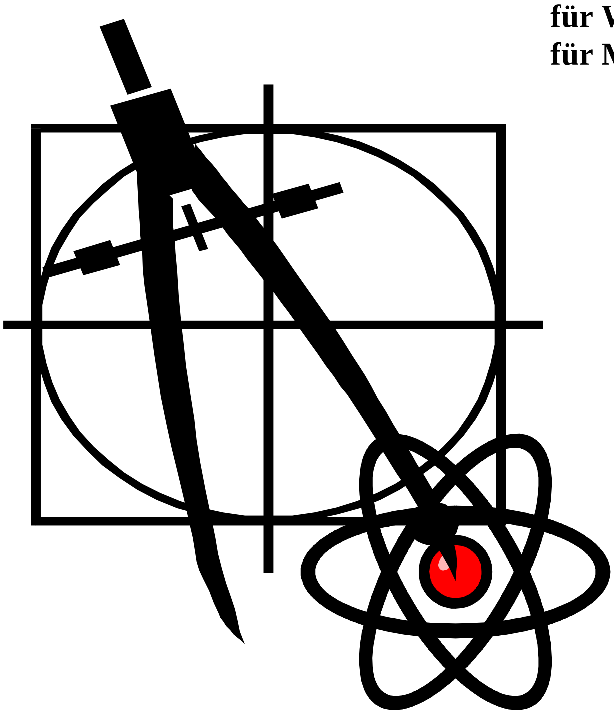
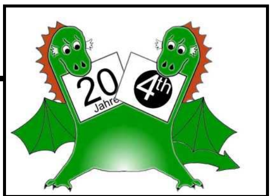
VIERTE DIMENSION

2 / 2004

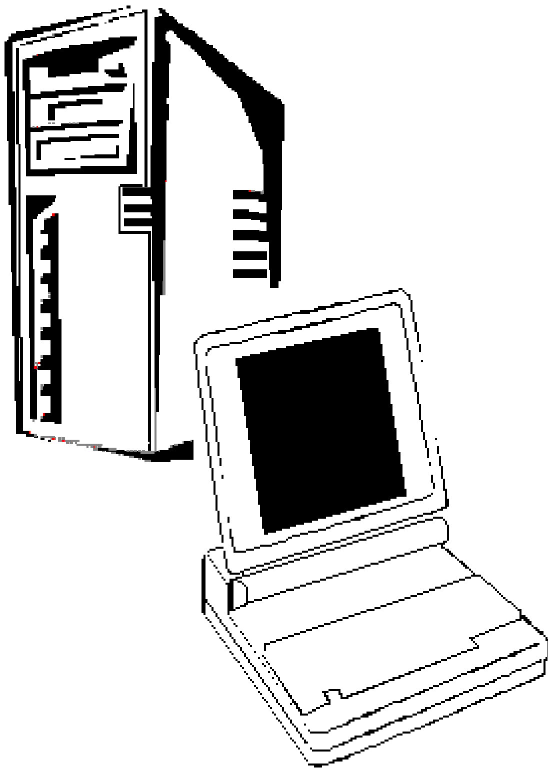
Das FORTH-Magazin



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



In dieser Ausgabe:



Dienstleistungen und Produkte fördernder Mitglieder des Vereins

Inhalt

Diese Ausgabe der VD wird vermutlich ca. vier bis sechs Wochen nach dem Erscheinen der Druckausgabe im Internet auf der Web-Seite der Forthgesellschaft e.V. veröffentlicht.

http://www.forth-ev.de

Eine PDF-Version dieser Ausgabe wird ab dem Zeitpunkt der Veröffentlichung im Internet ebenfalls zur Verfügung stehen. Bitte wenden Sie sich hierzu über die oben angegebene Adresse an den Webmaster der Forthgesellschaft e.V. oder an die Redaktion der „Vierte Dimension“. *fep*

In der nächsten Ausgabe finden Sie voraussichtlich:

- Inhalte tauschen, ohne Intermezzo
- Lineare Interpolation in Tabellen
- Biographie von Charles Moore
- Hashing in der Praxis
- OCR Eingabegeräte
- CF Speicherkartentreiber
- Forth auf Flashcontrollern

Editorial

IMPRESSUM

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth-Gesellschaft e.V.
Postfach 19 02 25
80602 München
Tel.: (0 89) 1 23 47 84
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

Friederich Prinz
Hombergerstraße 335
47443 Moers
Tel.: (0 28 41) 5 83 98
E-Mail: ***VD@FORTH-EV.DE***

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluß

März, Juni, September, Dezember jeweils in der dritten Woche

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00 € + Porto u. Verpackung

Manuskripte und Rechte

Liebe Leser,

die Tagung und die Mitgliederversammlung auf Fehmarn waren ein einziges Fest. Schade, wenn Sie nicht dabei waren! Wir werden aber, wie bereits in den letzten Jahren geübt, Beiträge und Berichte von Fehmarn in den nächsten Ausgaben unserer Zeitschrift aufarbeiten.
Eine besonders schöne Nachricht will ich aber schon jetzt an Sie weitergeben. Auf Fehmarn konnten die Tagungsteilnehmer zwei neue Mitglieder persönlich in der Forthgesellschaft willkommen heißen. ***Joachim Soll*** aus *Pinneberg* ist seit 1. April 2004 Mitglied in unserem Verein. ***Malte Bitter*** aus *Hamminkeln*, vielen von uns bereits persönlich bekannt und seit der Tagung in Oberammergau häufig auch Gast unserer Tagungen, ist seit dem 1. Januar 2004 Mitglied der Forthgesellschaft. Und auch ***Stefan Schmiedl***, den wir hoffentlich in 2005 auf der Tagung in Sachsen begrüßen dürfen, ist seit dem 1. Januar 2004 Mitglied der Gesellschaft. Herzlich willkommen Ihnen/Euch allen!

Eine weitere schöne Nachricht betrifft die Zusammenarbeit mit unseren englischen Forth-Freunden, bzw. mit unserer Schwesternzeitschrift Forthwrite. Der erste Teil des Artikels von James Boyd über die virtuelle nichtdeterministische Maschine hat einige von Ihnen, liebe Leser, sehr interessiert. Natürlich waren wir deshalb darum bemüht, von Boyd und von der Forthwrite die Erlaubnis zu bekommen, auch den zweiten, ausführlicheren Teil für die Vierte Dimension übersetzen und abdrucken zu dürfen. Graeme Dunbar hat uns für die Forthwrite sofort jegliche Erlaubnis gegeben, auch den zweiten Teil dieser interessanten Arbeit zu veröffentlichen. Und damit uns die Aufbereitung nicht allzu schwer fällt, hat uns Hr. Dunbar gleich alle originalen Quelldateien zur Verfügung gestellt.

So macht das Arbeiten Spaß.

Ihr

Friederich Prinz

Quelltext-Service

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können diese Texte auch direkt bei uns anfordern und sich zum Beispiel per E-Mail schicken lassen. Schreiben Sie dazu einfach eine E-Mail an die Redaktionsadresse.

fep

Die Forthgesellschaft wird durch ihr Direktorium vertreten:

Prof. Dr. Fred Behringer
Dr. Ulrich Hoffmann
Dipl.Inf. Bernd Paysan

Kontakte: Direktorium@Forth-ev.de

Betreff: **Nicht deterministische Maschine**
Von: Ulrich Paul <upaul@paul.de>

... so ganz teile ich die Ansichten von Bernd Paysan nicht, aber Endgültiges kann ich auch erst sagen, wenn ich den Quellcode gesehen habe. Mit ihm stimme ich in einem überein: Der Autor (James A. Boyd) nimmt stillschweigend für CHOICE die Komplexität O(1) an, und das muß nicht unbedingt für die Realität (Implementierung auf einem deterministischen Rechner) gelten. Aber es kann. In meinen Augen hat er sich von rein theoretischen Überlegungen etwas blenden lassen (ohne Bosheit Bernd gegenüber: Das fällt unter die Rubrik "Betriebsblindheit"). Denn, wie ich den Artikel verstanden habe, beansprucht der Autor nicht, daß seine Maschine jede mögliche Lösung findet, sondern nur eine von ihnen - na ja, und das eben mit einem Aufwand von O(1). Und genau das halte ich für möglich, jedenfalls wenn folgende Bedingungen gegeben sind:

1.) Es gibt sehr viele Lösungen
oder
2.) Eine (Teil-)Lösung kann in einem sehr frühen Stadium verworfen, bzw. favorisiert werden.

Ist Fall 1 gegeben, dann ist das nur eine Abwandlung der Monte-Carlo-Methode. Die braucht (Bernd: Korrigiere mich, wenn meine Erinnerung hier versagt) tatsächlich nur O(1) Operationen zum Finden einer (nicht vorher bestimmten!) Lösung.

Im Fall 2 steht und fällt die Komplexität damit, ob es möglich ist, einen Entscheidungsbaum effektiv aufzustellen. Leider ist das in den meisten Fällen nicht trivial, und er muß ständig modifiziert werden. Damit steigt die Komplexität rapide an und - na ja, wir haben dann eben ein NP-Problem.

Was ich aber an dieser Stelle erwähnen will, sind die "genetischen" Algorithmen. Wie wohl allseits bekannt, emulieren sie die Vererbung und das Überleben des Stärksten. Damit kann man viele Optimierungsprobleme für praktische Zwecke hinreichend gut erschlagen - und das tatsächlich mit einem P-Aufwand! Wie die Praxis gezeigt - zumindest nach meinem Kenntnisstand - hat, liefern sie im Fall 1 recht häufig optimale, also nicht verbesserbare Ergebnisse, in der Regel zumindest solche, die in der Praxis als ein Optimum gelten. (In der Praxis ist das "Optimum" nicht identisch mit dem Optimum des Theoretikers! Der Theoretiker sucht das Optimum des Problems an sich; der Praktiker sucht das Optimum der Kombination "Lösung plus Aufwand zum Finden derselbigen". Für den Theoretiker ist also manche "praktisch optimale" Lösung schlicht und ergreifend suboptimal - und er kann das sogar beweisen!)

Im Fall 2 liefern die "genetischen" Verfahren überraschend gute Lösungen. Klar, sie sind praktisch alle (wohl um die 99,99%) für den Theoretiker suboptimal, aber erweisen sich in der Praxis doch als sehr nützlich. Aber, daß ich hier diese Algorithmen überhaupt mit Theoretikern in einem Satz erwähne, ist schon eine Sünde. Denn was liegt dem Philosophen ferner als Sex (ohne das aber keine Vererbung)?

Ich will hier nicht das Wort für die "genetischen" Algorithmen pauschal reden, sondern nur auf einen Aspekt aufmerksam machen; nämlich daß NP-Probleme unter gewissen Einschränkungen (Lösung ist nicht unbedingt optimal oder sie ist nur eine von vielen) durch P-komplexe Algorithmen gelöst werden können. Und damit bin ich wieder am Anfang: Ich bin gespannt auf die Implementierung von CHOICE und FAILURE.

Abschließend eine Frage: Fritz, weißt Du irgendetwas von Forth und den genetischen Ansätzen? Eigentlich sollte das bei Forth schon lange im Repertoir sein, da das sehr erfolgreich bei der Optimierung von Steuerungen eingesetzt wird, vor allem wenn es darum geht, sich verändernden Umständen anzupassen. Aber gehört habe ich davon nur in allen anderen Zusammenhängen.

CU Uli; Ulrich Paul

Die Frage von Ulrich Paul muß ich an dieser Stelle an die Leser der Vierte Dimension weitergeben. Ich selbst habe keine Kenntnisse über die Implementierungen genetischer Algorithmen; weder in Forth, noch in anderen Sprachen. Aber sicher können Leser unserer Zeitschrift hier weiterhelfen, und solche Erfahrungen und Kenntnisse mit uns teilen – vorzugsweise über die Vierte Dimension. fep

Betreff: **Ringtausch ausgeschlossen**
Von: Ulrich Paul <upaul@paul.de>

Hi Fritz,

habe gerade Deinen Beitrag in der VD gelesen. Deine Lösung rechtfertigt wohl nicht die Aufgabenstellung, da Dein Programm die Einsparung der Hilfszelle mehr als vernichtet. Es geht doch viel einfacher: Mit genau 3 Operationen, genausoviel wie der Ringtausch braucht.

Die beiden Speicherstellen heißen A und B. Zur Erklärung verwende ich aber auch die Bezeichner A' und B', aber nur um anzudeuten, daß sie ihren Inhalt geändert haben.

A' = A XOR B
B' = B XOR A' = B XOR A XOR B = A XOR B XOR B = A
A" = A' XOR B' = A XOR B XOR A = B

Voilà!

Den Trick habe ich ganz zu Anfang meiner Programmiertätigkeit einmal kennen- und liebengelernt. Der geht auch mit Floats, wenn diese in einem zusammenhängenden und ausgerichteten (aligned) Speichersegment untergebracht sind, so daß ein Zugriff mit Integeroperationen auch wirklich alle Bits erfaßt.

CU Uli; Ulrich Paul

Leserbriefe

Eine tolle Lösung! Vielen Dank! Dieser Kniff ist bei mir unter der Rubrik „echtes Fachwissen“ abgespeichert.
Daß die von mir vorgestellte Lösung weder perfekt ist, noch einen irgendwie gearteten Effizienzanspruch befriedigt, macht mir wenig Kopfzerbrechen. Schließlich war nur gefordert, auf die dritte Variable/Speicherstelle zu verzichten. In diesem Sinne hoffe ich, noch weitere Lösungsvorschläge vorgestellt zu bekommen.
Eine ausführliche Arbeit über das Warum und Woher Deiner Lösung, liegt bereits hier vor. Fred Behringer hat sich einer Mühe unterzogen, deren Ergebnis wir in der nächsten Vierte Dimension lesen können.
Auf Fehmarn kam im Zusammenhang mit minimalistischen Implementierungen eines Forthkerns in FPGAs die Frage auf, wie man sich helfen kann, wenn das SWAP fehlt. Nun, die Antwort hast Du hier gegeben.

fep

Betreff: **Ringtausch ausgeschlossen**
Von: Ulrich Paul <upaul@paul.de>

Friederich Prinz wrote:
> ...und natürlich in der nächsten Ausgabe der VD "unsterblich" machen

Aber schreibe auch dazu, daß ich das auch von jemandem gelernt habe und die Lösung also keinesfalls mir selber eingefallen ist. Aber von wem ich sie her habe, kann ich beim besten Willen nicht mehr nachvollziehen. Das liegt ja mindestens 20 Jahre zurück. Und damals habe ich erst einmal viel lernen müssen, da ich einfach ins kalte Wasser geworfen wurde.
Mir waren die Mikroprozessoren suspekt und überbewertet. Aber wer zahlt, schafft an! Also durfte ich in Rekordzeit BASIC und 6502-Assembler lernen, ganz zu schweigen von den grundlegenden Konzepten. Daß ich da jede Quelle ausgeschöpft habe, ist wohl klar. Und aus irgendsoeiner habe ich eben diesen Trick gelernt. Aber auch viele andere, viel weniger saubere Kniffe (selbstmodifizierender Code und so, nicht zu verwechseln mit Overlays!). Heute gebrauchen nur noch Viren-Programmierer solche Techniken und wohl auch zu recht sind solche Methoden verpönt.

Aber wenn ich die Entwicklung eines speziellen Projektes über die letzten 20 Jahre betrachte, dann ist das bezeichnend. Es geht um die Nivellier-Kalibrations-Anlage des Geodätischen Instituts an der TU München. 1982 wurde die erste automatische Anlage in Betrieb genommen. Die externe HW ist seit dieser Zeit immer die gleiche gebleiben, mit einer Ausnahme, nämlich daß der Schrittmotor durch einen leiseren ausgetauscht wurde.

Der Rechner, der damals die gesamte Steuerung des Meßablaufs, die Auswertung der Daten und die Erstellung der Plots und der Protokolle erledigte, war ein CBM 3032 mit 31743 Byte freiem RAM. Du hast richtig gelesen, nicht einmal 32 kB standen für Programme zur Verfügung. Also habe ich die Steuerung in Assembler geschrieben und dieser Teil blieb als einziger immer im RAM. Die anderen Module wurden jeweils von der Diskette nachgeladen. Diese Anlage funktionierte viele Jahre problemlos.

Jetzt mach ich einen Sprung in die Gegenwart, und wie sieht die Anlage derzeit aus? 2 Rechner teilen sich die Aufgabe (2 nur deswegen, weil nur auf dem einen die SW für die Kamera läuft und nur auf dem anderen die Steuerung der Anlage). Der eine Rechner hat 2 CPUs mit je 1,5 GHz und 1 GB RAM, der andere 1 CPU mit 1 GHz und 512 MB. Gut, die Erfassung der Striche auf den Latten geht jetzt nicht mehr über ein Mikroskop, sondern über eine Kamera, aber trotzdem sieht man ganz klar, daß all die Tricks von damals hier verpuffen. Jeder dieser Rechner hat tausendmal mehr Leistung als er braucht und trotzdem haben diese beiden Rechner ungefähr soviel gekostet wie damals der CBM (ca. 3500 DM) plus das Dual-Floppy-Laufwerk (ca. 4500 DM).

In den Zwischenversionen kam FORTH zum Einsatz, aber abgekapselt in einem eigenen Kasten mit dem RTX (Version für LVA Bad Godesberg), bzw. 2x Z80 (TUM). Die beiden Z80 teilen sich die Aufgabe und haben beide ein kleines Forth (ohne Compiler, nur eingeschränktem Interpreter) am Laufen. Aber seit der neuesten Version ist alles in C geschrieben, auch mein Teil. Mit Forth hätte ich nie einen Treiber unter Linux schreiben können. Aber das war für eine saubere Implementierung einfach notwendig.

Aber damit sind wir doch wieder beim alten Thema: Wie integriert sich Forth in moderne Umgebungen? Gar nicht, es will immer alles selbst und allein erledigen! Und dadurch ist es endgültig aus diesem Projekt verschwunden und wird auch nie mehr eine Chance haben, zurückzukehren.
Tja, so steht's; so ist das Leben.

CU Uli; Ulrich Paul

Neues vom Forth-Büro

Liebes Mitglied, lieber Abonnent,

es waren in diesem Jahr wieder neun Mitglieder, die ihren Beitrag nicht fristgerecht einbezahlt hatten. Das war sicher nur ein Versehen.

Überlegen Sie bitte, ob es nicht für Sie und uns einfacher wäre, am Lastschrift-Einzugsverfahren teilzunehmen. Einen Vordruck dazu finden Sie im Heft, wenn Sie bisher überwiesen haben.

Herzlichen Gruß, Ihr Forth-Büro

Rolf Schöne

Ein bemerkenswertes Beispiel dafür, wie sich Forth in eine ausgesprochen moderne Umgebung aus Hard– und Software integriert, durfte ich im Beschleuniger in Garching sehen. Andere, vermutlich aktuellere Beispiele können sicher die Profis unter unseren Lesern geben. Nur zu... *fep*

Betreff: **Korrekturbitte**
Von: Fred Behringer <behringe@ma.tum.de>

Lieber Fritz,
bewundernswert, wie schnell du die VD 1/2004 trotz deiner anfänglichen Befürchtungen geschafft hast. Das hat aber auch seine Schattenseiten. Es war als schnelle Antwort "zwischendurch" gedacht. Ich hätte es mir sonst sicher noch einmal durchgesehen. Es hat sich ein Fehler eingeschlichen: Im zweiten Beispiel muss anstelle der ersten 3 eine 2 stehen. Mir wäre sehr daran gelegen, wenn im nächsten Heft eine klitzekleine Korrekturbemerkung stehen würde.

Herzlichen Gruß, Fred

Vielen Dank, Fred, für das Lob. Wie Du weißt, wird es zunehmend schwieriger, immer in ausreichendem Maße Beiträge für die Vierte Dimension zu bekommen. Aber sowohl für unsere Zeitschrift, wie auch für die Forthgesellschaft insgesamt, gilt in Abwandlung ein alter Satz aus meiner beruflichen Welt, der besagt, daß “Bergbau nicht eines Mannes Sache allein“ ist. Darum gebe ich das Lob an alle fleißigen Helfer (Übersetzer, Korrekturleser, Autoren, Leserbriefschreiber...) weiter. *fep*

Korrektur:

> Die Fernsehwerbung verlangt, das Haus so zu bauen, das
> bestimmte, als wesentlich aufgefasste Kanten in einem
> einzigen Zug genau einmal durchlaufen werden, wobei die
> Punkte mehrfach berührt werden dürfen: "Das ist das Haus
> des Ni-ko-laus'."

> Beispiel: 4 **2** 5 3 2 1 3 4 5 . [Hier 2 statt 3]

> Das ist ein "Eulerscher Weg", aber ein "offener".

beh

Betreff: **forth in wikipedia**
Von: Michael Kalus <michael.kalus@onlinehome.de>

Schon mal da reingeguckt?:
http://de.wikipedia.org/wiki/Hauptseite
Stichworte: Forth und Moore

mka

Betreff: **Drahtlos**
Von: Michael Kalus <michael.kalus@onlinehome.de>

Moin.
Kennst du Herrn Drahtlos eigentlich schon?
Adolf hat den entdeckt.
http://www.dietrich-drahtlos.de/index.htm

Herzliche Grüße aus Bochum, *mka*

Betreff: **Neuigkeiten zum Beitrag "USB-Entwicklung mit FORTH" aus der VD 4/2003.**
Von: Carsten Strotmann <carsten@strotmann.de>

Nach dem Erscheinen des Beitrages über das ATARI-USB Projekt hat sich das Projektteam für eine neue Hardwarebasis entschieden. Anstelle des National Semiconductor NS9603 verwenden wir nun einen Cypress SL811HS USB-Controller-Chip. Dieser Chip hat den Vorteil, das er sowohl in USB-Slave sowie in USB-Host Geräten eingesetzt werden kann.

Auf der Projekt-Webseite unter

http://www.strotmann.de/twiki/bin/view/APG/ProjUSBCart

finden sich die Schaltpläne und Bilder der neuen Platine. Die FORTH-USB-Treibersoftware wird derzeit an den neuen USB-Chip angepasst und ist in der aktuellen Version auf der Webseite zu finden.

Wir würden uns freuen, wenn Entwickler anderer Systeme unser Design als Anregung nehmen. Insbesondere ist es interessant, einen Pool von über Systemgrenzen hinweg portablen Quellcode für USB-Gerätetreiber zu erstellen.

Carsten Strotmann

Betreff: **Fehler in mark (empty view etc)**
Von: Martin.Bitter@t-online.de (Martin Bitter)

Hallo Fritz,
wg. RCX arbeite ich jetzt wieder vermehrt mit ZF.
Ein 'alter' Fehler ärgert mich deshalb wieder.

Probiere es mal aus:

```
ZF starten
view dup (oder irgendetwas, wo VIEW funktioniert)
fload irgendetwas Gültiges
empty
view dup
```

Bei mir funktioniert dann VIEW nicht mehr, weil FILELIST bei EMPTY nicht angepasst wird. FILELISTs erster Eintrag zeigt dann auf einen Bereich, wo vor EMPTY der Name der letzten benutzten Datei stand. Der ist jetzt leer, bzw. nicht mehr im definierten Zustand, d.h. evtl. von PAD überschrieben etc. Die Linkliste FILELIST liefert keine gültigen Werte mehr und VIEW bzw. <VIEW> geht ins Leere: Kann File nicht finden.
...hier mein Lösungsvorschlag:

```
: MARK        ( -- )
        CREATE YHERE , FILELIST @ ,
        DOES> DUP 4 + @ FILELIST !
        DUP 2+ SWAP @
        (FRGET)  FORTH DEFINITIONS  ;
```

Gruß, Martin

Eine virtuelle nichtdeterministische Maschine in Forth

James A. Boyd
(JimBoyd@techemail.com)

Übernommen aus der Forthwrite,
Ausgabe 124, Febr. 2004,
mit freundlicher Genehmigung des Autors
und der Forthwrite.
Übersetzt von Friederich Prinz

In der vorausgegangenen Ausgabe (*der Forthwrite und der Vierte Dimension*) hat James Boyd die Technik der virtuellen nichtdeterministischen Maschine und ihre Implementierung vorgestellt. In der hier vorliegenden Fortsetzung diskutiert er die Implementierung und die Leistungsfähigkeit der Software.

Theorie zu Ablauf und Einsatz

Virtuelle nichtdeterministische Maschine, Theorie zum Ablauf

Obwohl die virtuelle nichtdeterministische Maschine den gleichen theoretischen Rahmen nutzt wie eine (tatsächliche) nichtdeterministische Maschine, handelt es sich hier einfach um eine Simulation. Das nichtdeterministische Verhalten wird durch die Implementierung einer nichtsequentiellen Suche im Lösungsraum simuliert, mit impliziten Rückschritten falls eine Auswahl nicht „korrekt“ gewesen ist. Dies ermöglicht (für gefundene Lösungen) die gleiche Aussagekraft, die auch eine (echte) nichtdeterministische Maschine erreichen würde, und führt gewöhnlich in deutlich weniger Schritten zu der angestrebten Lösung, als ein deterministischer Algorithmus benötigen würde. Der nichtdeterministische Algorithmus für das Acht-Damen-Problem fand eine Lösung in nur 58 Versuchen, während der deterministische Algorithmus hierfür 876 Versuche benötigte.

Das implizite Rückschrittverfahren ist durch eine Sicherung des Maschinenzustandes (Daten- und Returnstack) realisiert, die jedes Mal in eine Liste gültiger Auswahlen auf einen *Historystack* gerettet werden, wenn `choice` ausgeführt wird. Das Ausführen von `failure` erzwingt einen Rückschritt zum jüngsten Eintrag auf den Historystack, einfach durch eine Wiederherstellung des bei diesem Eintrag gültigen Maschinenzustandes, wobei die aktuelle Auswahl getilgt wird. Es ist, als wäre `failure` nie ausgeführt worden. Es wird einfach an der Stelle vor der letzten Auswahl eine andere Auswahl getroffen als diejenige, die zu der Ausführung von `failure` geführt hatte. Wenn die Auswahlmöglichkeiten der aktuellen Instanz von `choice` erschöpft sind, wird der Maschinenzustand für das vorhergehende `choice` wiederhergestellt. Das implizite Rückschrittverfahren arbeitet so lange, wie gültige Auswahlmöglichkeiten zur Verfügung stehen. Wenn alle Auswahlmöglichkeiten erschöpft wurden, ohne daß eine gültige Lösung gefunden wurde, bedeutet dies, daß es eine solche Lösung nicht gibt. Die Ausführung des Programms wird mit dem Wort nach `failure` fortgesetzt. Beachten Sie bitte, daß das Rückschrittverfahren von einer (echten) nichtdeterministischen Maschine in gleicher Weise genutzt würde und relativ transparent ist. Das Ausführen von `success` leert schließlich den Historystack und beendet damit das nichtdeterministische Verhalten.

Die Notwendigkeit, Rückverfolgungsdaten zu sichern, ist einer der Nachteile bei der Simulation des Verhaltens einer nichtdeterministischen Maschine. Eine echte nichtdeterministische Maschine würde keine Rückverfolgungsdaten benötigen, weil jede Auswahl, die sie träfe, „korrekt“ wäre; vorausgesetzt, es gäbe überhaupt eine solche Auswahlmöglichkeit. Bis heute existieren solche Maschinen allerdings nur in der Theorie.

Gewöhnlich besteht der Maschinenzustand aus den Inhalten zweier Stapel; dem Daten- und Returnstack. Bei einigen Algorithmen müssen vielleicht noch andere Speicherbereiche als Teil des Maschinenzustandes gesichert werden. Es werden daher zwei DEFER Worte als Platzhalter für den Code vorgesehen, der zur Sicherung und Wiederherstellung anderer Datenstrukturen notwendig ist; `SaveOther` und `RestoreOther`. Das vierte Beispiel in der Quelldatei VNMMISC.F zeigt, wie diese Worte genutzt werden können. Beachten Sie bitte, daß auch die zusätzlichen Daten in der umgekehrten Reihenfolge ihrer Ablage wiederhergestellt werden müssen.

Datenstrukturen, die nicht Teil des Maschinenzustandes sind, müssen mit besonderer Vorsicht genutzt werden. Ein Array wie das der Damen im n-Damen-Problem ist einfach zu handhaben, weil die Einträge (die Damen) sequentiell dem Array hinzugefügt werden. Datenstrukturen mit Einträgen, die vermischt oder verändert werden, müssen Teil des Maschinenzustandes sein, entweder als Einträge auf dem Datenstack, oder als in den Maschinenzustand eingebundene Codes, die diese (externe) Datenstrukturen sichern und wiederherstellen. Es wäre klug, das, was dem Maschinenzustand hinzugefügt werden soll, auf ein Minimum zu beschränken, um zu verhindern, daß der Speicherplatz für Rückverfolgungsdaten knapp wird.

Das n-Damen-Problem ist das eigentliche Demonstrationsobjekt dieses Beitrages. Das Problem wird in nicht wirklich vorhersagbaren Zeiten gelöst, die aber exponentiell zur Größe des Problems (zur Anzahl der Damen) anzusteigen scheinen. Die nichtdeterministische Version dieses Problems scheint sich polynominal entwickelnden Lösungszeiten anzustreben, die aber nicht unterhalb der polynominalen Lösungszeiten theoretischer nichtdeterministischer Maschinen liegen.

Die besten Kandidaten für eine virtuelle nichtdeterministische Maschine scheinen solche Probleme zu sein, deren beste Algorithmen exponentielle oder polynominale Lösungszeiten von hoher Ordnung benötigen. Das Max-Cliquen-Problem und das Knapsack-Problem (*Rucksack-Problem, der Übers.*) [1] sind zwei solcher Probleme, die sich exponentiell entwickeln, jedoch in nichtdeterministischen Versionen dieser Algorithmen polynominale Anforderungen an die Lösungszeiten stellen. Diese nichtdeterministischen Algorithmen, ebenso wie weitere, die in „Fundamentals of Computer Algorithms“ [1] aufgeführt werden, können vermutlich gut in virtuellen nichtdeterministischen Maschinen arbeiten.

Virtuelle nichtdeterministische Maschinen im Vergleich zu nichtdeterministischen Steuerworten

L.L. Odette [2] hat uns nichtdeterministische Steuerungsworte gegeben, eine interessante Kontrollstruktur, die verspricht, die Ausdruckskraft von Forth zu erhöhen. Schon diese Struktur hat mit der impliziten Rückverfolgung gearbeitet und mit einer nichtsequentiellen Suche im Lösungsbereich.

Das nichtdeterministische Wort `nqueens` ist in seiner ganzen Erscheinung Odettes `queensoln` sehr ähnlich. Beide sind von einer nichtdeterministischen Natur. Der große Unterschied zwischen `nqueens` und `queensoln` liegt im unterschiedlichen theoretischen Rahmen, den beide Worte zum Ausgangspunkt haben.

Odettes nichtdeterministische Steuerungsworte basieren auf der Vorstellung eines “Controllers, der die Münze wirft ”, der zufallsmäßig auswählt, welche von zwei Richtungen weiter verfolgt werden soll. Odettes Münzen werfender Controller entspricht den theoretischen Erwartungen an diese Art von Controllern, bis auf die Tatsache, daß man gar keinen echten Zufallszahlengenerator hat. Das macht es schwierig, nach Verbesserungsmöglichkeiten für die Leistungsfähigkeit von Odettes Controller zu suchen.

Die virtuelle nichtdeterministische Maschine basiert auf einer nichtdeterministischen Maschine, die von Horowitz und Sahni beschrieben wurde. Die virtuelle nichtdeterministische Maschine zeigt einige der positiven Aspekte einer nichtdeterministischen Maschine, obwohl die Notwendigkeit von Verbesserungen deutlich sichtbar ist. Und obwohl die virtuelle nichtdeterministische Maschine mit Odettes nichtdeterministischen Steuerungsworten arbeitet, kann ihre Leistungsfähigkeit dadurch verbessert werden, daß ihr Gesamtverhalten näher an das theoretische Verhalten einer nichtdeterministischen Maschine herangeführt wird.

Test der ersten Auswahl der beiden nichtdeterministischen Worte `randomn` und `choice`

50 RandomnTest									
9998	9999	9999	9997	10000	10000	9998	10000	9998	9999
10000	10000	10000	9998	10000	9998	10000	10000	9998	9998
9999	9997	9999	10000	10000	10000	10000	9999	10000	10000
9999	10000	10000	10000	9998	9999	10000	9999	10000	10000
9998	9999	10000	9997	10000	9996	10000	10000	9999	9997
50 ChoiceTest									
3737	4547	9179	4726	7880	4659	8727	3934	4200	1362
9125	8719	9417	4501	839	7674	7749	1419	1842	6892
914	6961	7602	9968	8134	3469	1084	7994	6123	787
2761	9566	9104	6261	600	775	8793	4732	69	5219
4410	8545	4139	3987	2673	7543	3276	5676	7803	9071

Tabelle 1

Beide nichtdeterministischen Methoden sind von probabilistischer Art. Ihre nichtdeterministische Natur zieht eine nichtsequentielle Suche im Lösungsraum eines gegebenen Problems nach sich. Die virtuelle nichtdeterministische Maschine arbeitet so gut, weil sie ihre Auswahl in höherem Maße zufällig trifft.

`randomn` ist Odettes nichtdeterministischer Zahlengenerator. Die Tabelle 1 stellt beispielhaft Ausgaben von `choice` und `randomn` einander gegenüber. Der Code zur Erzeugung dieser Tabelle ist aus dem zweiten Beispiel in VNMMISC.F zu entnehmen.

Die Tabelle 1 zeigt die Ergebnisse der ersten Auswahl von `choice` und `randomn`, nach der Eingabe von 10.000. Die Rückverfolgungsdaten werden für jede Iteration gelöscht. Beide Tests mußten 50 Iterationen durchlaufen. Das nichtdeterministische Kontrollwort `randomn` neigt sich stark dem oberen Ende zu. Etwa die Hälfte aller Ausgaben war 10.000 und ein Viertel der Ausgaben war 9.999. Die Ausgaben von `choice` sehen recht zufällig aus. Sie sind nur durch die Güte der Zufallserzeugung des in `choice#` verwendeten Zahlengenerators begrenzt. Also ist auch die Wahl auf jede Rückverfolgungsinstanz dementsprechend zufällig. Auch die Auswahl in einer jeden Backtracking-Instanz ist (begrenzt durch die Fähigkeiten des Zufallsgenerators) recht zufällig.

Zeiten für drei n-Damen-Algorithmen.
Alle Zeiten sind in ms auf einer 1,6 GHz Maschine gemessen worden.

Tabelle 2

queens	nqueens			queensoln			deter-ministic
	fast	average	slow	fast	average	slow	
4	0	4	16	0	3	16	0
8	0	11	32	0	9	16	31
10	0	12	16	0	13	16	0
12	0	9	32	0	24	47	16
14	0	21	109	0	66	438	31
16	0	21	47	0	272	1313	250
18	0	39	203	0	1036	3656	1250
20	0	42	500	47	3452	15297	7125
22	15	78	750	297	26403	147437	73922
24	15	48	172	79	24604	143766	19953
26	15	98	485	563	191463	1082562	22047
28	16	121	1125	27672	628442	1713859	188750

Tabelle 2 zeigt die mittleren, kürzesten und längsten Zeiten, die zur Lösung des n-Damen-Problems von `nqueens`, `queensoln` und von einer deterministischen Methode benötigt wurden. Tabelle 3 ist eine Graphik der mittleren Zeiten, mit einer logarithmischen Skalierung der vertikalen Achse. Die benötigte Zeit zum Auffinden einer Lösung scheint für die deterministische Methode und `queensoln` exponentiell mit der Größe des Problems zu wachsen. Die Lösungszeit scheint für das nichtdeterministische Wort `nqueens` in Abhängigkeit von der Problemgröße nach einer Polynomfunktion dritten oder vierten Grades zu wachsen. Von `nqueens` wurde eine

Lösung für das 100-Damen-Problem sogar in rund 3 Minuten auf einem Commodore 64 gefunden!
Mit Win32Forth wurde eine Lösung für das 500-Damen-Problem in weniger als 18 Sekunden gefunden! Eine echte nichtdeterministische Maschine könnte das n-Damen-Problem in einer Zeit lösen, die nach einer Polynomfunktion zweiten Grades mit der Eingabegröße anwachsen würde.

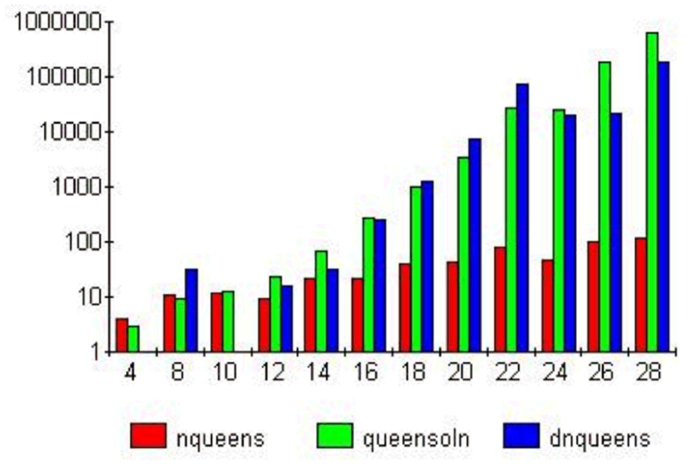


Tabelle 3

Tabelle 4 zeigt eine Graphik des History-Speichers, der von `nqueens` und `queensoln` in Beschlag genommen wird. Für alle Tests ist `queensoln` so modifiziert worden, daß die Damen in einem Array verwaltet wurden, anstatt auf dem Stack (Dies war hilfreich für die Leistungsfähigkeit von `queensoln`). Alle Rückverfolgungsdaten wurden vor jedem neuen Lauf entfernt. Die virtuelle nichtdeterministische Maschine benötigt deutlich weniger Speicher als die nichtdeterministischen Steuerungsworte. Obwohl `choice` mehr Daten ablegt als Odettes `oneof`, wird `choice` weitaus seltener ausgeführt.

Eine weitere Demonstration von Odettes Worten ist die Permutation von Listen. Dasselbe Konzept kann so geschrieben werden, daß es auf einer virtuellen nichtdeterministischen Maschine arbeitet, anstatt mit den nichtdeterministischen Steuerungsworten. Die Listen-Permutation als virtuelle nichtdeterministische Maschine wird als drittes Beispiel in der Datei VNMMISC.F wiedergegeben.

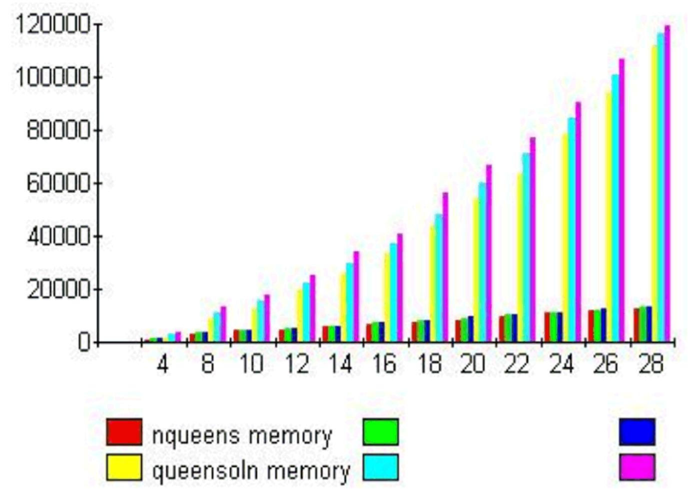


Tabelle 4

Bei dem Versuch Odettes Code in Win32Forth zu implementieren, sollte man übrigens sehr umsichtig vorgehen. Win32Forth arbeitet mit relativen Adressen, aber der Return-Stack enthält absolute Adressen.

Erzwungene Auswertung

Die Art und Weise in welcher die virtuelle nichtdeterministische Maschine implementiert ist, läßt ihren Aufruf „bei Bedarf" zu. Wenn ein nichtdeterministischer Algorithmus seine Arbeit nicht erfolgreich abschließt, kann dieser Algorithmus gezwungen werden, nach anderen Antworten zu suchen. Die Eingabe von `failure` über die Tastatur oder durch den Algorithmus selbst, kann die virtuelle nichtdeterministische Maschine starten. Die gleiche Idee kann auch mit Odettes nichtdeterministischen Steuerungsworten verfolgt werden. In dieser Weise arbeitet bereits das Beispiel zur Permutation.

Wir müssen dabei allerdings beachten, daß dies mit einer echten nichtdeterministischen Maschine nicht funktionieren würde. Jedes Auftreten von `failure` vor `success` würde den jeweiligen Algorithmus insgesamt in eine Fehlersituation führen. Dies ist als Technik zum Erzwingen eines Rückgriffs (backtracking) im ersten Beispiel in VNMMISC.F als einfacher Test der nichtdeterministischen Operationen genutzt worden. Wenn `ShuffleTest` ausgeführt wird, werden die Zahlen von Null bis Zehn in einer zufälligen Reihenfolge auf den Bildschirm ausgegeben.

Multitasking

Multitasking, wenn überhaupt nötig, mit einer virtuellen nichtdeterministischen Maschine, erfordert wegen der impliziten Stackmanipulationen besondere Vorsicht. Die virtuelle nichtdeterministische Maschine sollte ohne Modifizierungen auskommen, solange die nichtdeterministische Task unabhängig von beliebigen anderen Tasks arbeiten kann. Sobald mehr als eine nichtdeterministische Task gleichzeitig arbeitet, müssen die Puffer-Worte implementiert sein, damit jede nichtdeterministische Task ihren eigenen Puffer nutzen kann.

Wenn nur ein einziger Puffer zur Verfügung steht, dann darf nur die Task `failure` ausführen, die den nichtdeterministischen Algorithmus abarbeitet. Anderenfalls würde der Maschinenzustand, der mit der letzten Ausführung von `choice` gesichert wurde, durch die Stackinhalte ersetzt werden, die von der Task kommen, die gerade `failure` ausführt. Bedauerlicherweise schließt dies aus, daß nichtdeterministische Algorithmen in einer Hintergrundtask abgearbeitet werden, und daß Rückgriffe auf eine andere Lösung zum Beispiel über die Tastatur erzwungen werden.

Implementierung

Schnittstelle zum Puffer der virtuellen nichtdeterministischen Maschine

Anders als bei Odettes nichtdeterministischen Steuerungsworten werden die Rückverfolgungsdaten der virtuellen nichtdeterministischen Maschine in einem Puffer gesichert, anstatt sie dem Dictionary aufzukompilieren. Mit Hilfe der standardisierten Worte `allocate`, `resize` und `free` wird der Puffer für die Rückverfolgungsdaten dem jeweiligen Platzbedarf angepaßt und dem System zurückgegeben, wenn ein Algorithmus seine Arbeit beendet. Alle Worte, die geändert werden müssen, wenn die Puffer-Implementierung geändert werden soll, sind in der Datei VNMHISTORYBUFFER.F zusammengefasst. Implementierungsspezifische Bereiche wie das Multitasking sind dabei ausgespart worden.

Sichern und Wiederherstellen der Stacks

Die Worte `SaveDataStack`, `SaveReturn`, `RestoreDataStack` und `RestoreReturn` sind spezifisch für Win32Forth. Langsamere Versionen dieser Worte stehen in der Datei VNMSTACKS.F zur Verfügung. Diese Versionen sind allgemeiner, gehen aber immer noch davon aus, daß der Returnstack für die Rücksprungadressen verwendet wird. Win32Forth und eine der langsameren Versionen benötigen die folgenden Worte:

> `sp0` und `rp0` sind gewöhnlich zwei Uservariablen, welche die Anfangsadressen der beiden Stacks enthalten (dort, wohin die Stackpointer zurückgesetzt werden).
>
> `sp!` setzt den Datenstackpointer auf den auf dem Datenstack befindlichen Wert.
>
> `rp!` setzt den Returnstackpointer auf den auf dem Datenstack befindlichen Wert.
>
> `sp@` und `rp@` legen die Werte von Daten- und Returnstackpointer auf den Datenstack. Der Ausdruck `rp@ @` bewirkt das gleiche wie `r@` und der Ausdruck `sp@ @` entspricht einem `dup`.

Die Hauptoperationen der virtuellen nichtdeterministischen Maschine

`choice` nimmt eine Zahl vom Datenstack und sichert die Rückverfolgungsdaten in einem Satz von Null bis N einschließlich und wählt ein Element aus diesem Satz aus. Vorher wird von `choice` die Stacktiefe geprüft, um sicherzustellen, daß zumindest eine Zahl auf dem Datenstack liegt, auf deren Basis sich die Größe des zu erzeugenden Datensatzes spezifizieren lässt. Die CFA von `expand` wird auf dem Returnstack abgelegt, eine Kopie des Returnstacks wird gesichert, und nach der Restaurierung des Returnstacks wird die CFA dort wieder entfernt. Die Größe des Datensatzes (oder der Gruppe) wird vorübergehend auf dem Returnstack gesichert, während eine Kopie des Datenstacks angelegt wird. Am Ende dieser Arbeit wurde der Datensatz (oder die Gruppe) insgesamt von `group` erzeugt und gesichert, einschließlich der ersten Auswahl, die auf dem Datenstack bereit liegt.

`group` erzeugt die Daten einer Gruppe durch das Sichern eines Offsets in die noch zu erzeugende Datengruppe hinein und auf das jüngste Element der Gruppe. `group` nutzt den Zahlengenerator `choice#`, um einen Offset in die Gruppendaten zu erzeugen, und die Zahl zu produzieren, die an der Adresse dieses Offsets zu finden ist. Die gesamte Liste aller Auswahlmöglichkeiten wird solange nicht erzeugt, wie `failure` diese nicht anfordert.

`expand` dekomprimiert die Gruppendaten mit der Hilfe von `group`, legt die CFA von `(choice)` auf den gesicherten Returnstack in den Rückverfolgungspuffer und ruft `choice` auf, um ein Element auszuwählen. Die Rückverfolgungsdaten der jüngsten Arbeit von `choice` werden durch die Rückverfolgung selbst entfernt. `expand` und `choice` schützen den Puffer der Rückverfolgungsdaten durch Umschalten des entsprechenden Zeigers für Haupt- und Hilfspuffer. Dies macht das Wort aus der Puffer-Schnittstelle; `pswitch`, solange es eine gültige, noch nicht ausprobierte Auswahlmöglichkeit im jüngsten Arbeitsgang von `choice` gibt.

`generate` (n - -) legt n Einträge auf den Rückverfolgungsstack, von Null bis n-1 in aufsteigender Reihenfolge.

`failure` führt `(failure)` aus und schreibt die Nachricht „*No Solution*“ auf den Bildschirm. Das nichtdeterministische Verhalten von `failure` ist nach `(failure)` faktorisiert worden, für den Fall, daß die Ausgabe der Nachricht nicht gewünscht oder nicht benötigt wird. `(failure)` zeigt zwei unterschiedliche Verhaltensweisen, die davon abhängig sind, ob es Rückverfolgungsdaten gibt oder nicht. Wenn Rückverfolgungsdaten vorhanden sind, setzt `(failure)` den Zeiger des Hilfspuffers gleich dem Zeiger des Hauptpuffers, entfernt die Gruppendaten des Hilfspuffers und restauriert beide Stacks. Der Prozess der Wiederherstellung des Maschinenzustandes bringt den Zeiger auf den Hauptpuffer exakt an die Stelle zurück, auf die er zu dem Zeitpunkt gewiesen hat, bevor der letzte Batzen an Rückverfolgungsdaten gesichert wurde. Dies entfernt die zuletzt gesicherten Rückverfolgungsdaten effektiv. Die CFA auf dem Returnstack wird ausgeführt und damit entfernt. Der Programmfluß wird dort fortgesetzt, wo er sich befand, als der letzte Maschinenzustand gesichert wurde, also nicht nach `(failure)`. Wenn es keine Rückverfolgungsdaten gibt, dann führt `(failure)` das Wort `success` aus, um jegliches nichtdeterministische Verhalten abzubrechen. Der Programmfluß wird mit dem Wort fortgesetzt, das `(failure)` aufgerufen hat.

`success` setzt `SaveOther` und `RestoreOther` auf ein no-op und führt `remove` aus, um den Puffer zu leeren und dabei jegliches nichtdeterministische Verhalten zu beenden.

Wenn irgendwelche anderen Daten außer den Stacks als Teil des Maschinenzustandes gesichert werden müssen, dann müssen `SaveOther` und `RestoreOther` nach jeder Initialisierung von `success` und vor dem ersten Aufruf von `choice` initialisiert werden.

Erweiterungen der virtuellen nichtdeterministischen Maschine

Zur Erweiterung der virtuellen nichtdeterministischen Maschine können dem System andere Worte zugefügt werden. Die neuen Worte müssen die Rückverfolgungsdaten in einer für `failure` angemessenen Form ablegen. Auf dem Returnstack muß stets die CFA des Wortes abgelegt werden, das ausgeführt werden soll, nachdem `failure` gearbeitet hat. Und das muß natürlich geschehen, bevor `SaveReturn` ausgeführt wird und den Zugriff auf die CFA unmöglich macht. Auf dem Returnstack **muß** eine CFA abgelegt worden sein, und wenn auch nur die CFA von `noop` , einem Wort, das nichts bewirkt. Alle Parameter, die nicht ausdrücklich auf dem Datenstack erhalten bleiben sollen (wie zum Beispiel die Gruppengröße), müssen vor dem Ausführen von `SaveData` temporär vom Datenstack entfernt werden. Letztlich werden alle Gruppen durch den Aufruf von `group` eingerichtet, mit der Gruppengröße auf dem Datenstapel, oder durch Erzeugen einer besonderen Gruppe (wie zum Beispiel eine Liste von CFAs) und durch Sichern ihrer Größe auf dem Datenstack, oder durch Ablegen einer Null auf dem Datenstack, als Anweisung, keine Gruppe anzulegen. Beachten Sie, daß `failure` die Rückverfolgungsdaten während seiner eigenen Ausführung nicht sichert. Das Wort dessen CFA mit dem Returnstack gesichert wurde, muß dies beachten, wenn `failure` dazu führen soll, daß die gleichen Rückverfolgungsdaten bei seinem nächsten Aufruf restauriert werden.

`suspend` ist ein handliches Werkzeug zur Verwendung in Programmen; anstelle des üblichen Programmabbruchs als Reaktion auf Tastenanschläge. Es sichert eine Instanz von Rückverfolgungsdaten mit der Gruppengröße Null, gibt den Inhalt des Datenstacks auf dem Bildschirm aus und bricht das Programm ab. Ein `noop` wird auf dem Returnstack abgelegt, da die von `suspend` gesicherten Rückverfolgungsdaten nicht aufbewahrt zu werden brauchen. Das Programm setzt seine Arbeit an der Stelle fort, an der es von `suspend` unterbrochen wurde, auch wenn eine Rekursion ausgesetzt wurde und ohne Rücksicht auf die Stacktiefe bei einem Aufruf von `failure`. Dies ist sehr hilfreich, wenn es darum geht, ein Programm zu unterbrechen, um seine Funktionen zu überprüfen, oder um Variableninhalte einzusehen und anschließend die Ausführung durch die Eingabe von `failure` fortzusetzen. `suspend` gibt gleichzeitig ein hübsches Beispiel dafür, wie Rückverfolgungsdaten für `failure` aufbereitet sein müssen. Es kann auch sehr nützlich sein, `suspend` grundsätzlich anstelle von `abort` in einem Programm zu nutzen, nicht zuletzt in Fällen unbeabsichtigter Tastenanschläge.

Weiterführende Überlegungen

Parallele Verarbeitung

Wegen der probabilistischen Natur der virtuellen nichtdeterministischen Maschine ist die Lösungszeit für ein gegebenes Problem jedes Mal eine andere. Wie aus der Tabelle 2 entnommen werden kann, wird das Verhältnis der durchschnittlichen zur schnellsten Lösung umso größer, je größer die Anzahl der Damen im n-Damen-Problem wird. Als die Problemgröße fünfzig Damen erreicht hatte, war das schnellste Fünfzig-Damen-Problem über vierhundert Mal schneller als das durchschnittliche. Das gleiche Phänomen würde auftreten, wenn verschiedene Computer am gleichen Problem nichtdeterministisch arbeiten würden, wobei die jeweiligen Zufallszahlengeneratoren jeweils mit separaten Startwerten initialisiert würden. Die kürzeste Zeit für die Lösung aller n-Damen-Probleme von 8 bis 50 betrug weniger als 17 Millisekunden! Diese Erfahrung kann dazu genutzt werden, einen parallel arbeitenden Computer so zu programmieren, daß jeder seiner Prozessoren mit einem nichtdeterministischen Algorithmus an dem jeweils gesamten Problem arbeitet, aber ebenfalls mit einem separaten Eingangswert für den Zufallszahlengenerator. Der erste Prozessor der seine Arbeit beendet, würde dies dem führenden Prozessor rückmelden.

Neuronale Netzwerke als Zahlengeneratoren

Eine Möglichkeit die Leistungsfähigkeit der virtuellen nichtdeterministischen Maschine zu verbessern, ist der Zugriff auf ein neuronales Netzwerk als Lieferant für Zufallszahlen. `choice`, `success` und `failure` würden so modifiziert werden, daß sie ein Signal an einen Ausgabeport senden würden, um damit das Netzwerk zu trainieren. Das zahlengenerierende Wort könnte das Netzwerk „takten“ (anstoßen, eine Instanz der Signalabgabe aufzuarbeiten), nachdem es eine Zahl vom Netzwerk bekommen hat. `success` könnte das Ende oder den Start eines neuen Algorithmus anzeigen, und die Anzahl der Fehler für jeden Durchlauf von `choice` könnten als Trainingssignal verwendet werden. Je geringer die Anzahl der Fehler, umso besser ist die Ausgabe des Netzwerks. Ein hinreichend kompliziertes Netzwerk könnte das „Muster“ der richtigen Lösung erlernen und mit besseren Lösungsansätzen aufwarten.

Literatur

[1] Fundamentals of Computer A1gorithms
by Ellis Horowitz and Sartaj Sahni

[2] Nondeterministic Control Words in Forth
by L.L. Odette; Dr. Dobbs Journal September 1983

Nichtdeterministische Maschinen?

Friederich Prinz
Friederich.Prinz@t-online.de

Fragen an die Fachleute aus der Informatik und der Mathematik.

Beim Lesen, und später bei der Übersetzung des Aufsatzes von Boyd, haben sich mir Fragen aufgedrängt, die ich nur zu gerne mit den Lesern der Vierte Dimension diskutieren möchte. Es ist nicht allzu schwierig, James Boyds Ausführungen zu folgen. Aber mir fällt es sehr schwer, die Möglichkeit einer nichtdeterministischen Maschine zu akzeptieren. Eine „von Neumann Maschine" ist eine deterministische Maschine. Ist eine nichtdeterministische Maschine keine „von Neumann Maschine"?

Boyd beschreibt eine Maschine (virtuell oder real), die sich in den Ästen und Zweigen eines imaginären binären Baumes bewegt, sich dabei immer wieder zwischen zwei möglichen Alternativen "entscheidet" und sich ganz sequentiell der erwarteten Lösung oder Aussage nähert. Aus diesem Blickwinkel betrachtet, sehe ich die von Boyd beschriebenen Maschinen als klassische determinstische Maschinen. Mir scheint, daß der künstliche metamaschinelle Überbau – der sich hinter der Bezeichnung nichtderministisch verbirgt – mehr eine Frage der Sichtweise ist, vor allem aber eine Frage des Ergebnisses, das zu einem bestimmten Problem angestrebt wird.

Deterministisch heißt, aus der Sicht des Praktikers, daß die Parameter eines gegebenen Problems von einem bestimmten Algorithmus mit festen Regeln bei stets gleicher Maschinenleistung immer in der gleichen Zeit zu dem gleichen Ergebnis geführt werden.
Nichtdeterminstisch heißt, so wie ich Boyd verstanden habe, daß die Parameter eines gegebenen Problems von einem bestimmten Algorithmus mit festen Regeln bei stets gleicher Maschinenleistung in unterschiedlichen Zeiten zu irgendeinem Ergebnis geführt werden.

Dieses Verhalten wird durch einen inhärenten „Zufall" erreicht. Der Grad der Unvorhersagbarkeit ist wesentlich von der Qualität des Zufalls abhängig, das heißt von der tatsächlichen Zufälligkeit bei der Entscheidung für eine Auswahl zwischen Alternativen. Unbeschadet aller philosophischer Fragen zu Zufälligkeiten existieren Zufallsgeneratoren, mit deren Hilfe sich auch von sehr leistungsfähigen Maschinen die jeweils erzeugten Zufallswerte weder voraussagen noch hinreichend exakt wiederholen lassen. Ist diese gedankliche Basis ausreichend, um auf der technischen Grundlage sequentieller Maschinen physikalische (nicht virtuelle), nichtdeterministische Maschinen zu bauen?

Boyd spricht in seinem Artikel parallel arbeitende Mehrprozessorsysteme und neuronale Netze an. Seine Vermutungen bezüglich der möglichen Leistungssteigerung bei einem Einsatz eines Mehrprozessorsystems sind ohne weiteres nachvollziehbar. Aber sie beschreiben nach wie vor klassische, sequentiell arbeitende Maschinen.

Die Frage, ob sich der Aufwand rechtfertigen läßt, ist bisher noch gar nicht angesprochen worden. Die probabilistisch anzunehmenden Geschwindigkeitszuwächse, die sich bei der Erzielung irgendeines relevanten Ergebnisses erreichen lassen, sind beeindruckend. Diese Geschwindigkeit durch den Einsatz möglichst vieler Prozessoren und/oder Rechner mit größerer Wahrscheinlichkeit abzusichern, kann sehr schnell dazu führen, daß für ein relativ sicher sehr schnell erreichbares Ergebnis mehr Prozessorzeit und Energie aufgewendet werden muß, als ein klassischer, deterministischer Algorithmus verbrauchen würde. Welche Probleme und Aufgabenstellungen von allgemeinem Wert (keine Fragen aus den Elfenbeintürmen, bitte) rechtfertigen solchen Einsatz?

Die von Boyd vorgestellte virtuelle nichtdeterminstische Maschine erreicht ihre Ergebnisse durch Ausprobieren. Versuch und Irrtum führen zu relativ zufälligen Entscheidungen für einen neuen Versuch, ausgewählt aus verfügbaren Alternativen. Abhängig von der Qualität des Zufalls wird sich die Menge der Irrtümer und deren Qualität[1)] stets im Rahmen einer für das gegebene Problem charakteristischen Bandbreite bewegen.

Wie sieht ein sicheres statistisches Mittel für die Anzahl der Irrtümer aus, wenn ein gegebenes Problem fünf gleichwertige Lösungen haben kann, aber $100 * 10^6$ mögliche Irrtümer bietet?

Warum ist das statistische Mittel der real aufgegriffenen Irrtümer bei der Verwendung einer nichtdeterministischen Maschine kleiner als bei einer deterministischen Maschine? Warum spricht die Wahrscheinlichkeit für Versuch und Irrtum? Eine deterministische Maschine, die klassisch durch Ausprobieren aller Möglichkeiten eine oder alle Lösungen eines gegebenen Problems aufzufinden versucht, benötigt mehr Zeit für die Lösung(en), weil sie keinen möglichen Irrtum ausläßt. Die nichtdetermistische Maschine scheint effizienter zu sein, weil sie durch das Ausprobieren und Verwerfen – insbesondere durch das Verwerfen auch größerer Abschnitte, in denen durchaus auch Lösungen enthalten sein können – statistisch und real weniger Irrtümer „zur Verfügung“ hat als eine deterministische Maschine. Die planmäßige Einbeziehung des Zufalls führt nicht sicher zu der besten möglichen Lösung. Nichtdeterministische Maschinen oder Algorithmen decken auch nicht sicher alle möglichen Lösungen auf (selbst dann nicht, wenn man sie beliebig oft mit einem gegebenen Problem beschäftigt). Versuch und Irrtum scheinen effizienter als stringentes Arbeiten zu sein, wenn es darum geht, irgendeine mögliche Lösung zu finden (Nun, das kennen wir von der Arbeit mit Forth).

Wenn die möglichen Lösungen eines bestimmten Problems nicht quantitativ und qualitativ bekannt sind, läßt sich zu Beginn beliebiger Untersuchungen weder für eine deterministische noch für eine nichtdeterministische Maschine voraussagen, in welcher Zeit wie viele Lösungen in welcher Qualität von der jeweiligen Maschine oder dem jeweiligen Algorithmus „geliefert“ werden. Heißt die Aufgabenstellung, irgendeine der mögliche Lösungen in möglichst kurzer Zeit vorzulegen, scheint die planmäßige Einbeziehung des Zufalls bei der Entscheidungsfindung deutliche statistische Vorteile zu Gunsten der nichtdeterministischen Maschinen mit sich zu bringen. Verlangt die Aufgabenstellung aber, daß alle möglichen Lösungen vorgelegt werden, wird auch eine nichtdeterministische Maschine nicht darum herumkommen, alle möglichen Irrtümer durch Anfassen/Hingucken/Probieren zu eliminieren – bis alle verbleibenden Auswahlen korrekte Lösungen beinhalten. Im für die nichtdeterministische Maschine günstigeren Fall werden beide Maschinen nahezu gleichviel Zeit für diese Aufgabe benötigen.

Als Techniker neige ich dazu anzunehmen, daß eine ordentlich vorbedachte und vorbereitete Arbeit immer schneller, gründlicher und hochwertiger ausgeführt wird, als eine Arbeit, die Jemand „mal eben anfängt“. Das verführt mich zu der Vermutung, daß eine klassische deterministische Maschine immer schneller und vor allem sicherer alle möglichen Lösungen eines gegebenen Problems aufzeigen wird als eine nichtdeterministische Maschine das tun kann.

Liebe Leser – ich hoffe sehr, daß Boyds Artikel Sie ebenso nachdenklich gemacht haben wie mich. Bitte lassen Sie uns Ihre Meinungen wissen. Schreiben Sie zu Boyd und/oder zu meinen Überlegungen. Klären Sie auf. Belehren Sie mich und uns alle, bitte. Und versuchen Sie, Antworten auf die vielen offenen Fragen zu geben.

---

1) Die Erkenntnis, soeben eine nicht weiterführende Entscheidung getroffen zu haben, kann dazu führen, daß ein ganzer Ast mit vielen, bisher noch nicht bewerteten Möglichkeiten gekappt wird. Unter diesen noch nicht bewerteten Möglichkeiten könnte eine (oder mehrere) sein, die nur wenige Versuche entfernt eine Lösung beinhaltet. Die zufällige Entscheidung für eine der möglichen Alternativen kann zu einer sehr ineffizienten Lösungsfindung führen.

fep

```
\ Virtual nondeterministic machine
\ James Boyd    October 12th, 2003 - 5:04
\ needs words from just about all over the standard
\ plus a few that aren't
\ see file NONSTANDARD.F

decimal
needs vnmhistorybuffer.f

\ ************************ Stacks ******************

comment:
  The words used to save and restore the stacks. The idea to
  save and restore stacks came from L. L. Odette's article. This
  fast version is specifically for WIN32FORTH. See file
  VNMSTACKS.F for other stack saving and restoring words.
comment;

: SaveReturn   ( -- )
      rp@  cell+  rp0 @ over - tuck  ( # adr # )
      m>history  >history ;

: RestoreReturn   ( -- )
      r>  ( naddr ) \  nesting address
      history> rp0 @ over -  rp!  ( naddr # )
      rp@ swap mhistory>  ( naddr )
      >r ;

: SaveDataStack   ( -- )
      sp@ sp0 @ over - tuck  ( # adr # )
      m>history  >history ;

: RestoreDataStack   ( -- ? )
      sp0 @ history> dup>r  ( adr # )
      - sp! ( ? )
      sp@ r>  mhistory> ;

comment:
  These defered words allow adding code to save and restore
  other data with the machine state
  Just remember, whatever is saved to the history stack must be
  restored in the reverse order.
comment;

defer SaveOther          ' noop is SaveOther

defer RestoreOther       ' noop is RestoreOther

: SaveData   ( -- )       SaveDataStack  SaveOther ;

: RestoreData   ( -- )   RestoreOther  RestoreDataStack ;

\ ************************ VNM primitives **********
comment:
  choice# is a defered function generator. the generator MUST
  return a number in the range zero to N-1 where N is the
  number supplied to the generator. The number generator does
  NOT HAVE to be a random number generator.

Safer function generator:

: SaferChoice#   ( n -- n2 )   dup choice# swap 1- umin ;

  choice# is NEVER called with a parameter of zero.
comment;

defer choice#          ' random is choice#

\ Generate the group data in compressed form with one choice
\ removed

: group   ( n -- n2 )
      dup 1+  choice#
      2dup >history >history  2 >history  - ;

\ Randomly select one choice and remove from the group.

: (choice)   ( -- n )
      pswitch  history> dup 2 u<      \ ( size f )
      if   drop  history>  pswitch  exit   then
      history>  main>aux  over        \ ( size last_element size )
      choice# pointer-                \ ( size last_element )
      1 pointer+ history> -rot        \ ( chosen_elemen size
                                      \ last_element )
      >history pswitch  1- >history ; \ stuff last where
                                      \ chosen_element was

: generate   ( n -- ) \ Generate the set or group from compressed
                      \ form.
      dup cell  over
      if   um*   then   ?history      \ room for entire group?
      0  do   i (>history)   loop ;   \ generate group fast, room is
                                      \ available

: expand   ( -- n ) \ expand the rest of the group data with data
                    \ from history
      ['] (choice) >history   \ replace top of saved ret stack with
                              \ (choice)
      1 pointer-              \ restore pointer to where it was
      pswitch  history> drop  \ discard top of history stack -- it
                              \ is 2
      history> history>  over \ ( last_element displacement
                              \ last_element )
      1+ generate             \ ( last_element displacement )
      tuck  1+ pointer-       \ back pointer to where first chosen
                              \ value is
      >history  pointer+      \ stuff last where first chosen value is
      pswitch (choice) ;

\ ************************ Main VNM words *********
comment:
  Most nondeterministic algorithms can be written with just the
  three nondeterministic operations choice success and failure.
comment;

: choice   ( n -- n2 )
      depth 1 < abort" Unspecified set"
```

```
     dup if  ['] expand >r  SaveReturn  r> drop
           >r SaveData r>  group ( n2 )
     then ;

: success   ( -- )
     ['] noop dup is SaveOther  is RestoreOther  remove ;

\ Failure primitive -- will not display error message if history
\ data exausted

: (failure)   ( -- ? ) \ if machine state not restored then no stack
                       \ effect
     exhausted?                          \ if any history left
     0= if   main>aux  history> pointer-  \ skip over group
                                         \ data
           RestoreData  RestoreReturn     \ restore machine
                                         \ state
           r> execute              \ and execute cfa left on
                                   \ return stack
     else    success               \ else free memory for history
     then ;

: failure   ( -- ? ) \ if machine state not restored then no stack
                     \ effect
     (failure)
     cr ." No Solution" ;
 \ if there is history then this will not execute

\ ************************ suspend utility ***********
comment:
 A useful "utility" word, suspend saves history data and aborts
 so an algorithm can be "suspended" checked out and then
 resumed by typing failure.
 upon failure a no-op is executed and the machine state is
 restored to where it would be if suspend had not occured.
comment;

: suspend
     ['] noop >r  SaveReturn
     SaveData
     0 >history              \ no group so save zero group size
     cr ." Type failure to resume or"
     cr ." Type success to clear history data"
     cr .s
     abort ;
```

```
\ History buffer implemented with heap memory
\ James Boyd    October 31st, 2003 - 19:15

Comment:
 This implementation uses the Standard Forth words
 ALLOCATE FREE & RESIZE
Comment;

\ The history stack/buffer is a lifo structure for the most part.

decimal

0 value history          \ when non-zero, it holds location of
                         \ history stack
0 value historySize
variable MainPointer    MainPointer off
variable AuxPointer     MainPointer off
cell value pagesize     \ actual size made little difference in
                        \ nqueens times

: main>aux   ( -- )   MainPointer @  AuxPointer ! ;

: pswitch   ( -- ) \ switch main and auxilliary pointers
     AuxPointer @  main>aux  MainPointer ! ;

: remove   ( -- ) \ zero both pointers and free allocated memory
     MainPointer off  main>aux
     history if
          history free drop
     then
     0 to history  0 to historySize ;

: ?history   ( d -- ) \ make sure there is room for d bytes
     MainPointer @ 0  d+            \ ( d2 )
     2dup historySize 0  d<         \ ( d2 flag )
     if  2drop exit   then
     pagesize 0  d+                 \ ( d2 )
     dup if  remove  then           \ clear history and
     abort" History Full"           \ abort on overflow
     history  ?dup                  \ ( n addr addr or n 0 )
     if     over resize             \ ( n addr2 ior )
     else    dup allocate           \ ( n addr2 ior )
     then
     dup if  remove  then          \ clear history and
     abort" History Full"          \ abort if no more memory
                                   \  ( n addr2 )
     to history  to historySize ;

: exhausted?   ( -- f )   MainPointer @ 0= ;

: pointer-   ( n -- )   cells negate  MainPointer +! ;

: pointer+   ( n -- )   cells  MainPointer +! ;

: (>history)   ( n -- ) \ save cell 'n' to the history stack
     MainPointer @ history + !
     1 pointer+ ;

: >history   ( n -- ) \ save cell 'n' but check to make sure there
                      \ is room
     cell 0 ?history  (>history) ;

: history>   ( -- n ) \ retrieve cell 'n' from history stack
     1 pointer-
     MainPointer @ history + @ ;

: m>history  ( addr n2 -- ) \ save n2 bytes to history from addr
     dup 0 ?history
```

```
      MainPointer @  history + swap dup
      MainPointer +! cmove ;

: mhistory>   ( addr n2 -- ) \ transfer n2 bytes from history to
                             \ addr
      dup negate MainPointer +!
      MainPointer @  history + -rot cmove ;

remove
```

```
\ Win32Forth words used which are not standard

\ dup>r   just replace dup>r with  dup >r

: -rot   ( n1 n2 n3 -- n3 n1 n2 )   rot rot ;

1 cells constant cell

: noop ;

: off   ( addr -- )   false swap ! ;

: ?cr   ( n -- )   drop ;   \ Don't know how to do this one, sorry.
\ Code for IS borrowed from Rick VanNorman's 32 bit Forth
\ for its portability
: is
   state @ if    ' >body postpone literal postpone !
          else  ' >body !
          then ; immediate

: defer   create  ['] noop ,
   does>   @ execute ;

\ 31 bit pseudorandom shift register in software

variable rseed  here rseed !

: random   ( -- )   rseed @  dup
      16 lshift   swap
      2* dup   3 lshift   xor   16 rshift
      or dup rseed !
      16 lshift  um*  nip ;
```

```
\ N queens problem
\ James Boyd    September 22nd, 2003 - 3:30

Decimal

0 value queens

: freeQueens   ( -- )
      queens if   queens free drop   then
      0 to queens ;

: setQueenSize   ( n -- )
      freeQueens
      cells allocate   abort" No room for queens array"
      to queens ;

: attacks?   ( column queen -- column queen f )
      key?
      if   suspend   then
      over cells queens +    queens        \ if two queens on
                                           \ same row or
      ?do     i @   over = ?dup            \ diagonal we exit with
                                           \ true
            if     unloop exit   then      \ flag -- the new queen
                                           \ is under
            over  i queens -  cell /  -    \ attack.
            over  i @ -   abs = ?dup       \ otherwise we exit
                                           \ with a
            if     unloop exit   then      \ false flag -- no attack
      cell +loop
      false ;

: addqueen   ( column queen -- column+1 )
      over cells  queens + !  1+ ;

: showqueens   ( n -- )
      dup  cr space 3 .r ."  queens" cr
      0 do    i cells queens +  @ 1+  space 3 .r  16 ?cr
      loop ;

: (nqueens)   ( n -- )
      ?dup 0= if   cr ." No solution for zero queens" exit   then
      dup setQueenSize
      1- 0    ( n-1 column )
      begin
            over choice attacks?   ( n-1 column queen f )
            if     failure  2drop drop
                   exit
            then
            addqueen  2dup <       ( n-1 column f )
      until   ( n-1 n )
      space showqueens   drop ;

: nqueens   ( n -- )   success (nqueens) success ;
```

- Teil II -

```
\ Miscellaneous examples
\ James Boyd    10/13/01 03:03

Decimal

\ Example one -- simple test of nondeterministic words
\ The following should display the numbers 0 thru 10 in
\ random order.

: ShuffleTest  ( -- )
      success   10 choice  cr .  (failure)
      cr ." Test complete." ;
```

```
\ Example two -- test of choice vs. Randomn

: ChoiceTest  ( n -- )  success cr
      0 do   10000 choice  8 .r  success  16 ?cr   loop ;

\ RandomnTest requires L. L. Odettes Nondeterministic
\ Control Words

\ : RandomnTest  ( n -- )  cr
\       0 do   0 , 10000 randomn  8 .r  new  16 ?cr   loop ;

\ Example three -- Odette's permute example using choice

: -roll \ the following has no net stack effect:  n roll  n -roll
      dup
      begin  ?dup  while  3 roll >r 1- repeat
      begin  ?dup  while  r> swap 1- repeat ;

: insert
      1+ dup>r  dup  roll
      swap 1- choice -roll
      r> ;

: permute   ( N1 ... Nm m -- N1 ... Nm m )
      success  dup  if  1- recurse  insert  then ;

: printpermute   ( N1 ... Nm m -- )
      cr  0 ?do  3 .r  loop ;

: ptest   ( N1 ... Nm -- )
      depth dup>r permute  key?  if  suspend  then
      printpermute  (failure)
      cr ." No more permutations for"  r> 3 .r ."  numbers" ;

\ Example four -- Demo how to use SaveOther and
\ RestoreOther
\ three byte arrays

: array  create  allot ;        \ simple byte arrays
20 array small
50 array large
40 array medium

\ m>history and mhistory> take an address and byte count
\ as parameters

: SaveArrays
      small 20 m>history
      medium 40 m>history
      large 50 m>history ;

: RestoreArrays
      large 50 mhistory>
      medium 40 mhistory>
      small 20 mhistory> ;

: IncludeMyArrays   ( -- )
      ['] SaveArrays is SaveOther
      ['] RestoreArrays is RestoreOther ;

\ trivial example

: demo   ( n n2 -- )   success  IncludeMyArrays
      choice swap choice   cr .s  +  20 = 0= if   failure   then
      success ;
```

```
\ Alternate stack saving and restoring words
\ James Boyd    September 13th, 2003 - 3:25

: -?rdepth   C" rdepth" FIND NIP 0= ;

  -?rdepth [IF]

( requires -- rp0 rp@ cell )
: (NestDepth)   ( -- n )   rp0 @ rp@ - cell / ;
: NestDepth   ( -- n )   (NestDepth)   rp0 @ rp@ - cell /  - ;
: m-   NestDepth  0 do  postpone 1-  loop ; immediate
: rdepth   ( -- n )   rp0 @ rp@ -  cell /  m- ;

[ELSE]

( requires -- rdepth )
: (NestDepth)   ( -- n )   rdepth ;
: NestDepth   ( -- )   (NestDepth) rdepth - ;

[THEN]

: mr>   NestDepth  0 do  postpone r>  loop ; immediate
: m>r   NestDepth  0 do  postpone >r  loop ; immediate

: rStackPurge   mr>
      begin  rdepth  while   r> drop   repeat   m>r ;

: (RestoreReturn)   mr>  history>
      begin   ?dup   while   history> >r  1-   repeat   m>r ;

: RestoreReturn   mr>  rStackPurge  (RestoreReturn)  m>r ;

: SaveReturn   mr>  0
      begin   rdepth   while   r> >history  1+   repeat
      >history  main>aux  (RestoreReturn)  pswitch  m>r ;

: depth  ( -- n )  sp@  sp0 @ swap -  cell / ;

: StackPurge   begin   depth   while   drop   repeat ;

: (RestoreDataStack)   history> 0 ?do   history>   loop ;

: RestoreDataStack   StackPurge  (RestoreDataStack) ;

: SaveDataStack   0
      begin   depth 1-   while   swap  >history  1+   repeat
      >history  main>aux  (RestoreDataStack)  pswitch ;
```

```
\ Some more VNM stuff
\ James Boyd    October 31st, 2003 - 19:15

: sqrt   ( n -- n2 )   s>f fsqrt f>s ;

: ?prime   ( n -- f ) \ returns false flag for 0, 1 and negative
                     \ numbers
      dup  2 < if   drop false  exit   then
      dup  sqrt 1+ 2 ?do dup i mod 0= if 0= unloop exit then
      loop 0<> ;

: ?even   ( n -- f ) \ returns true flag if n is even
      1 and 0= ;

\ Always chooses an even number
: evenDemo   ( n -- )
      choice  dup ?even
      if     success  .
      else    failure  drop
      then ;

\ Always chooses an odd number if n > 0

: oddDemo   ( n -- )
      choice  dup ?even
      if     failure drop
      else    success  .
      then ;

\ Always chooses a prime number -- range 0 - n

: primeDemo   ( n -- n2 f ) \ returns n2 and a flag -- true if n2 is
prime
      choice  dup ?prime  dup
      if     success
      else    failure
      then ;

\ Always chooses two primes such that the second prime is
\ greater than 3 * the first prime

: prime2Demo   ( n -- )
      dup  choice  dup ?prime  0= if  failure  2drop exit  then
      swap choice  dup ?prime  0= if  failure  2drop exit  then
      2dup  swap 3 * <
      if     failure  2drop
      else    success  swap  cr 8 .r 8 .r
      then ;

\ Always chooses a number such that when added to n, the
\ result is prime.

: prime3Demo   ( n n2 -- n3 flag ) \ returns n3 and flag -- true
                                  \ if n3 is prime
      choice  +  dup  ?prime  dup
      if success  else  failure  then ;
```

```
\ Last minute update to Virtual Nondeterministic Machine
\ article ; James Boyd    November 11th, 2003 - 3:39

\ correction, badnqueens from the file on what NOT to do was
\ mistakenly defined:
\ badnqueens   ( n -- )   success (nqueens) success ;

\ it should have been defined as:
: badnqueens   ( n -- )   success (badnqueens) success ;


\ Once again, all the bad version does different is print a partial
\ result before failure with the result that a lot of nonsense is
\ displayed.

\ To work on Gforth a random number generator needs defined.
\       for the demos to work on Gforth:
\       need to define

: s>f   ( n -- f: f )   s>d  d>f ;
: f>s   ( f: f -- n )   f>d  d>s ;
: ?cr   ( n -- )   drop ;   \ Don't know how to do this one. Sorry.
                            \ need to replace the word   DUP>R
                            \ with the two words  DUP >R



\ nqueens with mistake
\ James Boyd
needs nqueens.f

Comment:
  Printing partial results with a nondeterministic machine is a
  bad idea. In the file VNMMISC.F ShuffleTest  displays
  partial results to test the virtual nondeterministic machine.
  In the same file Demo  displays the stack to show what the
  virtual nondeterministic machine is doing.
Comment;

\ prints partial results, or tries to. Run to see the mess!
: (badnqueens)   ( n -- )
      ?dup 0= if   cr ." No solution for zero queens" exit   then
      dup setQueenSize
      1- 0    ( n-1 column )
      begin
            over choice
            dup .       \ printing partial results is not a good idea!
            attacks?   ( n-1 column queen f )
            if      failure  2drop drop
                  exit
            then
            addqueen  2dup <        ( n-1 column f )
      until   ( n-1 n )
      space showqueens   drop ;

: badnqueens   ( n -- )   success (nqueens) success ;
  cr cr
  8 badnqueens
10 badnqueens
```

dah8300
Hitachi H8/300 Disassembler

von Risto A. Karola

http://www.pcuf.fi/~rak/index-eng.html

Übersetzt von Michael Kalus

Im Frühjahr 1995 habe ich mich mit dem Hitachi C-Compiler für den Microcontroller der Serie H8/300 abgemüht. Er war fehlerhaft und dem erzeugten Code war nicht zu trauen. Ich brauchte etwas, um den compilierten Code zu mustern. Daher schrieb ich den dah8300 - Hitachi H8/300 Disassembler. (Später bekam ich eine neuere, weniger fehlerhafte Version des C-Compilers und das Projekt konnte tatsächlich erfolgreich abgeschlossen werden. Wie die heutige Qualität der Compiler ist, weiß ich nicht – dieses Projekt war mein erstes und sicher auch letztes, bei dem ich einen C-Compiler für den H8/300 benutzt habe.)

Über...

Der Hitachi H8/300 code Disassembler dah8300 arbeitet unter DOS. Das binäre Image des H8/300 Code wird gelesen und in Assembler auf den Screen ausgegeben. Die Ausgabe kann in ein File umgelenkt werden. dah8300 ist in Forth geschrieben und wurde mit TCOM compiliert, dem public domain Forth–Compiler von Tom Zimmer; sehr praktisch für kleine Programme wie den dah8300.

Verwendung

Tippe einfach dah8300 und du erhältst Hilfe.

Ausgabe

Die Ausgabe erfolgt in fünf Spalten: Assembler Instruktion, Parameter der Instruktion, Programmadresse, Maschinenbefehl, Maschinenbefehl in ASCII Darstellung.

Ausschlussklausel für Haftung

dah8300 Version 1.0.0, Copyright © 1999 Risto A. Karola. dah8300 verfügt über ABSOLUT KEINE HAFTUNG. Es ist freie Software und darf gerne weiter verteilt werden, wenn die Bedingungen von GNU General Public License wie von der Free Software Foundation, Version 2 der Lizenz oder spätere Versionen, eingehalten werden.

Download

Diese erste veröffentlichte Version 1.0.0 wird möglicherweise auch die letzte sein. Trotzdem: Falls ich Fehlerberichte bekomme, könnte ich diese beheben.

Das gezippte Paket:

dah8300.zip ver 1.0.0 liegt zum Download auf meiner Homepage (s.o.). Den Sourcecode finden Sie darin in dah8300.4th ver 1.0.0.

(übersetzt: mka, 27.1.2004)

(Anmerkung: Warum heute, 9 Jahre später, dieser Beitrag noch aufgenommen wird? Weil das Problem weiter besteht. Um den Code im ROM des H8/300 im Lego Mindstorm zu lesen, gibt es komplizierte C-Tools. Einfacher war es, den binären Dump des ROM dem dah8300 zu geben. Keine Sekunde später gab es ein säuberlich disassembliertes, schlichtes Listing, gerade recht um in eine Exeltabelle eingelesen zu werden. Damit lässt sich vortefflich dokumentieren, wie dieser LEGO Brick programmiert ist. Die schon im Internet vorhandenen Beschreibungen der Routinen können so direkt und klar dem Code gegenüber gestellt werden. So wird der Brick bis ins letzte Flag und Register durchschaubar.) *mka*

Gehaltvolles

zusammengestellt und übertragen
von Fred Behringer

VIJGEBLAADJE der HCC Forth-gebruikersgroep, Niederlande

Nr. 41, Dezember 2003

Reguliere expressies
Albert van der Horst

In einem Musikstück, beispielsweise für das "wohltemperierte Klavier", mögen Noten in "editor-gerechter" Umschreibung vorkommen: c3 für "c in der dritten Oktave" usw. (Das Datum kommt in dem betreffenden Text auch vor.) Viele Editoren kennen Befehle wie s/1/2/g: Verändere (substituiere) alle Einsen in Zweien (global). Ohne Überlegung angewandt, wird auch das Datum entsprechend verändert. Das wollen wir nicht. Also müssen wir den globalen Befehle auf "reguläre Ausdrükke" beschränken, bespielsweise [a-g][0-9], was bedeuten soll: ein Kleinbuchstabe aus der Menge a bis g, gefolgt von einer Dezimalziffer. Es geht in diesem Artikel um String-Matching.

GOTO in Forth
De Schoolmeester

Hinter dem Pseudonym "De Schoolmeester" verbirgt sich Albert Nijhof mit "belehrenden" Artikeln - "Tutorials" würden wir Englischhörigen sagen. Ich, der Rezensent, übersetze einfach mal den Vorspann:

In diesem Artikel geht es um die GATE-Konstruktion. Sie ist eine flache Struktur, ohne Verschachtelung, die sich wie ein Forth-Wort verhält und die man also in einem Forth-Programm verwenden kann. Sie ähnelt einem endlichen Zustandsautomaten. Man kann damit auf übersichtliche Weise den wenig geschätzten, aber von manchen Problemen dringend verlangten GOTO-Spaghetti-Code compilieren.

Speciale Ushi-dag

Am 10. Januar 2004 fand von 10.00 bis 15.00 Uhr der "Ushi-Tag" in Maarsenbroek statt. Es wurden der Roboter "Ushi" und alles Drumherum vorgeführt. Alle waren eingeladen. Friederich Prinz und Martin Bitter bildeten mit ihrem "privaten" Besuch eine Abordnung der Forth-Gesellschaft.

Forth-Gesellschaft - Jahrestagung 2004

Eine Ankündigung unserer Jahrestagung auf Fehmarn im April 2004. Wir danken Albert Nijhof für die freundliche Zusammenarbeit mit uns, der Forth-Gesellschaft.

FORTHWRITE der FIG UK, Großbritannien

Nr. 124 Februar 2004

2 Editorial
Graeme Dunbar <g.r.a.dunbar@rgu.ac.uk>

Graeme berichtet von zwei neuen Projekten: (1) Der Vorrat an Einplatinen-Computern (F11-UK) der Gruppe um Jeremy Fowell ist ausverkauft. Eine neue Entwicklung (F12-UK) fürs "eingebettete Rechnen" ist im Anmarsch. (2) Die FIG-UK will eine CD mit Forth-Material für die Mitglieder herausgeben. Es wird um Mitarbeit an diesem Projekt gebeten und es wurde ein Preis ausgelobt.

3 Forth News
Graeme Dunbar

Unsere Tagung 20FG04 - euroFORTH 2004 - FIG-UK wird im November 25 - Spams (siehe unten) - upgedatetes Win32Forth-Tutorial von Dave Pochin - F11-UK wird weiter ausgeliefert werden können. --- Ich, der Rezensent, halte für wichtig: Webmaster Jenny Brien erstickt in Spam. E-Mailer werden gebeten, FIGUK in das Betrefffenster zu setzen. Alles andere wird schon auf dem Server eliminiert. Hurra, sage ich, endlich eine vernünftige Reaktion! Ich werde jetzt auch alles gleich auf dem Server wegschmeißen. Aber werde ich damit meiner Verantwortung gerecht? Was, wenn, wie kürzlich zweimal, jemand "von außen" Informationen über Forth und die FG haben möchte?

4 New Project: F11-UK Becomes F12-UK
Jeremy Fowell

Bei der Neuentwicklung sollen Vorschläge aus der F11-UK-Mailing-List aufgegriffen werden: I/O-Leitungen von 20 auf 40 erhöhen. Flash-Memory von 32k zu 128k. Echtzeit-Uhr von Batterie. Einsatz von Motorolas HCS12. Die Leser werden um ihre Meinung gebeten.

6 A Virtual Nondeterministic Machine in Forth
James A. Boyd <JimBoyd@techemail.com>

Zweiter Teil. Der erste Teil erschien auch im VD-Heft 1/2004, und zwar in der Übersetzung von Michael Kalus. Der zweite Teil, von dem ich gerade spreche, steht, so hoffe ich, in Übersetzung auch im vorliegenden VD-Heft 2/2004. Aus dem Inhalt (des zweiten Teils): "Nichtdeterministische Maschinen im Vergleich zu nichtdeterministischen Forth-Kontroll-Worten. Beide nichtdeterministischen Methoden enthalten ein wahrscheinlichkeitstheoretisches Element. Ihre nichtdeterministische Vorgehensweise zieht eine nichtsequentielle Suche im Lösungsraum nach sich. Die nichtdeterministische Maschine arbeitet so gut, weil sie ihre Auswahl mehr zufallsmäßig betreibt." Graeme Dunbar a.a.O. zur deutschen Übersetzung: "Ich begrüße die deutsche Übersetzung sehr. Die VD hat von ihrer ganzen Aufmachung her bessere Mittel, den Artikel zu präsentieren, als die Forthwrite das hat."

19 Letters

Eine Nachfrage nach Dave Pochins Win32-Forth-Tutorial, das der Schreiber im Internet nicht finden konnte. Dave hatte seine Website gerade "repariert".

20 New Project: the FIG-UK CD
Jeremy Fowell

Die Idee stammt von Douglas (Doug) Neale und Chris Jakeman und wurde auf der letzten Jahresversammlung heiß diskutiert. Die Mitglieder werden aufgerufen, Ideen beizusteuern. Bis jetzt vorgesehen sind unter anderem: eine VFX-Version, Win32Forth, Pygmy, F-PC und für Linux gForth und bigForth, Forth-Dokumente, Artikel über Forth, eine Bücherliste, eine Liste von Forth-Websites. Bis jetzt etwa 200 Megabyte. Auch ältere Forthwrite-Ausgaben ab Nr. 105.

Gehaltvolles

21 The FIG-UK CD: call for input
Douglas Neale

Doug sucht bei den Mitgliedern nach Ideen, wie die Forthwrite-Hefte vor Nr. 105, die nicht als PDF vorliegen, aufgenommen werden können. Vorgesehene Inhaltseinteilung: Anwendungen, Cross-Compiler, DOS-Forths, "eingebettetes" und PDA-Forth, FAQ, Forth im Internet, Forth-Standards, Forth Dimensions, Forthwrite-Archiv, Einleitung, Forth für Linux, Literatur, Forth für Mac, Leute von heute, ältere Maschinen, wissenschaftliches Forth und Windows-Forth. -- Ein Preis (ein Jahr freie Mitgliedschaft) wird für das beste CD-Label ausgelobt. Auswahl-Kriterium ist neben dem Erscheinungsbild die kürzeste Druckzeit auf Dougs HP-Tintenstrahldrucker.

22 Forth Gesellschaft 20FG04 20th Annual Conference
Fred Behringer

Ankündigung unserer Forth-Tagung 2004 in der von mir eingereichten Kurzfassung. Wir danken Graeme für die freundliche Aufnahme in der Forthwrite.

23 Nominations for the FIG UK Awards - 2003

Es wird um Kandidaten-Vorschläge für den Forthwrite-Preis und für den Forth-Preis gebeten. Die Gewinner erhalten ein Jahr freie Mitgliedschaft und werden in der "Ruhmeshalle" (auf der FIG-UK-Website) und in der Forthwrite lobend erwähnt.

24 Across the Big Teich
Henry Vinerts <Volvovid@aol.com>

Henrys Berichte vom Dezember 2003 und vom Januar 2004 über SVFIG-Aktivitäten in Originalfassung. Wir kennen sie in der Übersetzung von Thomas Beierlein. Diese Berichte werden in der Forthwrite stets mit den folgenden Worten eingeleitet: This material was prepared for Vierte Dimension by Henry Vinerts, and printed by kind permission of Forth Gesellschaft (German FIG).

26 Vierte Dimension 3/2003
Joe Anderson <jia@jus.abel.co.uk>

Joe rezensiert unser VD-Heft 3/2003. Die Zusammenarbeit zwischen ihm und uns hat sich bestens eingespielt.

28 Errata

Die Fehlangabe in der letzten Forthwrite ("VD 4/2002" statt richtig "VD 2/2003") wird hiermit korrigiert.

29 Dutch Forth Users Group

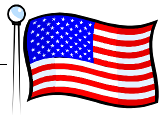
Die Anzeige zur Anwerbung von Mitgliedern der niederländischen Forth-Freunde. Es pendelt sich alles ein: Die Preisangaben erscheinen inzwischen in Euro-Beträgen.

30 Forthwrite Index
Jenny Brien

Jenny Brien unterhält eine Liste aller bisher erschienenen Forthwrite-Artikel auf der Website der FIG-UK, dreifach in verschieden sortierter Ausführung, die ständig den neueren Gegebenheiten angepasst wird. Zu Beginn eines jeden Jahres, auch diesmal wieder, wird diese Liste auch in der Forthwrite abgedruckt, und zwar ab 1990. Der Redakteur, Graeme Dunbar, macht darauf aufmerksam, dass sich die Mitglieder ältere Forthwrite-Hefte im "normalen" Buchleihverkehr (Porto erstatten) ausleihen können, dass aber auch die Möglichkeit besteht, Fotokopien zugeschickt zu bekommen. "Man möge mit dem Bibliothekar, eben auch Graeme, sprechen."

39 What Languages Fix - Not!
Graeme Dunbar

Graeme wartet noch stets auf Reaktionen zu Paul Grahams Bemerkungen im Forthwrite-Heft 122 darüber, dass neue Sprachen immer nur zu dem Zweck entwickelt werden, Fehler in bestehenden Sprachen zu beseitigen. Gilt das auch für Forth - und inwiefern?



Grüße aus Kalifornien!

Wenn Ihr das lest, wird die Silicon Valley Forth Interest Group wahrscheinlich im Web nicht länger als Silicon Valley Sektion (orig. Chapter) der FIG aufgeführt werden. Im Geiste von Forth wird das Wort "Chapter" ausfaktorisiert und SVFIG wird genügen, um dieses Schutzgebiet für gefährdete Arten in Kalifornien zu beschreiben. George Perry, der Präsident der FIG, sieht keinen Bedarf, die Autonomie der SVFIG zu diskutieren, des letzten überlebenden Zweiges einer Organisation, welche kurz vor dem Jahr 2000 noch über tausend Mitglieder zählte.

So, ich lade Euch ein http://www.forth.org zu besuchen und einen tieferen Blick in die Neuigkeiten der SVFIG und ihre Aktivitäten zu werfen. Da unser Webmaster, David Jaffe, sowohl Bemerkungen über die vergangenen monatlichen Treffen, als auch Ankündigungen über die zukünftigen fleißig veröffentlicht und Links zu den Webseiten unserer Redner beifügt, möchte ich nicht allzu detailliert über die technischen Vorträge in den Treffen sprechen (teilweise auch deshalb, weil sie oft "über meinem Horizont" sind und teilweise, weil sie zu spezifisch sind, um das Interesse des durchschnittlichen Lesers zu erwecken). Der Zweck meiner Botschaften - wie ich es schon zuvor gesagt habe - besteht darin, sie immer dann an Euch zu senden, wenn ich die Gelegenheit hatte, eine Live Show, mit echten

Forthern als Akteure zu erleben. Natürlich, wenn ich denke, daß ich meine persönlichen Beobachtungen von historischer, hysterischer oder philosophischer Natur mit Euch teilen sollte, werde ich sie wahrscheinlich der Botschaft beifügen.

Wie wir wohl erwartet hatten, war die Teilnahme nach dem jährlichen Forth-Tag und kurz vor den Ferien am 13. Dezember auf der knappen Seite. Gegen Mittag waren es maximal um die 18 Personen. Dr. Tim Duncan, Direktor des Bereiches für Digitale Audio Technik am Cogswell College, setzte im Laufe des Vormittages seine Vorlesungen und Vorführungen von MIDI-Implementationen fort. Aufgrund Tims Bekanntschaft mit Forth ist es im College Kurs-Verzeichnis zwischen C und Lisp als eine der empfohlenen Haupt-Programmiersprachen für den Bachelor-Grad enthalten.

Nach dem Mittagessen sprach Dr. Ting über Keyboards, chinesische Zeichen usw. Dies hielt die Zuhörer für ungefähr eine Stunde auf ihren Sitzen, woraufhin sie erleichtert "herausfielen" ('fell out' - Das ist ein Armee-Ausdruck: to "fall out" from being at attention -- aus dem aufmerksam sein "herauszufallen") zu einer großen Pause mit Geschwätz und Geplauder. Nebenbei schauten sie Kevin Appert zu, der mit einem Laptop, mit einem Beamer ausgestattet, quer durch das Web stöberte.

Das SVFIG Treffen am 24. Januar 2004 war ein wirklicher "Gewinn". Schon am Morgen waren mehr Leute anwesend als zum Forth-Tag im November. Unsere Organisatoren hatten eine super Arbeit geleistet. Sie hatten sogar den festlichen Auftritt in den San Jose Mercury News angekündigt. Wer war der Gefeierte? Vielleicht sollte ich vorher fragen, wie viele Leser sich noch an Rafael Delianos Artikel in der VD 1/96 erinnern. Und, nebenbei, wie viele haben schon mal von der Canon Cat gehört?

Nun, Dwight Elvey brachte eine Canon Cat, aber Jef Raskin kam sogar selbst, um über sein "THE"-Projekt zu sprechen (THE - The Human Environment / Die menschliche Umgebung). Da ist einfach nicht genug Platz in meiner Mailbox, um den Mann und alle seine Fertigkeiten zu beschreiben. Sogar Rafaels drei Seiten in der VD müssen noch ergänzt werden. Laßt mich nur sagen, daß Dr. Raskin wohl am besten als Schöpfer des Apple Macintosh Projekts vor 20 Jahren und als Autor des Buches "The Human Interface" (Addison Wesley, 2000), inzwischen in sieben Sprachen übersetzt, bekannt ist. Obwohl Raskin teilweise die Verantwortung für die "inhumanen" GUIs übernimmt, mit denen Computernutzer heute leben müssen, besteht seine selbstgestellte Aufgabe in der Verbesserung des Mensch-Computer Interfaces für zukünfige Generationen durch den Ersatz ineffektiver Designs und die Entfernung schlechter Verhaltensweisen. Er begann mit Forth auf der Canon Cat und er beabsichtigt zu Forth in seinem THE-Projekt zurückzukehren.

Am Nachmittag kam Randy Thelen, den wir auf dem Forth-Tag getroffen hatten, mit seinem selbstgebauten TTL-basierten Forth-Computer MIPPY und erntete Komplimente seitens der Oldtimer für sein Design und seine Präsentation. Dr. Ting beschloß den Tag mit einer Reihe von Themen, die mit seinen kürzlichen Arbeiten in Taiwan zusammenhängen. Er zeigte uns sein neuestes Buch "Programming Embedded Systems in Forth" (Programmierung eingebetteter Systeme in Forth), welches bei O'Reillys Ableger in Taipei verlegt wird und, außer dem verschiedenen eingestreuten Abschnitten in Forth-Code, auf allen 365 Seiten in chinesisch geschrieben ist. Der Hauptzweck des Buches ist es, den chinesischen Menschen beizubringen, wie man eine CPU entwickelt. Eine englische Version wird wahrscheinlich nur erscheinen, falls O'Reillys Hauptbüro sich für eine Produktion entscheidet.

Abschließend möchte ich Dr. Ulrich Hoffmann für die freundliche Einladung zur 20FG04 Forth-Tagung auf Fehmarn danken. Zuletzt war ich in dieser Gegend zum Kriegsende 1945 und zwar in Flensburg. Es tut mir leid, Freunde, ich werde nicht teilnehmen können, aber ich wünsche allen, die dort sein werden, eine gute Zeit.

Henry

übersetzt von Thomas Beierlein

Wie man Knoppix Linux das FORTH beibringt

Carsten Strotmann

Mit einer Knoppix-Linux-Live-CD kann (fast) jeder Rechner [1] im Handumdrehen in einen Linux Rechner verwandelt werden, ohne das Software auf die Festplatte installiert wird. Knoppix läuft direkt aus dem Hauptspeicher und von der CD. Ein auf der Festplatte installiertes System (Windows, DOS ...) bleibt unangetastet.

Eine solche Knoppix-Linux-CD lässt sich sehr einfach als portables Entwicklungs- und Demonstrationssystem verwenden, auch für FORTH-Programmierer. Im Rahmen der Beteiligung der FORTH-Gesellschaft am Linuxtag in Karlsruhe, Sommer 2003, wurde eine solche Knoppix-CD mit FORTH erstellt und an interessierte Besucher verteilt. Dieser Beitrag gibt ein paar Tips und Referenzen für interessierte Bastler, die eine eigenzusammengestellte Knoppix-CD erstellen möchten.

Knoppix-Linux beeinhaltet eine Reihe von vorinstallierten Programmiersprachen (GNU C/C++, Perl, Python, Java uvm.). Leider beeinhaltet die Standard-CD [2] (derzeit) noch keine vorinstallierte FORTH-Entwicklungsumgebung. Anhand der im

1) Knoppix für ia32 (x86) Rechner, Gentoo Linux auch für Apple PPC Mac. Live-CDs gibt es auch für NetBSD, FreeBSD und andere freie Unix-Systeme.)
2) http://www.knoppix.net/

Forth und Knoppix

Netz verfügbaren Anleitungen [3] zum "Remastern" (Ändern und Neu-Erstellen) der Knoppix-CD, kann jeder seine eigene Knoppix-CD zusammenstellen.

Vorgehensweise

Für das Remastern einer Knoppix-CD wird ein Linux System mit 3 GB freiem Festplattenplatz, 1 GB frei verfügbarer Hauptspeicher (nicht physisch, sondern z.B. 256 MB Ram und 750 MB Swapfile), Internetzugang und einen Nachmittag Zeit (am besten am Wochenende) benötigt.

1. Zuerst wird die Knoppix-CD gestartet und gemäß der Remaster-Anleitung auf die Festplatte kopiert.

2. Dann wechselt man mit dem "chroot" Befehl in das Verzeichnis mit dem Knoppix-CD Abbild auf der Festplatte (auch in der Anleitung beschrieben).

3. Für die zu installierenden FORTH-Programme muss Platz auf der Knoppix-CD geschaffen werden. Hierzu wird eines oder mehrere der installierten Knoppix-Programme entfernt. Für die Linux-Tag Knoppix-CD haben wir hierzu mit dem Debian Package-Manager "dselect" das Open-Office Paket entfernt. Dies ergibt ca. 70 MB freien Platz auf der CD, genug Freiraum für viele FORTH-Programme und Entwicklungsumgebungen.

4. Nun installieren wir die FORTH-Software in der unter Linux gewohnten Weise (wie in der Installationsanleitung der FORTH-Systeme beschrieben). Für einige Software, wie z. B. das GNU-FORTH, existieren schon vorgefertigte Debian-Pakete (Knoppix basiert auf der Debian GNU/Linux-Distribution), andere werden aus dem Quelltext übersetzt.

5. Einige FORTH-Systeme benötigen spezielle Dateien, welche beim Starten der Entwicklungsumgebung geladen werden. Diese Dateien werden im Heimverzeichnis des Benutzers gesucht. Unter Knoppix kann das Heimverzeichnis als Ramdisk im Hauptspeicher liegen, aber auch auf einem Wechseldatenträger (USB-Stick, PCMCIA-Speicherkarte etc.). Ist beim Start der Knoppix-CD das Heimverzeichnis nicht vorhanden, so wird es aus dem Inhalt des Verzeichnisses "/etc/skel" neu erstellt. Damit unsere FORTH-Systeme wie gewünscht funktionieren, müssen die notwendigen Dateien in "/etc/skel" angelegt werden. Um diese Dateien später beim Start des FORTH-Systems finden zu können, wird der aktuelle Dateisystem-Pfad vor dem Start des FORTH-Systems in das entsprechende Unterverzeichnis im Heimverzeichnis geändert werden. Hierzu werden die Programm-Dateien des FORTH-Systems gegen ein Startscript ersetzt (siehe Beispiel-Startscript für FICL). Die Programmdatei(en) des FORTH-Systems liegen nun unter "/usr/local/lib/<FORTH SYSTEMNAME>/bin/".

6. Die FORTH-Systeme werden direkt in der CHROOT-Umgebung innerhalb des Knoppix-Abbildes getestet. Für einen abschliessenden Test muss jedoch ein komplettes ISO-Abbild der Knoppix CD erstellt werden.

7. Vor dem Erstellen des ISO-Abbildes bitte nicht vergessen das eigene Heimverzeichnis (und andere Arbeitsverzeichnisse) wieder "aufzuräumen", d.h. Dateien, die während des Installierens und Testens der FORTH-Systeme erstellt wurden, wieder zu löschen, um Platzverschwendung auf der CD zu verhindern.

8. Die Erstellung des CD-ROM-ISO-Abbildes (Image) wird in der Knoppix-Remaster-Anleitung gut beschrieben.

9. Nun ist es Zeit für einen abschliessenden Test der CD. Um sich das Brennen einer CD zu sparen, kann ein System-Emulator wie VMWare [4] oder VirtualPC [5] benutzt werden.

Beispiel-Startscript für FICL

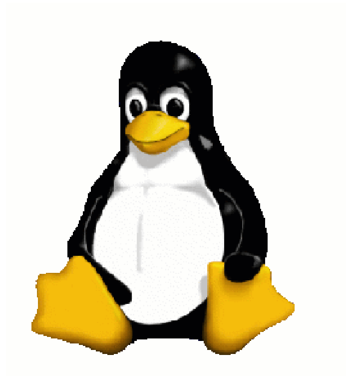
```
#!/bin/sh

cd ~/forth/ficl

/usr/local/lib/ficl/bin/ficl $*
```

Weitere Ideen

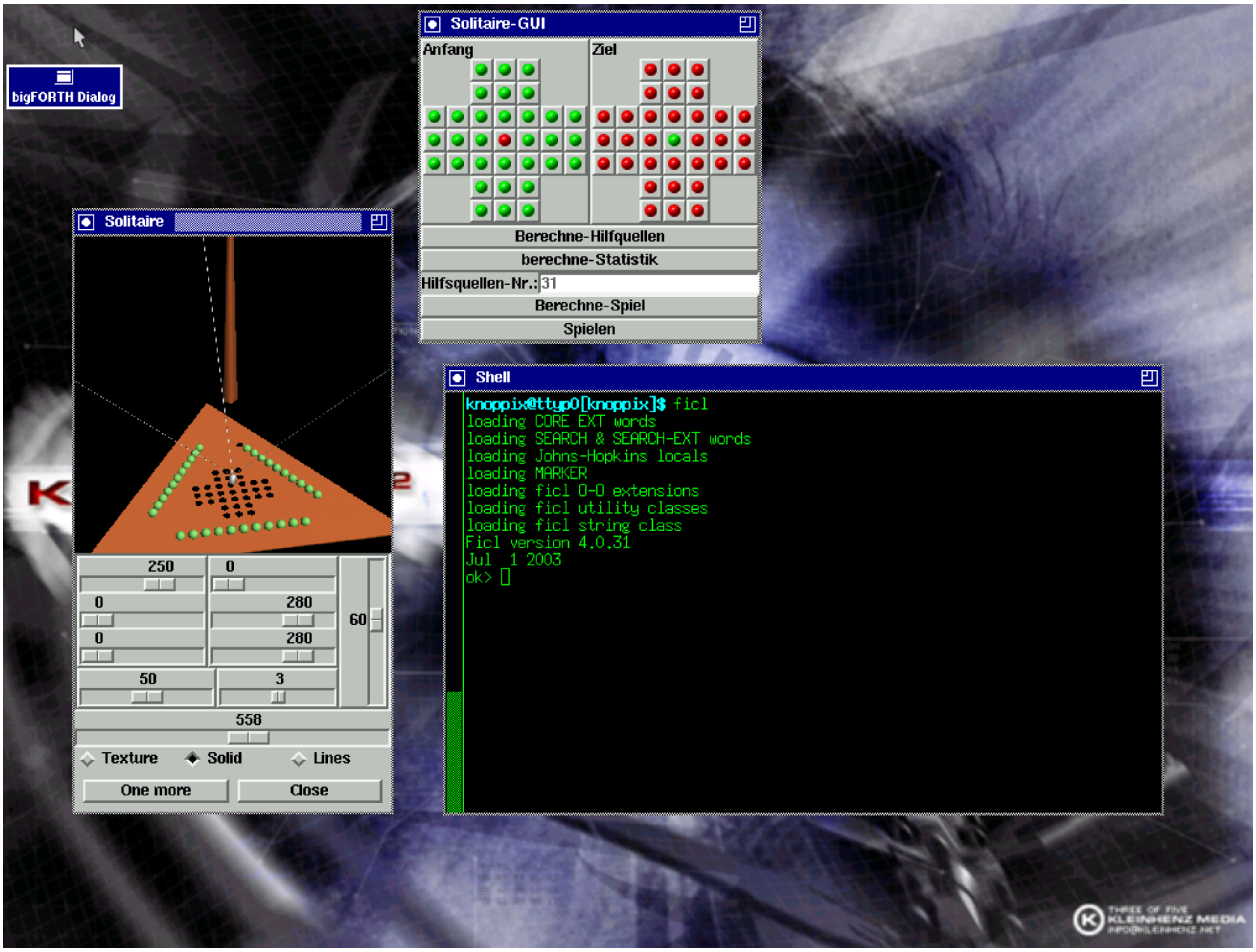
Für Linux existieren eine Reihe von System-Emulatoren. So gibt es Emulatoren für Windows (Wine), MS-Dos (dosemu, dosbox), Macintosh (Basilisk), Atari ST und Amiga, sowie viele Homecomputer und Microprozessorsysteme. Über eine Knoppix-CD mit diesen Emulatoren können entsprechnde FORTH-Systeme verwendet werden.

Zukunft

Für den Linuxtag in diesem Jahr (2004) wird es eine neue Knoppix-FORTH-CD geben. Bezugsinformationen sowie eine Sammlung von Installationsanleitungen für verschiedene FORTH-Systeme befinden sich auf der Homepage der FORTH-Gesellschaft; http://www.forth-ev.de/ .



4) http://www.vmware.com/
5) http://www.microsoft.com/windowsxp/virtualpc/



S-Record Datensätze für den RCX

Friederich Prinz
Friederich.Prinz@t-online.de

Das S-Record Format wurde von Motorola vor mehr als 20 Jahren entwickelt und ist längst zum Industriestandard ausgewachsen. Kleinere Änderungsversuche durch verschiedene Drittersteller haben die S-Records in Gänze unbeschadet überstanden. Der nachhaltigste Eingriff in die Vorgaben von Motorola ist wohl die Erweiterung von LSI, deren Mitarbeiter mit der Einführung ihres Logic Fast Load Formats den Standard um den S-Record-Typ S4 erweitert haben.

Wer im Internet nach Informationen über S-Records sucht, wird schnell fündig. Mit geringen Abweichungen in Details beschreiben die meisten Texte und Darstellungen den ursprünglichen Definitionsstand von Motorola. Daraus lassen sich alle Fragen beantworten, die z.B. bei der Arbeit an eigenen Werkzeugen zu S-Records aufkommen.

Ein solches Werkzeug wollte ich bauen, um einem in meinen Augen offensichtlichen Mißstand in LEGOs *.lgo Dateien abzuhelfen. Diese Dateien tragen in den Datensätzen nur 32 Byte an Code und/oder Daten. Jedem einzelnen Datensatz sind aber weitere 12 Byte an Steuerungsinformationen zugeordnet. Die stellen einen erheblichen Ballast dar, der vor allem den IR-Sendeturm belastet. Und diese Belastung führt zu den bekannten Problemen mit überlasteten Akkus in den Sendetürmen. Mit anderen Worten: mehr Daten in einem einzelnen Datensatz führen zu geringerem Verwaltungsüberhang. Was der IR-Turm nicht übertragen muß, belastet seinen Akku nicht. Und schneller wird die Übertragung insgesamt dadurch ebenfalls.

So war zumindest die erste, bezüglich der Übertragungszeit etwas naive Annahme. Tatsächlich kann ich mit den mir heute zur Verfügung stehenden Übertragungswerkzeugen (BricxCC[(1)]) S-Record Dateien an den RCX senden, deren Datensätze bis zu 128 Byte an Daten und/oder Code enthalten. Bei der Umwandlung ist die Dateigröße von 45.112 Byte auf 35.896 Byte zurückgegangen. Die Reduzierung der Dateigröße um rund 20 % hat allerdings nur eine rund 10%-ige Reduzierung der Übertragungszeit mit sich gebracht. Ich schließe daraus, daß nicht alle Byte aus dem Verwaltungsüberhang auch gesendet werden.

Das gilt aber vermutlich nur für das CRLF am Ende eines jeden Datensatzes.

Die maximale „Komprimierbarkeit" für S-Record-Dateien liegt bei 504[2] Daten-Byte in den Datensätzen. Damit lässt sich aber in bezug auf die Dateigröße kaum noch weiterer Reduktionsgewinn erzielen. Die von LEGO mit dem RCX 1.0 mitgelieferte Datei „firm0309.lgo" lässt sich mit 504 Byte in den S-Records von 45.112 lediglich auf 33.616 Byte komprimieren. Oberhalb von 128 Byte pro Datensatz lassen sich nur noch marginale Reduzierungen der Dateigröße erreichen. Zwar wird mit jeder weiteren Verdoppelung der Datenmenge pro Datensatz der Verwaltungsüberhang halbiert, allerdings hat dieser Überhang einen immer geringeren Anteil an der Datenmenge insgesamt.

Tatsächlich sind diese Überlegungen aber zur Zeit eher von theoretischer Natur. S-Record Dateien mit mehr als 128 Byte Dateninhalten lassen sich gar nicht an den RCX senden. Das mag an dem Werkzeug BricxCC liegen oder an der Firmware des RCX. Hier können schnelle Antworten der Leser langwierige Versuche überflüssig machen (bitte per Mail an VD@Forthev.de).

128 Byte Dateninhalte in einem S-Record führen aber, wie bereits zuvor angesprochen, zu rund 10% geringerer Übertragungszeit der Datei firm0309.lgo an den RCX. Das scheint mir für den Augenblick durchaus ein akzeptables Ergebnis zu sein. Erreicht habe ich dieses Ergebnis mit einem kleinen Programm, das ich mit dem HOLONForth (Holon86[3]) geschrieben habe. HOLON und der RCX gehören in meinen Augen einfach zusammen. NQC (Not Quite C) bringt den eifrigen RCX-Tüftler zwar deutlich weiter als die GUI-Programmiermodule von LEGO dies tun, aber die „große Freiheit" bringt auch dem RCX, selbstverständlich, nur Forth. Und weil mich am Holon nach wie vor die Übersichtlichkeit, Lesbarkeit und der Zwang zur Struktur faszinieren, ist Holon gerade in bezug auf den RCX für mich das „Forth-System der Wahl".

Grundlage für das Konvertierungsprogramm war die Beschäftigung mit den S-Records selbst.

Hierzu mussten, wie bereits ausgeführt, Informationen aus dem Internet beschafft werden. Die erhaltenen Informationen sind zum größten Teil sehr umfangreich, so daß ich mich in der nachfolgenden Beschreibung der S-Record Formate auf den für den RCX relevanten Teil beschränke. Gleichzeitig weise ich an einigen Stellen auf Aussagen hin, die in dieser Art in allen mir zugänglichen Veröffentlichungen zu finden sind, aber meinen eigenen Beobachtungen und Erfahrungen widersprechen.

Das S-Record Format

Eine S-Record Datei besteht aus einer Anzahl speziell formatierter ASCII Zeichenketten. Ein S-Record wird immer kleiner oder gleich 78 Bytes in seiner Gesamtlänge sein. (Hinweis: Das widerspricht den nachprüfbaren Erfahrungen mit 128 Byte Daten im Record. Plus der Verwaltungsbyte (ohne CRLF) ist der Record dann 138 Byte lang.)

Die Reihenfolge der S-Records innerhalb einer Datei ist völlig belanglos.

Type - ein Zeichenfeld (2 CHAR).
: Diese Zeichen beschreiben den Typ des Records (S0, S1, S2, S3, S5, S7, S8, oder S9). (Hinweis: LSI hat zusätzlich einen Typ S4 eingeführt, der bei Motorola nicht vorgesehen war.)

Count - ein Zeichenfeld (2 CHAR).
: Diese Zeichen geben als Zeichenpaar (!) in hexadezimaler Darstellung die Anzahl der verbleibenden Zeichenpaare (!) im Datensatz an. Beispiel: Count liest sich als “43“. Dann ist die Summe aus den Adresspaaren (4 CHAR = 2 Paare), aus den Data- oder Codepaaren und aus dem Paar für die Checksumme 43h, bzw. 67 dezimal. Das entspricht 134 Byte, von denen 6 Byte für die Adressangabe und die Checksumme abgezogen werden müssen. Der Datensatz enthält dann 128 Byte.

Adresse - ein Zeichenfeld (4, 6 oder 8 CHAR).
: Diese Zeichen geben zusammengefaßt in hexadezimaler Darstellung die Adresse vor, ab der die nachfolgenden Data- oder Codebyte im Speicher des RCX abgelegt werden sollen. Die Länge (Zeichenanzahl) dieses Feldes hängt von dem jeweiligen S-Typ ab. (Hinweis: für die Arbeit mit dem RCX, bzw. LEGO, werden nur Datensätze der Typen S0, S1 und S9 genutzt. Der Adressbereich ist darin immer 4 Zeichen groß.

Data - ein Zeichenfeld (0 – 64).
: (Hinweis: Die ursprüngliche Angabe von Motorola, ein S-Record könne maximal 78 Byte lang sein, basiert auf dieser Aussage über die maximale Datenlänge. Zuzüglich der Bytepaare für die Typ-Kennung, den Count, die Checksumme und maximal 4 Bytepaaren für die Adressangabe, können bis zu 14 Byte zu der Datenlänge dazukommen. Diese Aussage stimmt nicht

Generell sind S-Records wie folgt aufgebaut:

```
+------------------//------------------//-------------------------+
| Type | Count | Adresse  |            Data            | Checksumme |
+------------------//------------------//-------------------------+
```

überein mit den Erfahrungen zu 128 Byte langen Datensätzen für den RCX.) Die Datenbyte werden paarweise als hexadezimale Repräsentationen einzelner Daten- oder Codebyte aufgefasst.

Checksumme – ein Zeichenfeld (2 CHAR).
Dieses Zeichenpaar ist die Checksumme des Datensatzes. Die Checksumme wird als Summe aller binären Werte aller vorhergehenden Bytepaare des Datensatzes (außer der Typangabe, einschließlich Count und Adresse) ermittelt. Vom Einerkomplement dieser Summe wird das geringwertigere Byte als Checksumme verwendet.

Jeder Datensatz wird mit einem CRLF (DAh) beendet. Wenn irgendwelche zusätzlichen oder anderen Terminierungs- oder Verzögerungszeichen von einem Übertragungsprogramm verlangt werden sollten, dann ist Aufgabe des Übertragungsprogramms, dies selbstständig zu handhaben.

S0-Records – die Typenbezeichnung „S0“ liest sich im Hexeditor als „5330“. Das Adressfeld wird mit Nullen gefüllt. (Hex Zeichen in ASCII ! = 30303030). Die Header-Informationen im Datenfeld unterteilen sich wie folgt:

```
mName - 20 Zeichen, Modulname
Ver   -  2 Zeichen, Versionsnummer
Rev   -  2 Zeichen, Revisionsnummer
Descr.- 36 Zeichen, (0 bis 36)
                     Kommentar, Text
(Hinweis: firm0309 enthält im S0-Record
 lediglich den Text ‚?LIB_VERSION_L00,)
```

Für alle Subfelder gilt, daß sie aus ASCII Zeichen aufgebaut sind, die als Zeichenpaare jeweils ein Byte in hexadezimaler Darstellung repräsentieren.

S1-Records – die Typenbezeichnung „S1“ liest sich im Hexeditor als „5331“. Das Adressfeld wird als 2-Byte Adressangabe ausgewertet (4 Zeichen !). Im Datenfeld befinden sich Daten und/oder Code, die direkt im Speicher des RCX abgelegt werden können. Die Ablage in den Speicher geschieht ab der angegebenen Adresse.

S9-Records – die Typenbezeichnung „S9“ liest sich im Hexeditor als „5339“. Das Adressfeld enthält die Startadresse des auszuführenden Programms; ausgeführt als 2-Byte Adresse (4 Zeichen). Es gibt kein Daten- oder Codefeld.

Beispiel für eine Datei im typischen S-Record Format:

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

Die Datei besteht aus einem S0-Record, vier S1-Records und einem S9-Record.

Der S0-Record ist wie folgt aufgebaut:

S0 gibt an, das dies ein Header-Datensatz ist
06 sechs Zeichenpaare folgen
- 00 00 vier Zeichen als “leere” Adressangabe
- 48 44 52 “HDR”
- 1B die Checksumme

Der erste S1-Datensatz enthält nachfolgende Angaben:

S1 gibt an, das dies ein Record mit Daten und/oder Code ist
- 13 13h -> 19 Zeichenpaare folgen
- 00 00 die Daten/Codes werden ab der Speicheradresse 00 00 geladen
- cc cc 16 Zeichenpaare repräsentieren die binären Daten
- 2A die Checksumme

Der S9-Datensatz besteht aus:

S9 gibt an, das dies der letzte Datensatz der Datei ist. (Hinweis: damit relativiert sich die von Motorola gemacht Aussage, daß die Reihenfolge der Datensätze völlig unerheblich ist. Tatsächlich gilt diese Aussage für die Arbeit mit dem RCX ausschließlich im Bereich der S1-Datensätze. Zu diesen gibt die jeweils enthaltene Adressangabe vor, wo die Daten/Codes im Speicher abgelegt werden sollen. Wenn sichergestellt ist, daß es hier zu keinen Überschneidungen kommen kann, bzw. daß unbeabsichtigtes Überschreiben ausgeschlossen ist, dann ist die Reihenfolge der S1-Records belanglos.)
- 03 drei Zeichenpaare folgen
- 00 00 Startadresse des Codes, der ausgeführt werden soll
- FC die Checksumme

Die weiteren S-Record Typen habe ich hier bewußt nicht beschrieben, weil sie weder bei LEGO Verwendung finden, noch in den Dateien, die Ralph Hempel für sein pbForth erzeugt. Wenn sich die darin enthaltenen Möglichkeiten zu einem späteren Zeitpunkt für die Arbeit zwischen HOLON und dem RCX als wertvoll erweisen sollten, werde ich deren Beschreibung gerne nachholen.

Friederich Prinz

1 [http://sourceforge.net/projects/bricxcc/]
BricxCC ist eine Entwicklungsumgebung für die Programmierarbeit mit dem RCX. Die IDE arbeitet mit NQC, pbForth und anderen Entwicklungswerkzeugen zusammen und lässt sich nicht zuletzt hervorragend als Transmissionsprogramm nutzen. BricxCC ist kostenlos für alle privaten Nutzer verfügbar.

2) Verweis auf den COUNT
Der Count in den S-Records ist ein Byte groß und kann maximal den Wert 255 annehmen. Dem Countbyte selbst können in einem S–Record also maximal 255 Bytepaare folgen. Davon müssen drei Bytepaare von der „nutzbaren“ Länge abgezogen werden (2 Paare für die Adressangabe, 1 Paar für die Checksumme). Die verbleibenden 252 Bytepaare stehen dann für ein technisches Maximum von 504 Daten- oder Codebyte in einem Record vom Typ S1.

3) Holon86
Ist das Holon-Forth, das Wolf Weijgaard auf seiner Homepage allen privaten Nutzern kostenlos zur Verfügung stellt. Holon86 ist ein vollwertiges Holon-Forth für PC-Maschinen. Als Targets stehen dem Holon86 WinMonitore, separate PC oder die sogenannte Co-Routine zur Verfügung. Die Arbeit mit Holon am RCX geschieht auf der Basis der Co-Routine.

Bei der Durchsicht meines Berichtes zu den S-Records sind bei Wolf Wejgaard Fragen bezüglich der Klarheit einiger Begriffe aufgetaucht. Um inhaltlich ähnlichen Fragen der Leser gleich hier zu begegnen, füge ich einen Auszug der entsprechenden Mails an:

Frage:
> beim Durchlesen deines Doc's habe ich ein Problem mit
> Bytepaar und Zeichenpaar:
> ...
> Der Count gibt die Anzahl Bytes, nicht Bytepaare - dachte
> ich. Ein Byte ist im Text ein *Zeichenpaar*. Kann ein S1
> wirklich 504 Bytes enthalten?

Antwort:

Hier ein Beispiel aus der offiziellen firm0309.lgo von LEGO. Das ist das LEGO-System für die RCXe 1.0 und 1.1:

Der erste Datensatz, ein Record vom Typ S0

```
S01300003F4C49425F56455253494F4E5F4C303046
  ^^
```

Der Count 13h = dezimal 19 zählt ab einschließlich der Adresse bis einschließlich der Checksumme 19 Zeichenpaare. Das sind für mich, im hier abzulegenden Format, 38 Byte; weil 38 ASCII-Zeichen. Binär gesehen sind das natürlich

1. Adresse: 1 WORD,
2. Daten: 16 Byte,
3. Chksum: 1 Byte.

Daraus folgt, daß die maximal 504 technisch möglichen ZEICHEN für Daten 252 Daten-Byte entsprechen.

fep

Module: SrecTool

LEGO Dateien fuer den RCX (*.lgo) sind auf 32 Daten-/Codebyte in den S1-Records begrenzt. Theoretisch koennen bis zu 504 Daten-/Codebyte in einem solchen Record enthalten sein (siehe Dokumentation). Die mir heute zur Verfuegung stehende Software zum Uebertragen der *.log Dateien (BricxCC) ODER der RCX akzeptieren maximal 128 Byte pro Record.
Das reduziert den Datenueberhang, der stets auch den Sendeturm belastet, bereits ganz erheblich.

Group: Deklarationen

Pointer, Handles, Integers, Arrays, Puffer u.s.w.

```
Word: FilHnd
\ Handle fuer die Originaldatei und die noch zu erstellende
\ Datei.
0 INTEGER FilHnd

Word: MemHnd
\ Handle fuer den Zugriff auf den "Konvertierungsspeicher"
0 INTEGER MemHnd

Word: ^MemHnd
\ Zeiger in den "Konvertierungsspeicher"
0 INTEGER ^MemHnd

Word: ^MemHndKopie
0 INTEGER ^MemHndKopie

Word: #DataVorgabe
\ ACHTUNG ! Hier muss immer eine gerade Zahl stehen, weil
\ es sonst keine Daten-PAARE in den Records gibt !
128 INTEGER #DataVorgabe
\ 128 Byte ist die maximale Anzahl der Daten- oder Codebyte
\ in einem S-Record

Word: MemAdr
0 INTEGER MemAdr

Word: KopFil
\ Name der konvertierten SREC-Datei
" firm_fep.lgo" STRING KopFil

Word: OrgFil
\ Name der konvertierten SREC-Datei
" firm0309.lgo" STRING OrgFil

Word: TypS1
\ S-Record Typ -1-
Create TypS1 83 C, 49 C,
\ 53h 31h = "S1" - Daten- und Codetyp

Word: RecS0
\ Der Start-Datensatz kann maximal 74 Byte lang sein. Hier
\ gebe ich ein COUNT-Byte dazu...
CREATE RecS0 75 ALLOT
```

```
\ Bytegroesse 2 Rec-Type
\            2 Count
\            4 Adresse im Memory
\           64 Data | Code (0 bis 64)
\            2 Checksumme
\        ------
\           74 Byte in Summe
```

Word: RecS9

```
\ Der Start-Datensatz kann maximal 74 Byte lang sein. Hier
\ gebe ich ein COUNT-Byte dazu...
CREATE RecS9 75 ALLOT
\ Die Arrays RecS0 und RecS9 sollen die entsprechenden
\ Datensaetze der zu konvertierenden SREC-Datei aufnehmen.
```

Word: ChsDummy

```
0 INTEGER ChsDummy
```

Group: HilfsRoutinen

Word: hCh>Bin

```
\ Hex-CharacterToBin
\ wandelt ein Hex-Ziffernzeichen in ein binaeres Ziffernmuster
: hCh>Bin ( Chr -- b )
   48 - DUP 9 > IF 7 -
         THEN
  ;
\ Auf dem Stack wird ein gueltiges Ziffernzeichen zur
\ Zahlenbasis 16 erwartet [0..9 - A..F]. Das Ziffernzeichen liegt
\ als Zeichen aus dem Vorrat von ASCII vor.
```

Word: Bin>hCh

```
\ BinToHex-Character
\ wandelt eine Hex-Zahl in ein Hex-Ziffernzeichen
: Bin>hCh ( b -- Chr )
   48 + DUP 57 > IF 7 +
         THEN
  ;
\ Auf dem Stack wird eine gueltige Zahl zur Zahlenbasis 16
\ erwartet, aus dem Vorrat [0..15].
```

Word: 16e

```
\ liefert 16^n; mit n >0 und n <4
: 16e ( n -- 16en)
   1 SWAP 0 DO 16 *
        LOOP
  ;
```

Word: nBin>Num

```
\ wandelt Hex-Ziffern in Binaerzahlen; Anzahl
\ umzuwandelnder Ziffern wird vorgegeben;
\ Mindestanzahl ist 2
: nBin>Num ( Ziffern n -- n )
   1 DO SWAP I 16e * +
   LOOP
  ;
\ Aus 8 0 0 0 wird 8000h
\ Die Ziffern sind aus dem Hex-Vorrat [1..15], bzw [0..9-A..F]
```

Word: Num>nBin

```
\ wandelt Binaerzahl in Hex-Ziffern im Bin-Format um
: Num>nBin ( n n -- Ziffern )
   0 DO 16 /mod
   LOOP DROP
  ;
\ Aus 8000h wird 0 0 0 8
\ Die Ziffern sind aus dem Hex-Vorrat [1..15], bzw [0..9-A..F]
```

Word: MovFil

```
: MovFil ( n -- )
   FilHnd SWAP MOVEPOINTER
  ;
\ bewegt den Dateizeiger um "n" Zeichen in der mit FilHnd
\ angesprochenen Datei vor oder zurueck
```

Word: @Chr<Fil

```
: @Chr<Fil ( -- c )
   FilHnd C@H
  ;
```

Word: !Chr>Fil

```
: !Chr>Fil ( c -- )
   FilHnd C!H
  ;
```

Word: @Bin<Fil

```
: @Bin<Fil ( c -- b )
   @Chr<Fil hCh>Bin
  ;
```

Group: SRecRoutinen

Word: @Typ<File

```
\ Liest den aktuellen Record-Type aus der Datei ...
\ Dateizeiger MUSS auf das erste Byte des aktuellen
\ Datensatzes zeigen.
: @Typ<File ( -- n )
   FilHnd @H                  \ Typkennung auslesen
   FilHnd -2 MOVEPOINTER      \ Dateizeiger 2 Zeichen
  ;                           \ zurueck stellen
```

Word: Typ1?

```
: Typ1? ( -- f )
   @Typ<FIle
   TypS1 @ =
  ;
```

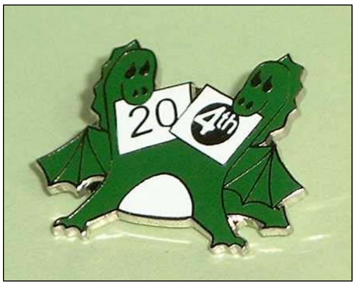
Word: @RecLen

```
\ Datensatzzeiger steht auf dem ersten Byte des aktuellen
\ Datensatzes.
: @RecLen ( -- n )
   2 MovFil                   \ Dateizeiger auf den Laengentext
    @Bin<Fil  @Bin<Fil        \ Zwei Hex-Ziffern aus der Datei
                              \ auslesen
    2 nBin>Num                \ in eine NUMBER umwandeln
    2*              \ Anzahl der "gezaehlten" Byte im Satz
    4 +             \ plus 2 Byte COUNT + 2 Byte TYPE
```

```
  -4 MovFil            \ Dateizeiger zurueck auf den Recordstart
 ;

Word: @DataLen
\ Die Anzahl der Datenbyte in einem S1-Record ist kleiner als
\ seine Gesamtlaenge in Byte
: @DataLen ( -- n )
   @RecLen 10 -
 ;
 \ Recordlaenge - 2 Typ
 \          - 2 Count
 \          - 4 Adresse im Memory
 \          - 2 Checksumme

Word: CrLf>Rec
\ CRLF werden an jeden Datensatz angehaengt
: CrLf>Rec ( -- )
   13 !Chr>Fil
   10 !Chr>Fil
 ;

Word: S1>Rec
: S1>Rec ( -- )
   83 !Chr>Fil
   49 !Chr>Fil
 ;

Word: ?Len
: ?Len ( -- n )
   ^MemHndKopie ^MemHnd - #DataVorgabe
   2DUP ( U> ) U< NOT IF NIP
        ELSE DROP
        THEN
 ;

Word: Len>Rec
: Len>Rec ( -- )
   ?Len     \ Anzahl der Datenbyte im S1-Record ist
            \ vorgegeben
   2/       \ Anzahl der entsprechenden Bytepaare
   3 +      \ plus 3 Bytepaare fuer Adresse (2) und ChkSum (1)
   2 Num>nBin        \ Zahl in Ziffern aufbrechen
   Bin>hCh !Chr>Fil  \ Bin-Ziffer in ASCII-Zeichen
                     \ umwandeln
   Bin>hCh !Chr>Fil  \ und in der KopDatei ablegen
 ;

Word: @MemAdr
\ @MemAdr erwartet, dass der Dateizeiger auf das erste Byte
\ im ersten Datensatz zeigt!
: @MemAdr ( -- )
   @RecLen         \ Recordlaenge
   2+              \ + CRLF = Start zweiter Datensatz
   4 + MovFil      \ + Type und Count = Start Adressangabe
   @Bin<Fil  @Bin<Fil  @Bin<Fil  @Bin<Fil
   4 nBin>Num IS MemAdr
   FilHnd FILERESET
 ;

Group: OrgDatLesen

Word: AllocMem
: AllocMem ( -- h|f )
   4096 MALLOC IF IS MemHnd
          THEN          \ 4096 Paragraphen a 16 Byte
 ;

Word: FreeMem
: FreeMem ( -- )
   MemHnd IF MemHnd MFREE  \ Wenn ein Handle
                           \ vergeben wurde, dann
      ( 0 ) IS MemHnd   \ wird der zugehoerige Speicher jetzt
      THEN              \ freigegeben und das Handle "geloescht"
 ;

Word: @StartRec
\ Dateizeiger zeigt auf das erste Byte des aktuellen Datensatzes;
\ hier auf das erste Byte des S0-Records
: @StartRec ( -- )
   @RecLen          \ Datensatzlaenge ist der
   DUP RecS0 C!     \ COUNT in der Kopie des Startrecords
   0 DO @Chr<Fil
      I 1+ RecS0 + C!
   LOOP
   2 MovFil       \ ueber CRLF hinweg auf den naechsten Satz
 ;
 \ Ab hier wird der Dateizeiger "weitergereicht"

Word: @EndRec
\ Dateizeiger zeigt auf das erste Byte des aktuellen Datensatzes;
\ hier auf das erste Byte des S9-Records
: @EndRec ( -- )
   @RecLen          \ Datensatzlaenge ist der
   DUP RecS9 C!     \ COUNT in der Kopie des Endrecords
   0 DO @Chr<Fil
      I 1+ RecS9 + C!
   LOOP
   2 MovFil       \ ueber CRLF hinweg auf den naechsten Satz
 ;

Word: @Data
: @Data ( -- )
   @DataLen       \ Anzahl der Datenbyte zum aktuellen Satz
   8 MovFil       \ Dateizeiger ueber Typ, Count u. Adr.
                  \ hinwegsetzen
   0 DO @Chr<Fil  \ Daten- oder Codebyte aus der Org.datei
      MemHnd ^MemHnd C!L  \ in den Speicher schreiben
      1 ADD ^MemHnd       \ Zeiger in den Speicher
                          \ inkrementieren
   LOOP
   4 MovFil       \ Ueber Checksum und CRLF hinweg auf den
 ;                \ naechsten Satz


Word: @S1Recs
\ Zum Start von @S1Recs steht der Dateizeiger von FilHnd auf
\ dem Start des zweiten Datensatzes.
```

```
: @S1Recs ( -- )
   BEGIN Typ1? IF @Data
              FALSE
          ELSE TRUE
          THEN
   UNTIL
   ^MemHnd IS ^MemHndKopie
 ;

Word: Init@OrgFilData
: Init@OrgFilData ( -- )
   CLEAR ^MemHnd    \ Zeiger in den Datenpool
   @MemAdr          \ Startadresse der Daten und Codes im H8
 ;

Word: @OrgFilData
: @OrgFilData ( -- )
   Init@OrgFilData    \ Initialisierungen
   @StartRec          \ StartRecord in Puffer kopieren
   @S1Recs            \ Daten ohne Struktur einlesen
   @EndRec            \ Endrecord in Puffer kopieren
 ;

Word: OrgFilOeffnen
: OrgFilOeffnen ( f -- h|f )
   OrgFil OPEN-R IS FilHnd
 ;

Word: OrgFilSchliessen
: OrgFilSchliessen ( -- )
   FilHnd IF FilHnd CLOSE IS FilHnd
       THEN
 ;

Word: OrgDat>Speicher
\ Was nicht NULL ist, gilt als WAHR
: OrgDat>Speicher ( -- f )
   AllocMem
   OrgFilOeffnen
   @OrgFilData
   OrgFilSchliessen
 ;

Group: KopFilAnlegen

Word: !S0|S9Rec
\ Auf dem Stapel liegt die Adresse eines Recordpuffers SO | S9.
\ Der Inhalt des Puffers wird in die KopDatei geschrieben
: !S0|S9Rec ( RecAdr -- )
    FilHnd SWAP Count 0 DO 2DUP I + C@ SWAP C!H
                 LOOP 2DROP ( Hnd Rec )
    CRLF>Rec
 ;

Word: !StartRec
: !StartRec ( -- )
   RecS0 !S0|S9Rec
 ;

Word: !EndRec

: !EndRec ( -- )
   RecS9 !S0|S9Rec
 ;

Word: Adr>Rec
: Adr>Rec ( -- )
   #DataVorgabe 2/ ADD MemAdr
   MemAdr 4 Num>nBin
   Bin>hCh !Chr>Fil
   Bin>hCh !Chr>Fil
   Bin>hCh !Chr>Fil
   Bin>hCh !Chr>Fil
 ;

Word: ChkSum>Rec
: ChkSum>Rec ( -- )
   0
   ChsDummy 6 + DUP NEGATE MovFil
        2/  0 DO @Bin<Fil @Bin<Fil
               2 nBin>Num +
            LOOP
   NOT 255 AND
   2 Num>nBin Bin>hCh !Chr>Fil Bin>hCh !Chr>Fil
 ;

Word: !RecKopf>Fil
: !RecKopf>Fil ( -- )
    S1>Rec     \ "S1"   = '5331'
    Len>Rec    \ Count
    Adr>Rec    \ Adresse der Daten/Codes im Speicher des H8
 ;

Word: !S1Recs
\ ChsDummy wird von ChkSum>Rec benoetigt, weil der Zeiger
\ in den Datentopf weiterbewegt wurde...
: !S1Recs ( -- )
   CLEAR ^MemHnd                          \ Start im DataBlock
   MemAdr #DataVorgabe 2/ - IS MemAdr  \ MemAdr auf
                                       \ ADD vorbereiten
   BEGIN !RecKopf>Fil       \ "S1"; COUNT; Adresse > Datei
    ?Len DUP IS ChsDummy
        0 DO MemHnd ^MemHnd C@L  !Chr>Fil
                                \ 1 Datebyte > Datei
           1 ADD ^MemHnd  \ Speicherzeiger inkr.
        LOOP ChkSum>Rec  CrLf>Rec
    ^MemHnd ^MemHndKopie =  \ solange nicht alle Daten
  UNTIL                     \ uebertragen sind
 ;

Word: KopFilAnlegen
\ Die neue Datei arbeitet mit dem gleichen Handle-Zeiger wie
\ vorher die Originaldatei
: KopFilAnlegen ( -- )
   KopFil MKFILE IS FilHnd
 ;
```

```
Word: KopFilAufbauen

: KopFilAufbauen ( -- f )
   KopFilanlegen  \ KopDatei erzeugen "firm_fep-lgo"
   !StartRec      \ S0-Record in die Datei schreiben
   !S1Recs        \ Alle Daten der S1-Records in die Datei
                  \ uebertragen
   !EndRec        \ S9-Record in die Datei schreiben
   FreeMem    \ DOS-Speicher freigeben; kommt immer gut...
 ;

Group: RecFilUmbauen

 Word: RecFilUmbauen
: RecFilUmbauen ( -- )
   OrgDat>Speicher      \ vor allem die Daten der S1-Records
                        \ lassen sich einfacher umstrukturieren,
                  \ wenn man sie aus einem grossen Topf
                  \ herausnehmen kann.
   KopFilAufbauen     \ Das ist dann die Arbeit dieses Wortes.
   OrgFilSchliessen    \ Das Handle ist gleich - da braucht man
 ;                     \ kein separates Wort.
```



Betreff: Beiträge für 2/2004
Von: Rafael_Deliano@t-online.de (Rafael Deliano)

Soweit der Redaktionsschluß nicht schon vorbei ist, könnte man die geneigten Leser noch kurz darauf hinweisen, daß es für das **FORTH** auf **68HC908GP32** jetzt auf

http://www.embeddedFORTH.de

ein vorläufiges Handbuch gibt.

MfG JRD

Hinweis zu den forthigen Andenken:

Tassen mit dem offiziellen Motiv „20 Jahre Forthgesellschaft“ können im Forthbüro erstanden werden. Solange der Vorrat reicht, gibt die FG diese Tassen gegen eine Spende von 5,- € zuzüglich Porto und Versandkosten ab.

Darüber hinausgehende Nachfragen nach Tassen, Tellern oder anderer Keramik mit dem SWAP richten Sie bitte direkt an Martin.Bitter@T-Online.de.

Die silberfarbenen SWAPs können, solange der Vorrat reicht, ebenfalls über das Forthbüro angefordert werden. Sie sollen bei Veranstaltungen wie dem Linux-Tag, bei lokalen Veranstaltungen zu Forth (Tag der offenen Tür, oder Ähnliches) oder wo immer sich Forth ins Gespräch bringen läßt, verteilt werden. Diese Anstecknadeln werden kostenlos abgegeben. Bitte teilen Sie bei entsprechenden Anforderungen dem Forthbüro mit, anläßlich welchen Ereignisses und in welcher Stückzahl Sie die Anstecker benötigen.

fep

Einen herzlichen Glückwunsch dem diesjährigen Preisträger!

Hochverdient und in Würdigung vieler Jahre Arbeit mit Forth und für die Forthgesellschaft wurde

Ewald Rieger

in diesem Jahr mit dem SWAP ausgezeichnet. Das 17. Mitglied des Drachenrates hat Thomas Beierlein, der Drachenträger des abgelaufenen Jahres, im Auftrag des SWAP in den Rat berufen. Herzlich Willkommen, Ewald!

fep

KI und Forth
Ein paar Überlegungen

Ulrich Paul

Hintergrund

Es gibt eine Reihe von Ansätzen für die Implementierung von künstlicher Intelligenz. Sie hier ausführlich oder gar sämtlich zu beschreiben, ist nicht Ziel dieses Artikels. Vielmehr sollen ein paar der prominentesten Vertreter herausgepickt und deren fundamentalen Einschränkungen aufgeführt werden.

■ Die wohl bekannteste Methode ist die der neuronalen Netze. Sie versucht das Ziel dadurch zu erreichen, daß sie die physikalische Struktur des Gehirnes auf Rechner abbildet. Aber sie unterliegt damit einem Trugschluß: Nicht das, was wie ein Auto aussieht, ist ein Auto, sondern das, was wie ein solches funktioniert. Es reicht nicht, ein Auto aus Holz detailgetreu nachzubauen, ohne die eigentliche Funktion der einzelnen Komponenten zu verstehen. Beim Fahrgestell und der Karosserie mag das vielleicht noch gehen, aber spätestens beim Antriebsstrang sieht jeder Laie sofort die Unmöglichkeit der Durchführbarkeit: Ein Motor aus Holz kann einfach nicht so funktionieren wie einer aus Metall. Was hier durch den augenfälligen Unterschied des Materials dargestellt wird, ist versteckt in dem Ansatz der neuronalen Netze enthalten: Die Regeln, nach denen sich so ein Netz im Gehirn aufgebaut hat und - das ist wichtig - ständig verändert. Alle mir bekannten Konzepte sehen nicht vor, daß bestehende Verbindungen abgebaut und neue aufgebaut werden - und zwar zu beinahe beliebigen anderen Neuronen. Solange das Problem nicht gelöst ist, wird dieser Ansatz wohl nur zu mehr Gewinn von Intel&Co führen. Der beste Beweis liegt wohl darin, daß es bis heute noch Menschen gibt, die einen Riesencomputer im Schach schlagen und das nicht bloß gelegentlich. Ein weiterer Nachteil dieses Ansatzes ist der, daß es wohl noch ein paar Jahrzehnte brauchen wird, bis wir die Anzahl der Neuronen eines Gehirnes als eigenständige Hardwaremodule auf Chips finden werden - wenn jemals einer diesen unsinnigen Schritt überhaupt unternimmt.
Die Betonung liegt hier auf der dynamisch änderbaren Rekonfiguration der Verbindungen, die eben leider nicht gegeben ist.

■ Die Hidden-Markov-Modelle haben zwar schon einiges an "Expertenwissen" in die Welt gebracht, aber sie leiden prinzipiell an den oben genannten Schwächen und weisen zusätzlich noch eine andere auf: Ihre Stufen sind hintereinandergeschaltet und es gibt praktisch keine effektive Rückführung vom Ausgang zum Eingang. Diese ist aber notwendig, um wirkungsvoll auf veränderte Bedingungen zu reagieren. Innerhalb ihres Teilgebietes sind sie sicherlich wirkungsvoll, aber eben auch nur in diesem, vom Konstrukteur vorgebenen, Rahmen. Aber eines haben sie den neuronalen Netzen voraus: ihre Abhängigkeiten sind in Software programmiert. Damit könnten - und der Konjunktiv ist hier wichtig, weil der Indikativ nicht gerechtfertigt ist - sie die Grundlage für KI bilden: Weil die "Verbindung" von einem Neuron zu einem anderen vollkommen in SW geschieht und keine dedizierte physikalische Verbindung voraussetzt. Auch sind die Neuronen nur Teile einer SW.
Obwohl sie recht dynamisch konfigurierbar sind, vermißt man doch eine effektive Rückkopplung. Es gibt - logischerweise - eine gewisse Rückführung, denn sie ist für das Training und die Adaption notwendig, aber sie ist bewußt eingeschränkt. Also nicht ganz das, was KI erstrebt.

■ Eigentlich nicht ganz hierzu gehört die Fuzzy-Logic. Aber sie vereint Elemente in sich, die einen Teilbereich der KI berühren: Entscheidungen zu fällen, wenn die Eingangsdaten nicht exakt sind. Sie vollbringt auf diesem Bereich Erstaunliches - in Anbetracht der Einfachheit des Konzeptes und der Implementierung. Betrachtet man dieses Konzept vom Standpunkt eines Analog-Ingenieurs, dann ist das eigentlich so ziemlich kalter Kaffee - aber obwohl alle "Analoge" von diesen Möglichkeiten geträumt haben (gut, eine Menge kann man recht einfach verwirklichen, aber andere sind kitzlig), erst durch Lofti Zadehs geschlossene Abhandlungen ist es uns möglich geworden, dieses diffiziele Problem mit Rechnern zu lösen. Und in SW ist inzwischen ja einiges lösbar, das wirtschaftlich in HW nicht realisierbar ist.

Was ist eigentlich Intelligenz?

Natürlich folgt hier keine definitive Erklärung/Definition. Im Sinne dieses Artikels sollen bloß ein paar Punkte hervorgehoben werden. Ganz gleich an welchem Maßstab man Intelligenz mißt, so setzt das Vorhandensein zwei wichtige Mechanismen voraus:

1.) Die Fähigkeit zur Ableitung von höheren Prinzipien aus bekannten Zusammenhängen (Induktion) und
2.) die Fähigkeit abstrakte Zusammenhänge auf konkrete Sachverhalte anzuwenden (Deduktion). Wobei es wichtig ist, daß ein und das selbe "Individuum" beide Schritte beherrscht und zwar nach Belieben.

Es muß also zuerst aufgrund der Erfahrung einen induktiven Schluß ziehen können, um das Ergebnis dann in einem deduktiven Schluß auf ein anstehendes Problem anzuwenden. Natürlich kann der übergeordnete Zusammenhang auch durch Lernen vermittelt werden, also auf dieser übergeordneten Ebene. Genau das passiert bei uns Menschen, wenn wir z.B. die Grammatik einer (Fremd-)Sprache lernen. Wir lernen nur die Zusammenhänge und an ein paar Beispielen die Anwendung, wobei letzeres i.d.R. auf die Unzulänglichkeit der Sprache an sich zurückzuführen ist.
Manchmal ist einfach einfacher etwas an einem - hoffentlich guten - Beispiel zu erklären als jemand die abstrakte Regel pauken zu lassen. Inwiefern dieser Aspekt in die KI rüberspielt, entzieht sich meiner Kenntnis, denn meines Wissens ist der Versuch noch nicht übernommen worden, daß ein intelligentes System sein Wissen - nicht seine Fähigkeiten! - auf ein anderes

transferiert. Denn das würde eben gerade die Kommunikation auf abstrakter Ebene voraussetzen.

Eines soll hier absolut klargestellt sein: Es gibt nicht "den" IQ! Zwar haben frühere Forscher versucht so etwas zu messen, aber sie sind alle gestrandet. Innerhalb einer bestimmten ethischen, ethnischen und kulturell homogenen Gesellschaft mag das Ergebnis vielleicht noch halbwegs aussagekräftig sein, aber wenn diese Bedingungen nicht erfüllt sind, dann ist das Ergebnis eines IQ-Tests blanker Unsinn. Und den häufigsten Fehler, den diese lieben Forscher gemacht haben ist, daß sie kulturell insensitiv vorgegangen sind. Sie haben die (meist) europäischen Denkweisen in den Vordergrund gestellt. Aber Intelligenz ist unabhängig von einer Kultur.

Sie ist die Fähigkeit, Probleme zu lösen. Aber wieweit gehen die Anforderungen dazu auseinander, wenn man verschiedene Bereiche betrachtet. Hier in Deutschland kann kein Mensch verdursten (verhungern vielleicht, aber nicht verdursten), denn jedes Gasthaus ist verpflichtet (per Gesetz) ihm (Trink-)Wasser kostenlos abzugeben. Abgesehen davon, daß sich wohl niemand weigert, jemandem etwas zu trinken zu geben.

Innerhalb eines kurzen Fußmarsches ist jeder in Deutschland also an einer Wasserquelle. Da braucht es keine Intelligenz, um zu überleben. Schauen wir uns dagegen das Leben in Zentralafrika an, dann sieht es komplett anders aus - und hier meine ich nicht das Leben in irgendwelchen Slums, sondern das freie Leben in der Savanne. Die Fähigkeit Wasser zu finden, ist hier überlebensnotwendig. Und wer das als "primitiv" abtut, den würde ich gerne einmal sehen, wenn er Tierspuren - selbst in unseren Gefilden - zu interpretieren hat. Nämlich nach den Gesichtspunkten, wann welches Tier diese Fährte gelegt hat und ob dieses Tier auf der Suche nach Wasser (oder sonst etwas) war oder sich einfach nur auf einer Wanderung befunden hat.

Um Gerechtigkeit walten zulassen, soll auch nicht verschwiegen werden, daß ein Buschmann ohne das kleine 1x1 in Deutschland genauso hoffnungslos verloren wäre wie ein Europäer in der Savanne. Aber es soll verdeutlichen, daß es keine einheitliche Skala eines IQ gibt. Die Umgebung bzw. die zu lösenden Probleme bestimmen den IQ.

Fähigkeiten, Wissen und Weisheit

Um der Natur der Intelligenz etwas näher zu kommen müssen wir diese drei Begriffe strikt auseinanderhalten. Fähigkeit beschreibt nur eine Eigenschaft etwas zu tun (aber nicht sie auch zu unterlassen!). Ich kann das Produkt zweier Zahlen im Kopf ausrechnen oder bei größeren Zahlen einen Taschenrechner zu Rate ziehen. Ich kann ein Loch an einer bestimmten Stelle unter bestimmten Umständen bohren. Das sind alles Tätigkeiten, die prinzipiell jederzeit von einer Maschine übernommen werden können - alles nur eine Frage der Programmierung. Die Entscheidungen, die zu fällen sind, sind klar umrissen und die erforderliche Aktion ebenso. Ja, auch wenn die Lochtiefe hier ein Parameter sein sollte, so ist doch klar, daß hier ein Loch einer gewissen Tiefe (wenn auch mit dem Wert Null) gebohrt werden muß. Wissen auf der anderen Seite ist die "Fähigkeit", Fähigkeiten gezielt einzusetzen. Wenn ich einen Stuhl bauen will, sagen wir einmal aus Holz, dann sagt mir mein Wissen wie ich ihn bauen kann (und vielleicht noch tausend andere Alternativen). Das Wissen stellt den Zusammenhang her zwischen meiner Konstruktion, dem gewählten Material und meinen Fähigkeiten. Letzeres ist wichtig: Das Wissen verwendet Fähigkeiten, um ein Ziel zu erreichen. Damit steht das Wissen ganz klar über allen Fähigkeiten. Und die Weisheit?

Die Weisheit ist leider nur spärlich erforscht, vor allem, weil es für sie (ebenfalls) keine absolute Norm gibt. Aber in meinen Augen ist eine brauchbare Definition: "Weisheit ist, beurteilen zu können, ob ein Wissen oder ein anderes Wissen besser angebracht ist", d.h. es abstrahiert eine Stufe mehr; ganz so wie der Übergang von Fähigkeit zu Wissen eine Abstraktionsstufe ist.

Was hat das mit Forth zu tun?

Eine Menge! Sicher, auch in anderen Sprachen gibt es so etwas wie DEFERs. Aber in Forth sind sie ein Teil des Konzepts und nicht nachträglich aufgesetzt. Das macht sie einfacher zu verstehen und zu verwenden - also mehr Forthler kennen das Konzept als z.B. C-ler. Und nur wenige andere Sprachen kennen überhaupt ein Konstrukt wie CREATE – DOES, wobei sich hier die Lücke rapide schließt, weil der Feind auch nicht schläft. Aber noch haben die Forthler hier einen historischen Vorsprung - nehmen wir ihn wahr!

Mit den DEFERs haben wir schon immer die Funktionalität ändern können. Geben wir nun den DEFERs eine variable Liste von Parametern und bauen wir noch eine Beschreibung ein, was welches Wort, das durch einen DEFER ausgeführt werden kann, implementiert, dann sind wir sehr nahe dran, was die Implementation von Fähigkeiten in der KI betrifft. Notwendig ist also vor allem eines: Eine "Beschreibung", was eine Routine (in Forth-Jargon: ein Wort) tatsächlich macht. Was für Parameter es verlangt (und was sie für einen Effekt haben) und eine "Beschreibung" der Effekten dieser Routine. Das Erstellen einer solchen Beschreibung ist sicherlich nicht trivial, aber Forth hat hier einen Vorteil: Es kann dem übergeordneten System "Fähigkeiten" mitteilen und muß sich nicht auf eine passive Beschreibung verlassen.

Um es weniger akademisch auszudrücken: Jedes Wort liefert bei seiner Installation ein Wort mit, das seine Kompilation übernimmt. Das geht in Forth ja absolut tivial: Das Installationswort wird einfach "IMMEDIATE" markiert und schon ist die Sache gegessen. Ein paar Checks muß das Installationswort schon machen, aber das ist nicht Thema dieses Artikels.

So, also die "Fähigkeiten" haben wir also damit abgehakt. Wie steht es mit dem Wissen? Auch da hat Forth gute Chancen. Aber eines muß vorher implementiert werden: ein FORGET, das nur Worte vergißt, die sich tatsächlich auf das gelöschte Wort beziehen. Wenn irgendwann ein Wort mittels CREATE-DOES eine Konstruktion erstellt hat, dann darf die nicht einfach gelöscht werden, nur weil ein Wort, das vorher definiert wurde, gelöscht wird. Und die CREATE-DOES-Konstruktion ist genau

Forth-Gruppen regional

Moers

Friederich Prinz
Tel.: (0 28 41) - 5 83 98 (p) (Q)
(Bitte den Anrufbeantworter nutzen!)
(Besucher: Bitte anmelden!)
Treffen: 2. und 4. Samstag im Monat
14:00 Uhr, **MALZ, Donaustraße 1**
47441 Moers

Mannheim

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

München

Jens Wilke
Tel.: (0 89) - 8 97 68 90
Treffen: jeden 4. Mittwoch im Monat
Ristorante Pizzeria Gran Sasso
Ebenauer Str. 1
80637 München

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)

Gruppengründungen, Kontakte

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

µP-Controller Verleih

Thomas Prinz
Tel.: **(0 62 71) - 28 30** (p)
micro@forth-ev.de

Forth-Hilfe für Ratsuchende

Jörg Plewe
Tel.: **(02 08) - 49 70 68** (p)

Jörg Staben
Tel.: **(0 21 03) - 24 06 09** (p)

Karl Schroer
Tel.: **(0 28 45) - 2 89 51** (p)

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
vlksFORTH
ultraFORTH
RTX / FG / Super8
KK-FORTH

Ingenieurbüro **Klaus Kohl**
Tel.: **(0 82 33) - 3 05 24** (p)
Fax : **(0 82 33) - 99 71**
mailorder@forth-ev.de

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 Forthgesellschaft und der VD finden Sie im Impressum des Heftes.

20 Jahre Forthgesellschaft

das, was wir brauchen, um "Wissen" zu implementieren, also die Verbindungen zwischen Neuronen herzustellen, sprich, Zusammenhänge zu erkennen. Um präzise zu sein, es sind CREATE-DOES-Konstruktionen, die CREATE-DOES-Konstruktionen kompilieren, um damit die dynamische Verbindung von Neuronen zu emulieren.

Warum geschachtelte CREATE-DOES?

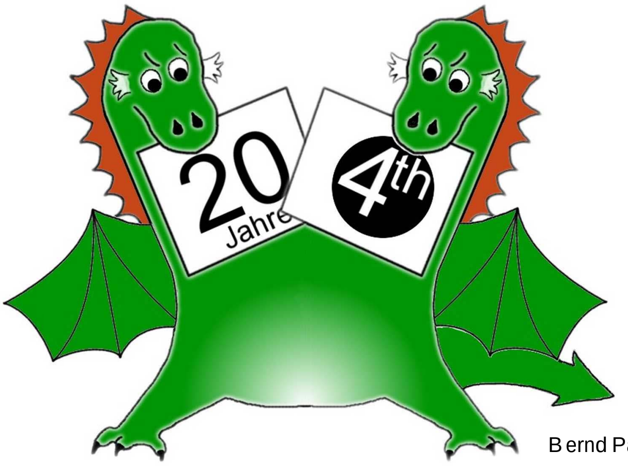
Es ist nicht ganz offensichtlich, warum nicht eine Ebene von CREATE-DOES ausreichend ist. Um es in das Bild zu passen, das eingangs entworfen wurde: Eine CREATE-DOES-Konstuktion kann ein Neuron repräsentieren, aber nicht seine variablen Verbindungen zu anderen Neuronen. Im menschlichen Gehirn ist die Anzahl der Neuronen wahrscheinlich bei der Geburt schon festgelegt. Das kann man aber derzeit von einem KI-System nicht behaupten - die erforderliche Anzahl ist einfach unbekannt. Also muß so ein System eine variable Anzahl von Neuronen adaptieren können.

Da nun jedes Neuron verschiedene Verbindungen zu anderen Neuronen haben kann, brauchen wir die zweite Ebene der CREATE-DOES-Konstruktion: Die, die übergeordnete Kommandos an die Endstellen (DEFERs) weiterleitet. Und vor allem: die, die - sollte besseres Wissen es notwendig machen - eben nicht dem alten Pfad folgen.

Fazit

Die Forth-Gemeinde sollte sich vorwärts orientieren und nicht ständig Erfolge der Vergangenheit als ihre Erfolge zitieren. Wenn dieser Artikel die träge Gemeinde nicht aufweckt, welcher will es dann?

Ulrich Paul



Bernd Paysan