-83  (ohne Berueecksichtigung  von ZF) *
\ *                                              *
\ * Fred Behringer - Forth-Gesellschaft - 01.06.2012 *
\ *                                              *
\ ************************************************

hex             \    Alle Zahlenangaben hexadezimal !

\ Glossar der im vorliegenden Artikel zur Anwendung kommenden Worte
\ ----------------------------------------------------------------

\ sectbuf        ( -- ad ) Konst., Adr. d. Sektorpuffers, 1 Sekt. = 512 Byte
\ (getsect)      ( seite spur/sektor -- ) Sektor von HD holen
\ (putsect)      ( seite spur/sektor -- ) Sektor auf HD schreiben
\ getmbr         ( -- ) 0 1 (getsect) , MBR von Boot-HD nach sectbuf holen
\ putmbr         ( -- ) 0 1 (putsect) , sectbuf als MBR auf HD schreiben

\ show-hide&act  ( -- )     Tabelle: hidden?, active?, 05 fuer DOS?, ...
\ aktivieren     ( n -- )   Offset 1be   = 80
\ deaktivieren   ( n -- )   Offset 1be   = 00
\ verstecken     ( n -- )   Offset 1be+4 = 1x
\ sichtbar-machen ( n -- )  Offset 1be+4 = 0x
\ erweitern-fuer-DOS ( -- ) Offset 1be+4 = 15 oder 05
\ erweitern-fuer-Win ( -- ) Offset 1be+4 = 1f oder 0f
\ anwenden       ( -- )     = putmbr (Sekt-Puffer in den MBR der Boot-HD)
\ errext?        ( -- )     Fehlermeldung: Mehr als eine erweiterte Part.
\ erract?        ( -- )     Fehlermeldung: Mehr als eine aktive Partition
\ errpri?        ( -- )     Fehlermeldung: Mehr als eine sichtb, Primpart
\ ewtrWIN        ( -- )     Anzeige: Fuer Win95, von DOS nicht anerkannt.
\ parttab        ( -- )     Partitionstab. mit Anzeige von versteckt usw.

\ Benoetigte Worte aus frueheren Artikeln dieser Serie (Kommentare dort)
\ ----------------------------------------------------------------------

hex

210 allot here  here 0f and - 200 -
     constant sectbuf                   \ Sektor-Puffer-Anf.

code (getsect) ( seite spur/sektor -- ) \ Sektor von Boot-HD
     ds push  es pop  80 # dl mov  cx pop  ax pop  al dh mov
     sectbuf # bx mov  201 # ax mov   13 int   next end-code

: getmbr ( -- ) 0 1 (getsect) ;         \ MBR von d. Boot-HD

code (putsect) ( seite spur/sektor -- ) \ Sektor auf Boot-HD
     ds push  es pop  80 # dl mov  cx pop  ax pop  al dh mov
     sectbuf # bx mov  301 # ax mov   13 int   next end-code

: putmbr ( -- ) 0 1 (putsect) ;         \ MBR auf d. Boot-HD

\ Hilfsworte zum vorliegenden Artikel
\ -----------------------------------

```

```
: errext? ( -- ) \ Fehlermeldung bei mehr als einer erweitert. Partition
    0 05 1 do sectbuf 1be + i 1 - 10 * + 4 + c@
               dup 05 = if swap 1 + swap then
               dup 15 = if swap 1 + swap then
               dup 0f = if swap 1 + swap then
                   1f = if swap 1 + swap then
           loop
    dup 1 > if   ."    Fehler(?): " dup . ." erweiterte Partitionen" cr
            then
        0 = if   ."    Keine erweiterte Partition vorhanden"         cr
            then ;

: erract? ( -- ) \ Fehlermeldung bei mehr als einer aktiviert. Partition
    0 05 1 do sectbuf 1be + i 1 - 10 * + c@ 80 = if 1 + then loop
    dup 1 > if   ."    Fehler(?): " dup . ." aktivierte Partitionen" cr
            then
        0 = if   ."    Keine aktivierte Partition vorhanden"         cr
            then ;

: errpri? ( -- ) \ Fehlermld. bei mehr als einer sichtb. prim. Partition
    0 05 1 do sectbuf 1be + i 1 - 10 * + 4 + c@ dup dup
               05 = swap 0f = or swap f0 and 0 > or
               if
               else
                 1 +
               then
           loop
    dup 1 > if   ."    Fehler(?): " dup .
                 ." sichtbare primaere Partitionen"                  cr
            then
        0 = if   ."    Keine sichtbare primaere Partition vorhanden" cr
            then ;

: ewtrWIN ( -- ) cr
     ."    Der Eintrag ewtrWIN heisst hier: Erweiterte Partition wurde"
cr
     ."    fuer Windows 95 eingerichtet. Wird von DOS nicht anerkannt!"
cr ;

: parttab ( -- ) cr
    05 1 do i 30 + emit ." : "
    10 0   do sectbuf 1be + i + 10 j 1 - * + c@ 0 <# # # #> type space
           loop
           sectbuf 1be + i 1 - 10 * +     c@ dup
           80 = if ."    aktiv"  then        dup
           00 = if ."  inaktiv"  then        dup 80 <> swap 00 <>  and
                if ."  ???????"  then
           sectbuf 1be + i 1 - 10 * + 4 + c@ dup 0f0 and
           10 = if ."  verstckt" then        dup 0f0 and
           00 = if ."  sichtbar" then        0f0 and dup
                                                 10 <> swap 00 <>  and
                if ."  ????????" then
           sectbuf 1be + i 1 - 10 * + 4 + c@ dup  0f and
                   0f <> swap 05 <>                 and
                if ."  primaer"  then
           sectbuf 1be + i 1 - 10 * + 4 + c@ dup
                   05 =  swap 15  =                  or
                if ."  erwtert"  then
           sectbuf 1be + i 1 - 10 * + 4 + c@ dup
                   0f =  swap 1f  =                  or
```

```
                  if ."  ewtrWIN" ewtrwin then cr
           loop ;


\ Betriebsworte zum vorliegenden Artikel
\ --------------------------------------

: hex! ( n cd -- )  \ n = in den Sektor-Puffer zu speichernde Hexzahl
                    \ c = Zeilennummer aus Part.-Tabelle   (Halbbyte)
                    \ d = Zeilen-Offset (0..3)             (Halbbyte)
    sectbuf 1be + swap 10 - + c! ;

: deaktivieren ( n -- )                              \ Unbootbar machen
    1 -
    dup 3 > if ." Wird in Teil B ergaenzt" drop exit
            then                                     \ Logisches Laufwerk ?
    dup 0 < if ." Gar kein Laufwerk?" drop exit then
    10 * sectbuf + 1be + dup dup                     \ Raus, wenn schon =00
    c@ 00 = if
    dup . ."   Schon inaktiv" drop drop exit then    \ Wenn aber war:   80 ,
    c@ 80 = if   dup 00 swap c! drop                 \ dann jetzt:      00 .
            else cr drop ." Was stimmt da nicht?"    \ Sonst:       Fehler
            then ;

: aktivieren ( n -- )                                \ Bootbar machen
    1 -
    dup 3 > if ." Wird in Teil B ergaenzt" drop exit
            then                                     \ Logisches Laufwerk ?
    dup 0 < if   cr ." Gar kein Laufwerk?" drop exit
            then
    10 * sectbuf + 1be + dup dup                     \ Raus, wenn schon =80
    c@ 80 = if cr ."   Schon aktiv" drop drop exit   \ Wenn aber war:   00 ,
            then
    c@ 00 = if   dup 80 swap c! drop                 \ dann jetzt:      80 .
            else cr drop ." Was stimmt da nicht?"    \ Sonst:       Fehler
            then ;

: verstecken ( n -- )
    1 -
    dup 3 > if ." Wird in Teil B ergaenzt"
            then                                     \ Logisches Laufwerk ?
    dup 0 < if cr ." ???" drop exit then             \ Gar kein  Laufwerk ?
    dup
    10 * sectbuf + 1be + 4 + c@ f0 and 10 =
            if cr ."   Schon versteckt" drop exit
            then
    10 * sectbuf + 1be + 4 + dup c@ 0f0 and 00 =     \ Wenn war:   0x ,
            if dup c@ 10 or swap c!                  \ dann jetzt: 1x .
            else cr drop ." Was stimmt da nicht?"    \ Sonst:      Fehler
            then ;

: sichtbar-machen ( n -- )
    1 -
    dup 3 > if ." Wird in Teil B ergaenzt"
            then                                     \ Logisches Laufwerk ?
    dup 0 < if cr ." ???" drop exit then             \ Gar kein  Laufwerk ?
    dup
    10 * sectbuf + 1be + 4 + c@ f0 and 00 =
            if cr ."   Schon sichtbar" drop exit
            then
```

```
180       10 * sectbuf + 1be + 4 + dup c@ 0f0 and 10 =    \ Wenn war:   1x ,
181               if dup c@ 0f and swap c!                \ dann jetzt: 0x .
182               else cr drop ." Was stimmt da nicht?"   \ Sonst:      Fehler
183               then ;
184
185   : erweitern-fuer-DOS ( n -- )
186       \ Abgesichert durch: ???. Muss 0x sein, nicht z.B. 83 (= Linux).
187       1 -
188       dup 3 > if ." Wird in Teil B ergaenzt"
189               then                                    \ Logisches Laufwerk ?
190       dup 0 < if ." ist gar kein Laufwerk?" drop exit then
191       10 * sectbuf + 1be + 4 + dup c@
192       dup 0f and 05 = if
193       ." ist schon fuer DOS eingerichtet" drop swap drop exit then
194       dup 0f and 0f = if 05 swap c!
195       ." ist jetzt erweitert fuer DOS" cr exit
196       else swap drop . ." ist wohl keine erweiterte Partition?" then
197   ;
198
199   : erweitern-fuer-WIN ( n -- )
200       \ Abgesichert durch: ???. Muss 0x sein, nicht z.B. 83 (= Linux).
201       \ Wenn mehrere erw. Partn, dann fuer alle: Wenn (1be+4) = 05, dann: 0f
202       1 -
203       dup 3 > if ." Wird in Teil B ergaenzt"
204               then                                    \ Logisches Laufwerk ?
205       dup 0 < if cr ." ???" drop exit then            \ Gar kein  Laufwerk ?
206       10 * sectbuf + 1be + 4 + dup c@ 0f0 and 00 =    \ Wenn war:   0x ,
207                   if dup c@ 10 or swap c!             \ dann jetzt: 1x .
208                   else drop cr ." ??? "               \ Sonst:      Fehler
209                   then ;
210
211   : show-hide&act ( -- )    \ Nur im Puffer - noch nicht auf der Festplatte !
212           parttab errext? erract? errpri?                                  cr
213       ." 80 = aktiv (Hoechst. eine prim. Partition sollte aktiv sein!)"    cr
214       ." 00 = nicht aktiv (heisst: steht dann nicht zum Booten bereit)"    cr
215       ." 05 = erweiterte Partition. (Es sollte nur eine solche geben!)"    cr
216       ." 0F = erweitert fuer Windows (fuer DOS dann nicht erreichbar!)"    cr cr
217       ." Auch logische Laufwerke koennen aktiv oder/und versteckt sein."
218       ."  (Wird in Teil "                                                  cr
219       ." B ergaenzt.)   Ob das im Einzelnen sinnvoll ist ,  "
220       ." erforsche man bei Google."                                        cr
221       ." Aenderungen zunaechst nur im Puffer. Uebernahme auf die "
222       ." Festplatte: anwenden"                                             cr
223       ." Zulaessige Eingaben (n = Nummer wie in der Partitions-Tabelle "
224       ." oben):"                                                           cr
225       ." n verstecken       n sichtbar-machen       n aktivieren       "
226       ." n deaktivieren"                                                   cr
227       ." n erweitern-fuer-dos   n erweitern-fuer-win    getmbr   "
228       ." show-hide&act   hex!"                                             cr
229       ." hex! dient der Eingabe von Hexzahlen in den Puffer nach dem "
230       ." Schema x cd hex! "                                                cr
231       ." Dabei ist x = Hexzahl,  c = Zeilennummer,  "
232       ." d = Offset (beginnend ab 0 bis 3)"                                cr ;
233
234   : anwenden ( -- )    \ Momentanen Puffer auf die Boot-Festplatte schreiben
235       ( putmbr ) ;
236
237   \ Zur Sicherheit: Vor dem Anwenden von "anwenden" Klammern beseitigen!
238   \ MBR auf Diskette, Stick oder HD speichern: Siehe VD-Artikel 01/2009!
```

Triceps 2.0

Bernd Paysan

Friedel Amend *war fleißig, und hat einen neuen Triceps gebaut — dieses Mal nicht mit Schrittmotoren, sondern mit Servos. Ziel war es, einen neuen Hingucker für den LinuxTag zu bauen. Für den diesjährigen LinuxTag haben wir nur das bekannte Triceps–Programm laufen lassen, denn die Entwicklung mit Kinect–Kamera und Mühle oder Go als Spiel gegen den Zuschauer wird etwas länger dauern. Dieser Artikel beschreibt, wie die etwas kompliziertere Trigonometrie mit dem b16 gelöst wird.*

Aufbau des Roboters

Der Roboter besteht aus drei Türmen, die im 120°–Winkel gedreht um das Spielfeld stehen. Die Servos lenken einen 20 cm–Arm aus. An den drei Seilen hängt ein Permanentmagnet, dessen Magnetfeld durch einen Abstandshalter soweit reduziert wird, dass man die Kugel durch schnelles Reißen am Magnet ablegen kann. Bei langsamer Bewegung bleibt die Kugel am Magnet hängen. Zumindest meistens.

Friedel hat zunächst einen Roboter gebaut, bei dem die Türme fest mit dem Spielfeld verschraubt sind. Da hier jedoch erst mal experimentiert werden musste, sind nach kurzer Diskussion freistehende Türme herausgekommen, bei denen der Servo–Motor in der Höhe verstellbar montiert werden kann. Es hat sich als zweckmäßig herausgestellt, die Servos an der unterst möglichsten Stellung zu montieren.

Die Maßeinheit für Längen im Programm ist mm, die für Zeit 1ms. Das Programm ist bezüglich der Längenmaße einheitenfrei, d.h. man kann sich eine beliebige Einheit wählen, muss aber überall dieselbe Einheit verwenden — das Ergebnis sind ja Winkel, und Winkel sind dimensionslos. Als Basis habe ich das Programm von Ewald Rieger für den ersten Triceps verwendet.

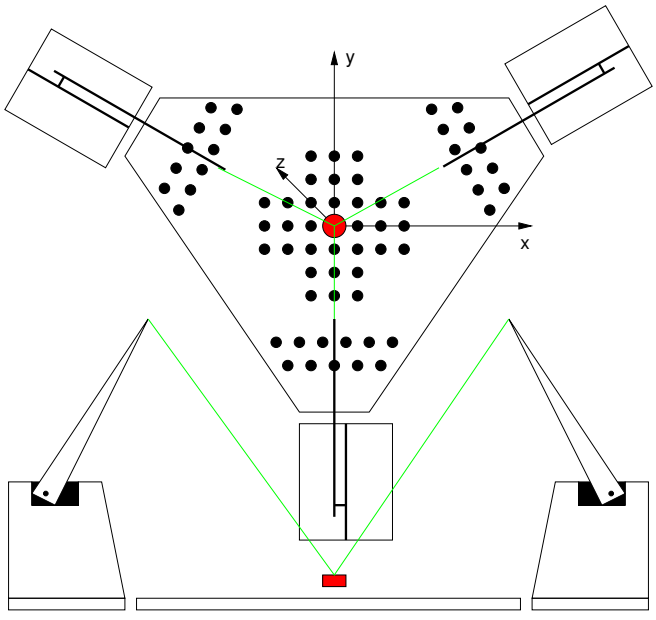


Abbildung 1: Triceps2 schematische Zeichnung (Aufsicht+Seitenansicht)

Trigonometrie

Beim Koordinatensystem habe ich mich dafür entschieden, den Ursprung in die Mitte des Spielfelds zu legen; dieses Konstrukt hat sich schon beim ersten Triceps bewährt. Die Rotation um 120°, damit man für alle drei Türme jeweils ein gleichwertiges Koordinatensystem hat, ist noch recht einfach:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{1}{2}\begin{pmatrix} -1 & -\sqrt{3} \\ \sqrt{3} & -1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}$$

Für die Drehung um −120° muss man nur das Vorzeichen an der $\sqrt{3}$ tauschen. Der b16–Code dafür sieht entsprechend einfach aus:

```
$DDB3 Constant sqrt3/2

: >xy  ( x y -- x' y' )
  >r 2/ negate sqrt3/2 # r> usmul nip ;
: xy-calc
    x # @ y # @
    2dup >xy + xl # !
    2dup >xy - xr # !  swap
    2dup >xy - yl # !
         >xy + yr # ! ;
```

Ich habe eine Sonderform der Multiplikation eingeführt, nämlich eine, mit der ein vorzeichenloser mit einem vorzeichenbehafteten Wert multipliziert wird. Die Skalierung für vorzeichenlose Werte ist 0..FFFF von 0 bis 1, für vorzeichenbehaftete dann −$8000..$7FFF von −1 bis 1. Analog gibt es dann auch eine Division, die einen vorzeichenbehafteten 32–Bit–Wert durch einen vorzeichenlosen 16–Bit–Wert teilt. Das hat sich bewährt, und ist auch einfacher zu implementieren.

Ziel der ganzen Rechnerei soll es sein, von der Position des Magneten auf die einzustellenden Winkel der Servos zu schließen. Dazu folgende Überlegung: Der vom Servo gesteuerte Arm hat den Freiheitsgrad eines Kreises, mit der Armlänge als Radius. Die Seile, an der der Magnet aufgehängt ist, haben den Freiheitsgrad einer Kugel — mit der Seillänge als Radius. Schneiden wir den Kreis mit der Kugel, so bekommen wir den Punkt, auf den wir den Arm hinbewegen wollen.

Der Schnitt einer Kugel mit einem Kreis reduziert sich auf das Schneiden zweier Kreise in der Ebene, wenn wir zunächst die Kugel mit der Ebene schneiden, in der der Kreis liegt (das ist dann auch die Ebene, die entlang der

y, z–Koordinaten durch den Ursprungsort unseres Koordinatensystems geht — wie praktisch). Die x–Koordinate steht also schon gleich senkrecht auf dieser Ebene, wodurch wir es mit einem einfachen rechtwinkligen Dreieck zu tun haben, dessen Hypotenuse die Seillänge ist, und dessen eine Kathete die x–Koordinate des Magneten ist. Also ein einfaches $b = \sqrt{c^2 - a^2}$:

```
macro: faden^2 faden # @ dup mul end-macro
: >sc>  ( addr -- )  @+ >r >r faden^2
  r> abs dup mul d- sqrt r> ! ;
: >sc     x  # >sc>  xl # >sc>  xr # >sc> ;
```

Ich habe das Speicherlayout für die drei Servos so gewählt, dass die berechneten Werte unmittelbar hinter ihren Ausgangswerten sind, wo das möglich ist. Damit kann man einfach die Auto–Increment–Funktionen verwenden.

Das Dreieck, das unseren Winkel vorgibt, besteht aus drei Seiten: Der Arm selbst (dessen Länge bekannt und konstant ist), die soeben berechnete Abbildung des Fadens auf diese Ebene, und als dritte Seite fehlt noch die Abbildung von der Servo–Achse zum Magneten auf die y, z–Ebene. Auch hier haben wir es mit einem rechtwinkligen Dreieck zu tun, die Höhe z bestimmt die eine Kathete (um den Offset der Turmhöhe verschoben), die y–Koordinate um die Distanz zum Turm erhöht, die andere. Also auch ein einfaches $c = \sqrt{a^2 + b^2}$:

```
: >c>   ( addr -- )  @+ >r distance # @ +
  dup mul height # @ z # @ -
  abs dup mul d+ sqrt r> ! ;
: >c      y  # >c>   yl # >c>   yr # >c> ;
```

Damit haben wir alle drei Seiten beisammen, und müssen nur noch den Cosinussatz anwenden:

$$\cos\beta = \frac{a^2 + c^2 - b^2}{2ac}$$

Die rechte Seite dieser Gleichung ist nicht schwierig zu berechnen:

```
: >cos> ( c-addr b-addr -- ) @+ >r >r
  @ u2/ dup dup mul drop  \ a^2
  arm # @ 2/ dup mul drop \ c^2
  r> 2/ dup mul drop - +  \ b^2
  over >r >r drop 0 # r> 2/ r>
  arm # @ mul drop sdiv drop r> ! ;
: >cos ( -- )
  c  # sc  # >cos>
  cl # scl # >cos>
  cr # scr # >cos> ;
```

Um den Winkel β zu bekommen, muss man „nur noch“ den Arcus Cosinus berechnen... wobei natürlich der b16 keine Trigonometriefunktionen kennt, und wir auch nicht den Aufwand spendieren, das wirklich genau ausrechnen zu wollen. Eine tabellengetriebene Lösung muss reichen, mit 64 Werten, zwischen denen linear interpoliert wird. Wir addieren $8000 auf den vorzeichenbehafteten Wert, denn die 0 soll in der Mitte der Tabelle liegen, und multiplizieren diesen Wert dann mit 64. Der Vorkommateil ist der Index in die Tabelle, wir holen zwei aufeinanderfolgende Werte heraus. Der Nachkommateil liegt noch auf dem Stack — für den ersten Wert (mit dem kleineren Index) nehmen wir das Komplement, was ungefähr $1 - x$ entspricht:

```
$40 Constant tablesize#
also forth
: gentable -1e tablesize# 1+ 0 DO
        fdup facos pi f/ $FFFF s>f f* f>s
        [ previous ] , [ also forth ]
        1e 32e f/ f+
    LOOP  fdrop ;
previous
| acostable  gentable
: acos ( cos -- alpha )
  $8000 # + tablesize# # mul
  cells acostable # + @+ @
  >r over r> mul >r drop
  >r com r> mul r> + nip ;
: >alpha> ( addr -- ) @+ >r acos r> ! ;
: >alpha ( -- )
  cos  # >alpha>
  cosl # >alpha>
  cosr # >alpha> ;
```

Damit können wir nun die benötigten Winkel berechnen, sowohl den zwischen Arm und Magnet, als auch den zwischen Magnet und Turm — beide zusammen ergeben dann (mit einem Bias von 90°) die Einstellung des Servo-Motors.

```
: >angle1 ( y-addr -- angle )  @ >r
  0 # height # @ z # @ -
  r> 2* sdiv drop acos ;
: >angle2 ( angle alpha-addr -- )
  @+ >r + $8000 # + r> ! ;
: >angle
  c  # >angle1 dup b  # !
       alpha  # >angle2
  cl # >angle1 dup bl # !
       alphal # >angle2
  cr # >angle1 dup br # !
       alphar # >angle2 ;
```

Das alles hintereinander ausgeführt ergibt die Koordinaten–Rechnung

```
: coord-calc ( -- )
  xy-calc >sc >c >cos >alpha >angle ;
```

Motorsteuerung

Ewald Rieger hat mir ursprünglich auch die Aufgabe zugedacht, ein PWM–Modul für den b16 zu schreiben, mit dem man dann ohne weitere Software die PWM–Ausgabe machen kann. Nun, dazu war ich zu faul, Software entwickeln ist nämlich einfacher als Hardware. Die Servos erwarten ein neues Signal alle 20ms, also nicht sonderlich oft. Die Pulse selbst sind im Mittel 1,5ms lang (senkrechte Position), und variieren um ±900µs. Da macht es auch nichts aus, wenn die 20ms nicht exakt eingehalten werden — die Ausführung war also, einfach

einen Puls nach dem anderen über die ohnehin vorhandenen GPIOs an die Servos zu geben, und den Timer zum Warten zu nutzen.

Ich muss hier noch etwas über den Timer im b16 einschieben, weil der für den einen oder anderen vielleicht ungewohnt ist: Der b16 hat als Peripherie einen freilaufenden 32–Bit–Timer, der jeden Takt um 1 erhöht wird. Dieser Timer kann nicht gesetzt werden, er läuft einfach weiter. In ein zweites Register kann man eine Alarm–Zeit einstellen, die den b16 aufweckt, wenn der Uhrzeiger vom Alarmzeiger aus gesehen links ist (das sind zyklische Zahlen, die miteinander verglichen werden). Deltas kann man immer als Differenz errechnen, absolute Werte sind es, die ein genaues Zeitmanagement ermöglichen. Bei 50MHz läuft der Timer in 85,9 s einmal im Kreis, d.h. man kann Zeiten bis knapp 43 s direkt verwenden — und zwar auf 20 ns genau.

Wir definieren uns also ein paar Wörter zum Warten, die jeweils die letzte Alarmzeit als Basis verwenden (und damit unabhängig davon sind, wie lange die Berechnung dauert):

```
: after ( ms -- dtime )
  #50000 # mul
: tick-after ( ticks -- dtime )
  TVAL1 # @  TVAL0 # @ d+ ;
macro: >irq  0 # IRQACT # c!*
       drop end-macro
: till ( dtime -- )
  TVAL0 # !+ !  >irq ;
```

Mit diesen Worten wartet es sich bequem, wobei die Eingabe entweder Millisekunden oder Takte ist. b16–Definitionen sind Labels, endet eine Definition nicht mit ;, so läuft der Code einfach weiter. Nun müssen wir noch den Ausschlag der Servos in eine Zeit umwandeln:

```
| motor-min#  #18300 ,
| motor-gain# #45000 ,
: ausschlag ( 0-ffff -- dtime )
  motor-gain# # @ mul nip
  motor-min#  # @ +
  dup + 0 # dup +c  tick-after ;
```

Eigentlich hätte das motor–min# nach Spec 15000 sein sollen, aber das war nicht so. Die eigentliche Kontrolle über den Motor ist nun ziemlich trivial, wir müssen lediglich die Ports der Reihe nach an- und ausschalten, und die berechneten Zeiten abwarten:

```
: -motor ( -- ) $0000 # port # ! ;
: motor1 ( -- ) $0001 # port # ! ;
: motor2 ( -- ) $0010 # port # ! ;
: motor3 ( -- ) $0100 # port # ! ;
: do-motor ( -- )
  motor1 pos1 # @ offset1 # @ +
         ausschlag till -motor
  motor2 pos2 # @ offset2 # @ +
         ausschlag till -motor
  motor3 pos3 # @ offset3 # @ +
         ausschlag till -motor ;
```

Diese Motoransteuerung bauen wir nun zusammen mit der Koordinaten–Berechnung in eine Warte–Schleife, die das alles mit 50 Hz ausführt.

```
: motor-step ( -- )
  coord-calc  12 # after  till
  8 # after  do-motor  till ;
macro: LOOP  -1 # + dup -UNTIL
       drop  end-macro
: wait ( n -- ) \ 50Hz wait
  BEGIN  motor-step  LOOP ;
```

Bewegungssteuerung

Die Servos sind viel zu schnell, um den Ball direkt von einer zur nächsten Position zu ziehen. Wir müssen also Zwischenschritte einfügen, und Rampen zur Beschleunigung und zum Abbremsen fahren. Der erste Triceps hatte es beim Rampen fahren noch einfach — die Taktung der Schrittmotoren wurde variiert. Das können wir bei den Servos nicht machen, die bekommen ihren konstanten 50 Hz–Takt.

Wir berechnen die jeweils nächste Position also, indem wir mehrere kleine Schrittchen in eine 50 Hz–Takt machen, je schneller die Bewegung, desto mehr. Dazu zählen wir eine Variable SPEED hoch, und die Distanz DIST zum Ziel herunter — das Minimum aus beidem und der Maximalgeschwindigkeit ist die aktuelle Geschwindigkeit.

```
24 Constant speedlimit#
: movesteps ( -- )
  speed # @ dist # @ umin  u2/ u2/
  speedlimit# # umin u2/ u2/ 1 # +
  BEGIN  movestep  1 # speed # +!
         dist # @ -IF  drop ;  THEN
  LOOP ;
```

Die Distanz wird in MOVESTEP dekrementiert, hier ändern wir auch jede der Koordinaten um einen Delta–Wert, der natürlich nur als Nachkommateil im Bereich $[-0.5..0.5)$ gespeichert wird.

```
: movestep ( -- )
  stepx # @+ @. >r >r dup 0<
          r> x # @ d+ x # ! r> !
  stepy # @+ @. >r >r dup 0<
          r> y # @ d+ y # ! r> !
  stepz # @+ @. >r >r dup 0<
          r> z # @ d+ z # ! r> !
  dist # @. >r -1 # +
  dup 0<IF  drop 0 #  THEN  r> ! ;
```

Um Ewalds High–Level unverändert lassen zu können, bewegen wir die Kugel auch nur entweder senkrecht in z–Richtung, oder waagrecht in x, y–Richtung. Senkrecht ist am einfachsten zu berechnen, einfach die Differenz zwischen alten Wert und neuen als Distanz setzen (der Faktor 8 passt die Geschwindigkeit an), und die Schrittweite ist eigentlich dann eine Konstante — wir machen die Division hier trotzdem, weil das Programm dann wartbarer ist.

```
: >movez ( z -- )
  z # @ - dup deltaz # !
  abs 2* 2* 2* dist # !
  0 # dup stepx # !+ !
  0 # dup stepy # !+ !
  0 # dup deltaz # @ dist # @
      sdiv drop stepz # !+ !
  0 # speed # ! ;
```

Bei Bewegungen in der Waagerechten müssen wir die Hypotenuse eines Dreiecks ausrechnen — eine einfache Übung im Vergleich zu vorher.

```
: >moveto ( x y -- )
  y # @ - deltay # !  x # @ - deltax # !
  deltax # @ abs 2* dup mul
  deltay # @ abs 2* dup mul d+
  sqrt 2* dup -IF  drop 1 #  THEN
  dist # ! \ avoid zero delta
  0 # dup deltax # @ dist # @
      sdiv drop stepx # !+ !
  0 # dup deltay # @ dist # @
      sdiv drop stepy # !+ !
  0 # dup stepz # !+ !
  0 # speed # ! ;
```

Damit sind wir fast am Ende unserer Low–Level–Wörter angelangt. Bleibt nur noch die Schleife, um eine Bewegung bis zum Ende durchzuführen (also bis DIST 0 ist), und diese zu nutzen, um die Bewegungen zu (mit absoluten Werten) angegebenen Positionen durchzuführen. Die Variable DESTINATION wird nur für Debugging–Zwecke benutzt.

```
: >pos ( -- )
  BEGIN  movesteps motor-step
         dist # @ -UNTIL ;
: moveto ( x y -- )
  2dup destination !+ !  >moveto  >pos ;
: movez ( z -- )  >movez  >pos ;
```

Spiel–Steuerung

So, jetzt haben wir alles, was zur Ansteuerung unseres Magneten benötigt wird. Der Rest ist die Ansteuerung des Spiels selbst. Da es ein Pick&Place–Roboter ist, brauchen wir zunächst einmal PICK und PLACE:

```
: down     #15 # movez ;
: lift     #40 # movez ;
: release  #40 # z # ! 10 # wait ;
: pick   down lift ;
: place  #20 # wait  down release ;
```

Unser Magnet hat keinen Schalter. Also, aufnehmen ist ganz einfach: Langsam nach unten und dann langsam wieder nach oben. Aber wie setzen wir ab? Erst mal auspendeln lassen (0.4s reichen), dann langsam nach unten, und mit einem Ruck den Magnet wegziehen. Danach noch etwas auspendeln lassen. Gelegentlich macht der Magnet dabei ein Looping, es wäre besser, nur einen kleinen Ruck zu machen, und dann langsam abzubremsen — aber dieser Code hier lief auf dem LinuxTag.

Als Nächstes kommt unser Spielfeld. Hier müssen wir nur die Koordinaten, die von 0 0 in der linken unteren Ecke des Spielfelds bis 6 6 in der rechten oberen Ecke reichen (beides Positionen, auf denen kein Loch gebohrt ist).

```
: spiel-feld ( x-nr y-nr -- )
    #20 # mul drop #-60 # + >r
    #20 # mul drop #-60 # + r>  moveto ;
```

Ähnlich berechnen wir die Ablagepositionen. Es gibt 11 Löcher pro Reihe, im Zickzack angeordnet:

```
: reihe ( nr -- x y )
    dup >r #10 # mul drop #-50 # +
    #-115 # r> 1 # and IF  #-20 # +  THEN ;
```

Für die drei Reihen müssen wir da nur noch etwas die Koordinaten transformieren:

```
: reihe1 ( nr -- )
    reihe  #20 # + moveto ;
: reihe2 ( nr -- )
    reihe  >r #15  # + r> over >r
    dup r>  >xy + >r >xy - r> moveto ;
: reihe3 ( nr -- )
    reihe  >r #-15 # + r> over >r
    dup r>  >xy - >r >xy + r> moveto ;
```

Und zur Berechnung, welche Reihe es jetzt sein soll, eine kleine Tabelle und eine Division durch 11:

```
| reihen ' reihe1 , ' reihe2 , ' reihe3 ,
: ablage ( nr -- )
  0 # #11 # div swap cells reihen # + @ goto ;
```

Wenn wir abgelegt haben, zeigen wir den Spielstand auf den 7–Segment–LEDs an, dezimal natürlich — also müssen wir das in eine BCD–Zahl wandeln, denn die 4 Ziffern sind natürlich Hex–Ziffern.

```
: .stand   #33 # freiablage # @ - 0 #
  #10 # div swap 2* 2* 2* 2* + LED7 # ! ;
```

Jetzt noch ein paar Wörter zum Kugeln wegnehmen und ablegen, entfernen holen und einräumen, wobei wir den Spielstand anzeigen, wenn sich der geändert hat.

```
macro: kugel-wegnehmen ( n m -- )
       spiel-feld pick end-macro
macro: kugel-ablegen   ( n m -- )
       spiel-feld place end-macro
: kugel-entfernen
  freiablage # @ ablage place
  1 # freiablage # +!  .stand ;
: kugel-holen
  -1 # freiablage # +!
  freiablage # @ ablage pick  ;
: einraeumen
  kugel-holen kugel-ablegen  .stand ;
```

Die nun folgenden vier Spielzüge sind zwar alle sehr ähnlich, aber Ewald Rieger hat auf ein komplexeres Factoring verzichtet, und so alle vier Richtungen auscodiert:

```
: spiele1 ( n m -- )
    2dup          kugel-wegnehmen
    2dup >r 2- r> kugel-ablegen
         >r 1- r> kugel-wegnehmen
    kugel-entfernen ;
: spiele2 ( n m -- )
    2dup          kugel-wegnehmen
    2dup    2+    kugel-ablegen
            1+    kugel-wegnehmen
    kugel-entfernen ;
: spiele3 ( n m -- )
    2dup          kugel-wegnehmen
    2dup >r 2+ r> kugel-ablegen
         >r 1+ r> kugel-wegnehmen
    kugel-entfernen ;
: spiele4 ( n m -- )
    2dup          kugel-wegnehmen
    2dup    2-    kugel-ablegen
            1-    kugel-wegnehmen
    kugel-entfernen ;
```

Zum Einräumen noch ein paar kleine Faktoren:

```
: 432einraeumen  ( n -- )
    dup 4 # einraeumen
    dup 3 # einraeumen
        2 # einraeumen ;
: 654einraeumen ( n -- n )
    dup 6 # einraeumen
    dup 5 # einraeumen
    dup 4 # einraeumen ;
: 210einraeumen ( n -- )
    dup 2 # einraeumen
    dup 1 # einraeumen
        0 # einraeumen ;
```

Damit sind wir fast fertig mit der Vorbereitung zum Hauptprogramm.

Da die Türme nicht fest am Spielfeld befestigt sind, und auch mal verrutschen können, gibt es am Anfang des Spiels eine Kalibrierfunktion, die wichtige Eckpunkte abfährt und genügend Zeit lässt, die Anordnung zu optimieren. Mit einem Druck auf den Resetknopf kann man die Kalibrierung jederzeit neu anwerfen. Diese Funktion testet auch, ob wir die Koordinaten richtig transformiert haben.

```
: calibrate
     0 # reihe1          250 # wait
    10 # reihe1          250 # wait
     0 # reihe2          250 # wait
    10 # reihe2          250 # wait
     0 # reihe3          250 # wait
    10 # reihe3          250 # wait
    3 # 3 # spiel-feld   250 # wait
    down                 250 # wait
    6 # 3 # spiel-feld   250 # wait
    3 # 6 # spiel-feld   250 # wait
    3 # 0 # spiel-feld   250 # wait
    lift                 250 # wait ;
```

Das Spiel

Das Spiel packen wir in die Haupt–Routine, die vom Reset–Vektor angesprungen wird. Klar, die muss erst ein bisschen initialisieren, aber dann kann's schon los gehen.

```
: boot
    $00 # LED7 # !
    0 # dup dup #40 # z #  !+ !+ !+ !
    0 # deltaz # !
    init-port
    calibrate
    BEGIN
        &33 # freiablage # !
                           0 # 432einraeumen
                           1 # 432einraeumen
    2 # 654einraeumen  dup 3 # einraeumen  210einraeumen
    3 # 654einraeumen                      210einraeumen
    4 # 654einraeumen  dup 3 # einraeumen  210einraeumen
                           5 # 432einraeumen
                           6 # 432einraeumen

        3 # 1 # spiele2
        1 # 2 # spiele3
        4 # 2 # spiele1
        6 # 2 # spiele1
        1 # 4 # spiele4
        3 # 4 # spiele1
        1 # 2 # spiele3
        3 # 2 # spiele3
        5 # 4 # spiele1
        2 # 0 # spiele2
        2 # 3 # spiele4
        6 # 4 # spiele4
        4 # 0 # spiele1
        4 # 6 # spiele4
        2 # 6 # spiele3
        4 # 3 # spiele2
        2 # 0 # spiele2
        0 # 4 # spiele3
        3 # 4 # spiele1
        6 # 2 # spiele1
        4 # 1 # spiele2
        0 # 2 # spiele2
        0 # 4 # spiele3
        4 # 6 # spiele4
        2 # 5 # spiele4
        4 # 3 # spiele2
        2 # 2 # spiele2
        4 # 5 # spiele1
        2 # 5 # spiele4
        2 # 3 # spiele3
        5 # 3 # spiele1
        3 # 3 # kugel-wegnehmen kugel-entfernen
        3 # 3 # spiel-feld
        #1000 # wait
    AGAIN ;
$3FFE org \ der Reset-Vector
     boot ;;
```

Das war's schon. Friedel Amend ist am LinuxTag vorbeigekommen, mit dem neusten Spielfeld, das schön grün ist (für die Kinect–Kamera), und 9 mal 9 Felder hat: Hier kann man alle angedachten Spiele programmieren, und nur für Go braucht man eine Ablage außerhalb des Spielfelds. Das aber ist dann Triceps 2.1, und eine andere Geschichte.

Jahr des Drachen

Bernd Paysan

Das letzte Stück aus Heinz Schnitters *Drachen–Shirt–Auflage ist vorletztes Jahr am Linuxtag in meinen Besitz gegangen, es gibt keine weiteren mehr. Deshalb, hat das LinuxTag–Team u.a. auch beschlossen, müssen neue T–Shirts her. Das alte Motiv ist, wie immer nach so langer Zeit, natürlich verschollen, und selbst wenn man es auftreiben würde, ob man das alte Corel–Draw–Format noch lesen kann? Und — wollen wir den alten Drachen überhaupt noch?*

Die Ausführung

Also habe ich mich bereiterklärt, einen neuen T–Shirt–Drachen zu malen. Als Vorlage habe ich ein chinesisches T–Shirt genommen, das ich letztes Jahr in Shanghai gekauft habe — und dessen Größe „L“ irgendwie anders gemeint war, als ich das dachte. Zumindest nach der ersten Wäsche... aber da wir das Jahr des Drachen schreiben, lag es nahe, einen chinesischen zu nehmen.

Chinesische Drachen (Long) sind Fabelwesen der besonderen Art [1], sie ähneln vom Designprinzip mehr dem Wolpertinger aus Bayern — man nehme verschiedene Tiere, und kombiniere die zu etwas, was es unmöglich geben kann. Jedenfalls hat ein Drache einen Wasserbüffelkopf, aber mit Barteln, Geweih und Bart und dem Gebiss eines Löwen. Den Körper einer Schlange mit den Schuppen eines Karpfens, aber trotzdem vier Beine mit den Füßen eines Adlers.

Anders als bei uns sind in China Drachen männlich, und mit Phönixen (Feng) verheiratet (mit denen sie neun völlig unterschiedliche Kinder haben). Und sie sind nicht Geschöpfe des Feuers, sondern des Wassers — auch wenn sie in der Luft sind (dort sind sie für den Regen zuständig). Der Drache mit zwei Köpfen ist in China nicht bekannt, aber das macht natürlich nichts.

Gezeichnet wird der Drache recht martialisch, und der Körper mit schönem Schwung. Eigentlich ist er nicht blau, sondern golden oder feuerrot. Aber der Drache auf dem T–Shirt aus Shanghai war schon blau, und hat mir mit der Farbe ganz gut gefallen. Als Symbol der Macht erkennt man den Rang eines Drachen an der Anzahl der Klauen — fünf pro Fuß sind kaiserliche Drachen, nur für den Kaiser selbst und seine höchsten Beamten gestattet. Vier und drei dann für entsprechend niedrigere Ränge. Natürlich hat unser Drache fünf Klauen.

Gezeichnet habe ich den Drachen mit Inkscape und einem kleinen Wacom–Tablet. Und gedruckt, nachdem der Berliner T–Shirt–Drucker, den Carsten ausgesucht hat, etwas von „direct on garment“ auf seiner Web–Site hat, in Wolfratshausen mit eben einer solchen DoG–Maschine. Das ist ein Tintenklecksdrucker, der mit Textilfarben und einem zusätzlichen Deckweiß druckt. Diese Technologie ist sehr präzise, erlaubt auch den Druck auf dunklen Stoffen, und hält das Waschen aus.

Als Schrift hat sich das Team spontan für den Morsecode entschieden, man kann jetzt also nicht ganz so einfach lesen, was für eine Botschaft das Shirt trägt.

Das Ergebnis

Die Textildruckerin meinte, der Drache sei viel zu klein, der müsse das ganze T–Shirt für sich einnehmen, welches dann nicht in die Hose gesteckt werden darf. Ich glaube, der Drache ist genau richtig, denn ohne Rahmen wirkt ein Bild nur halb so gut (chinesische Kunst setzt ziemlich viel leere Fläche ein).

Die frisch gedruckten T–Shirts habe ich am Vortag vor der Fahrt nach Berlin gleich fotografiert, und an das Team verschickt. Das verwendete T–Shirt hat eine Brusttasche, in die der Nerd dann seine Stifte 'reintun kann (und — weil ein Sponsor abgesprungen ist — auch für das Anheften des Name–Tags).

Abbildung 1: Drachenshirt weiß

Abbildung 2: Drachenshirt schwarz

Die T–Shirts waren dann die „coolsten T–Shirts“ auf dem LinuxTag, behauptet zumindest Erich Wälde, am Belug–Stand erfahren zu haben ;-). Ich würde sagen, die anderen haben sich einfach nicht genug Mühe gegeben. Leo Brodie meinte zu dieser Drachen–Variante „nice“ (Facebook–Kommentar).

Auch haben wollen?

Die von mir gedruckten Shirts sind alle weggegangen. Da aber Carsten Strotmann beim ursprünglich angedachten T–Shirt–Drucker in Berlin mit etwas älterem Transferverfahren auch 10 Shirts bestellt hat, von denen nach Ende des LinuxTags mit Bestellungen noch 8 übrig waren, kann der Ungeduldige dort sein Shirt recht schnell bekommen. Weitere Anfragen gehen dann wohl am besten an mich. Die Shirts plus Druck kosten brutto ca. 20 Euro (ohne Versand), Genaues kann ich erst sagen, wenn ich den Auftrag erteile, aber die Daten sind jetzt im System drin, was die Sache ein klein wenig billiger macht als beim ersten Druck.

Der Textildrucker hat einen umfangreichen Katalog, es kann also alles Mögliche bedruckt werden: `http://www.textil-online.com/extra-textildruck/`, vom Handtuch bis zur Laptoptasche ist da so ziemlich alles dabei, auch wenn ich jetzt beim Handtuch etwas skeptisch wäre, ob das Ergebnis dann so aussieht, wie gedacht.

Literatur

[1] Der chinesische Drache ist in der deutschen Wikipedia noch nicht gelöscht worden: `http://de.wikipedia.org/wiki/Long_(Mythologie)`

Recognizer

Bernd Paysan

Forth ist ein erweiterbares System, heißt es, denn man kann den Compiler beliebig erweitern — vorausgesetzt, man ist damit zufrieden, dass das Parsing alles zwischen zwei Spaces als Wort interpretiert, und man genau solche Wörter definieren kann. Das endet schon mit Literals, die müssen eingebaut sein. Je nach System hat man evtl. ein `notfound`*, in das man eine eigene Routine einhängen kann, aber das ist alles auch nicht so wohldefiniert.*

Motivation

2007, während der Diskussion zum RfD für Zahlenpräfixe, machte Anton Ertl den Vorschlag [1], ein Recognizer–System zu implementieren, mit dem man den Forth–Interpreter dynamisch umkonfigurieren kann. Der Vorschlag geht über reine Zahlenumwandlung natürlich weit hinaus, und war zunächst nur grob angerissen.

Matthias Trute hat den Vorschlag aufgegriffen und nach einigen Diskussionen im IRC auch in amForth eingebaut, und vor einem Jahr einen Artikel in der VD geschrieben [2]. Win32Forth hat auch einen Recognizer bekommen, und seit neuestem ist einer in Gforth. Natürlich sehen die Recognizer, die ja zunächst noch experimentell sind, erst mal alle etwas anders aus. Die in amForth sind state–smart, haben also eine Abfrage, in welchem State sie sind. In Win32Forth hängt man zwei verschiedene Recognizer jeweils in die Liste für den Compiler und Interpreter. Und in Gforth, ja das kommt im nächsten Kapitel.

Recognizer in Gforth

Die Recognizer in Gforth folgen lose einem OO–Design–Pattern, der Factory. Es wird zwar kein richtiges Objekt

erzeugt, aber das, was auf dem Stack liegt, kommt einem Objekt sehr nahe — der Recognizer hinterlässt eine Methodentabelle und zusätzliche Daten, die wir hier „Token“ nennen. Hat der Recognizer kein Glück mit dem ihm übergebenen String, hinterlässt er den String selbst, und eine `r:fail` genannte Methodentabelle. Der Recognizer selbst macht nur das Tokenisieren, den Rest übernehmen die drei Methoden, die im Moment nur Felder in einer Struktur sind. Das Parsen macht PARSE–NAME, also das Suchen nach Blanks. Der Recognizer bekommt also einen String auf dem Stack. So sieht der Stack–Effekt eines Recognizers aus:

***x*–RECOGNIZER** (addr u — token r:*x* | addr u r:fail)

Die drei Methoden haben natürlich auch einen Stack–Effekt, und der Einfachheit halber nennen wir sie mal INT, COMP und LIT (da der Gforth–Kernel noch kein OOP–System hat, gibt es diese Namen nicht wirklich, sondern Varianten mit R> vorne dran, die einen Index in die Tabelle darstellen).

INT (x*i token — y*j) Führt die Interpretations–Semantik des Tokens aus (die den Stack–Effekt (x*i — y*j) hat)

COMP (token —) Führt die Compilations–Semantik des Tokens aus

LIT (token —) Fügt folgende Run–Time–Semantics an das aktuell definierte Wort hinzu

LIT runtime (– token) Legt das Token wieder auf den Stack

RECOGNIZER: (xt-int xt-comp xt-lit „name“ —) Legt eine Recognizer–Tabelle an

Was ein Token genau ist, und wie viele Stack–Elemente es verbraucht, ist jedem Recognizer selbst überlassen. Das Token ist einfach das, was an Informationen aus dem String extrahiert wurde, die Verarbeitung geschieht ja mit den zurückgelieferten Methoden.

Die Recognizer sind in Stacks organisiert, die man ähnlich wie den Vokabulary–Stack mit

GET–RECOGNIZERS (rec–addr — rec_n .. rec_1 n) holen und mit

SET–RECOGNIZER (rec_n .. rec_1 n rec–addr —) setzen kann.

Damit kann man insbesondere die zwei Aktionen vorne oder hinten anhängen sehr leicht implementieren, und schwierigere Operationen wie „an dritter Stelle einfügen“ sind auch möglich. Recognizer selber sind einfach Forth–Wörter, was hier auf dem Stack liegt, sind also xts. Recognizer–Stacks kann man auch hierarchisch aufbauen, deshalb gibt es hier eine Adresse. Zur Abarbeitung eines Recognizer–Stacks gibt es:

DO–RECOGNIZER (addr u rec–addr — token r:table | addr u r:fail) Führt der Reihe nach alle Recognizer im Stack aus, bis der erste etwas anderes als r:fail zurückgibt

R:FAIL (– r:fail) Tabelle, deren drei Methoden bei der Ausführung `-13 throw` machen.

Vordefinierte Recognizer

Im Kernel werden zwei unverzichtbare Recognizer definiert: WORD–RECOGNIZER und NUM–RECOGNIZER. Das ist das, was man vom traditionellen Forth–Interpreter ja auch kennt: Erst nach Wörtern suchen, und dann Zahlen wandeln.

```
: lit, ( n -- ) postpone Literal ;
: nt, ( nt -- ) name>comp execute ;
: nt-ex ( nt -- ) name>int execute ;
' nt-ex ' nt, ' lit, recognizer: r:word
: word-recognizer ( addr u -- nt r:word | addr u r:fail )
  2dup find-name
  [ [IFDEF] prelude-mask ] run-prelude [ [THEN] ] dup
  IF  nip nip r:word  ELSE  drop r:fail  THEN ;

: 2lit, postpone 2Literal ;
' noop ' lit, dup recognizer: r:num
' noop ' 2lit, dup recognizer: r:2num
: num-recognizer ( addr u -- n/d table | addr u r:fail )
  2dup 2>r snumber?  dup
  IF  2rdrop 0> IF r:2num ELSE r:num THEN EXIT  THEN
  drop 2r> r:fail ;
```

Beispiel: String–Literal

Als Beispiel noch ein Recognizer, der ein echtes String–Literal parst, also eins ohne state–smartes S“ und die ganzen Probleme damit:

```
: slit,  postpone sliteral ;
' noop ' slit, dup recognizer: r:string
: string-recognizer
    ( addr u -- addr' u' r:string | addr u r:fail )
    2dup s\" \"" string-prefix?
    IF    drop source drop - 1+ >in !
          \"-parse save-mem r:string
    ELSE  r:fail  THEN ;
' string-recognizer
forth-recognizer get-recognizers
1+ forth-recognizer set-recognizers
```

Damit kann man dann sogar Sachen wie

```
postpone "Eine String-Konstante"
```

machen.

Ausblick

Als offene Frage bleibt für mich noch, ob der Recognizer wirklich schon ein vorgeparstes Wort übergeben bekommen soll, oder das auch besser selber macht — das String–Literal macht das ja im Wesentlichen selbst.

Literatur

[1] Anton Ertl, Usenet Posting *number parsing hooks,* `https://groups.google.com/forum/?fromgroups#!msg/comp.lang.forth/r7Vp3w1xNus/Wre1BaKeCvcJ`

[2] Matthias Trute, *Reconizer — Interpreter dynamisch verändern,* VD 2011–02

Forth–Gruppen regional

Mannheim **Thomas Prinz**
Tel.: (0 62 71) – 28 30 (p)
Ewald Rieger
Tel.: (0 62 39) – 92 01 85 (p)
Treffen: jeden 1. Dienstag im Monat
Vereinslokal Segelverein Mannheim e.V. Flugplatz Mannheim–Neuostheim

München **Bernd Paysan**
Tel.: (0 89) – 41 15 46 53 (p)
`bernd.paysan@gmx.de`
Treffen: Jeden 4. Donnerstag im Monat um 19:00 in der Pizzeria La Capannina, Weitlstr. 142, 80995 München (Feldmoching–Hasenbergl).

Hamburg Küstenforth
Klaus Schleisiek
Tel.: (0 40) – 37 50 08 03 (g)
`kschleisiek@send.de`
Treffen 1 Mal im Quartal
Ort und Zeit nach Vereinbarung (bitte erfragen)

Mainz Rolf Lauer möchte im Raum Frankfurt, Mainz, Bad Kreuznach eine lokale Gruppe einrichten.
Mail an rowila@t-online.de

Gruppengründungen, Kontakte

Hier könnte Ihre Adresse oder Ihre Rufnummer stehen — wenn Sie eine Forthgruppe gründen wollen.

µP–Controller Verleih

Carsten Strotmann
`microcontrollerverleih@forth-ev.de`
`mcv@forth-ev.de`

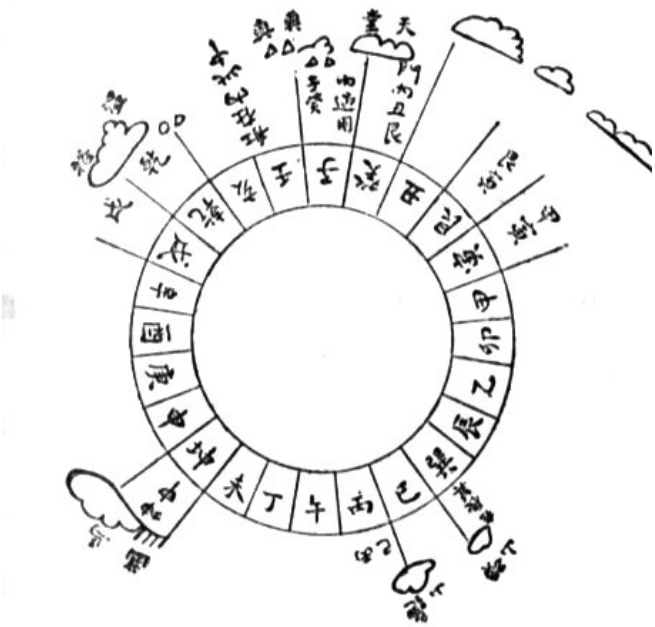
Spezielle Fachgebiete

FORTHchips (FRP 1600, RTX, Novix)	**Klaus Schleisiek–Kern** Tel.: **(0 40) – 37 50 08 03** (g)
KI, Object Oriented Forth, Sicherheitskritische Systeme	**Ulrich Hoffmann** Tel.: **(0 43 51) – 71 22 17** (p) Fax: **– 71 22 16**
Forth–Vertrieb volksFORTH ultraFORTH RTX / FG / Super8 KK–FORTH	Ingenieurbüro **Klaus Kohl–Schöpe** Tel.: **(0 70 44)–90 87 89** (p)

Termine

Mittwochs ab 20:00 Uhr
Forth–Chat IRC `#forth-ev`

12.–14. September 2012 Forth200x
14.–16. September 2012 EuroForth



Möchten Sie gerne in Ihrer Umgebung eine lokale Forthgruppe gründen, oder einfach nur regelmäßige Treffen initiieren? Oder können Sie sich vorstellen, ratsuchenden Forthern zu Forth (oder anderen Themen) Hilfestellung zu leisten? Möchten Sie gerne Kontakte knüpfen, die über die VD und das jährliche Mitgliedertreffen hinausgehen? Schreiben Sie einfach der VD — oder rufen Sie an — oder schicken Sie uns eine E–Mail!

Hinweise zu den Angaben nach den Telefonnummern:
Q = Anrufbeantworter
p = privat, außerhalb typischer Arbeitszeiten
g = geschäftlich
Die Adressen des Büros der Forth–Gesellschaft e.V. und der VD finden Sie im Impressum des Heftes.

EuroForth 2012
28th EuroForth Conference
Exeter College, Oxford, England
September, 14th to 16th, 2012

www.exeter.ox.ac.uk

EuroForth is consistently the best international Forth conference, attracting people from as far away as Australia, South Africa and the USA. EuroForth 2012 will be held at Exeter College, Oxford.

Exeter College is still situated in its original location in Turl Street, and was founded in 1314 by Walter de Stapeldon of Devon, Bishop of Exeter and later treasurer to Edward II. Its peaceful quadrangles and beautiful garden provide an idyllic setting for meetings and conferences, and the College has a long and proud tradition of providing for the needs of such guests.

The organiser is:

Janet Nelson
Micross Automation Systems
Tel: +44(0)1989 768080
Fax: +44(0)1989 768163

The conference will be preceded by a Forth 200x standards meeting.

August 1:	Notification of acceptance of academic stream papers
September 5:	Deadline for camera-ready paper submission (academic and industrial stream)
September 12–14:	Forth200x meeting
September 14–16:	EuroForth 2012 conference

Links

- EuroForth 2012 Home page `http://www.mpeforth.com/euroforth2012.htm`
- Call for Papers `http://www.complang.tuwien.ac.at/anton/euroforth/ef12/cfp.html`
- EuroForth Home `http://www.complang.tuwien.ac.at/anton/euroforth/index.html`