

Phonetische Suche mit Allegro-Avanti

Vortrag gehalten beim Avanti-Workshop in der Friedrich-Ebert-Stiftung Bonn

Th. Berger

8.10.1997

Inhaltsverzeichnis

1	Einleitung	1
2	Ein interaktiver Client	2
3	Der Soundex-Algorithmus	6
4	Räuber's Schiller	7
5	Integration der neuen Funktionalität	12

1 Einleitung

Anstatt eine auch in der Praxis einsetzbare oder bereits eingesetzte phonetische Suche mit Avanti vorzuführen, hatte diese kurze Demonstration eher folgende Ziele:

- Aufbau eines kommandozeilenorientierten minimalistischen Benutzerinterfaces für Avanti
- Demonstration der Fähigkeiten von perl als *Skriptsprache*, also einer Sprache für das Zusammenbinden von vorhandenen Lösungen zu einer Applikation
- Ein Plädoyer dafür, daß perl nicht nur ein notwendig inkauf genommenes Bindeglied zwischen Avanti-Server und WWW-Anwendungen darstellt, sondern gerade beim Testen von Funktionalitäten und für quick-and-dirty Anwendungen auch als einziger Partner von Avanti ein interessantes und effizientes Werkzeug ist.

Unter diesen Zielen und zusammen mit dem Zufall(?), daß ein Modul `Soundex.pm` mit einer Implementierung des zwar extrem trivialen, aber verblüffend funktionsfähigen Soundex-Algorithmus von Knuth zur Standardbibliothek von perl gehört, kam es dann zu obigem Titel als selbstgestecktem Ziel der Demonstration.

Die im folgenden angeführten Codebeispiele stammen aus einer etwas elaborierteren Version der jeweiligen Programme, die ich zur Vorbereitung des Vortrags angefertigt hatte.

2 Ein interaktiver Client

Zunächst soll ausgehend von `tcp-ip.pl`, dem mit Avanti ausgelieferten Demonstrationsskript, welches, wie nicht genug zu betonen ist, auch zum Testen von Teilen einer Avanti-CGI-WWW-Installationen nicht zu unterschätzen ist, ein interaktives Benutzerinterface für Avanti geschaffen werden:

Der Benutzer gibt auf einer Zeile mehrere durch Spatien getrennte Suchbegriffe an, die dann durch implizites Und verknüpft im Titelregister der Datenbank recherchiert werden. Das Resultat wird dann mit einer Standardparameterdatei formatiert (hier: gefeldertes Format) dem Benutzer zurückgeliefert und ihm sofort der Prompt für die nächste Recherche angezeigt. Eingabe einer vollständig leeren Suchbegriffszeile beendet das Programm.

```
#!/perl -w
require Socket;
```

Hier einige vorgezogene Setzungen, um das Skript etwas flexibler als die Vorlage `tcp-ip.pl` zu halten:

```
$PfadParam="c:\\tmp";
$Datenbank="avdemo";
$user="opac";
$password="opac";
```

Gewisser Standard-Code aus `tcp-ip.pl` ist einfach in das Unterprogramm verlagert: Initialisierungen und das Herstellen der Socket-Verbindung.

```
&TCPConnect;
```

Es beginnt eine Endlosschleife...

```
endlessly:
{
    print "Suchbegriff: ";
    chop($query = <STDIN>);
```

...die durch leere Eingabe beendet werden kann:

```
    last unless $query;
```

Der virtuelle Aufrufpfad wird später in einem Unterprogramm übermittelt, hier wird jetzt nur der „eigentliche“ Job zunächst in der Textvariablen `$jobtext` zusammengebaut. `\n` bedeutet Zeilenvorschub.

```
$jobtext = "xport param E-W\n";
```

Die Eingabe wird an den Leerzeichen zerlegt und dann mit ‘`␣AND␣`’ verbunden. D.h. aus `abc def` wird der Find-Befehl `find TIT abc AND def` für Avanti.

```
@patterns = (split (/ /, $query));  
$jobtext .= "find TIT ".join (" AND ", @patterns)."\n";
```

Der Rest des Auftrags ist statisch und wird an `$jobtext` angehängt. Wir benutzen das vereinfachende Kommando `download set`, ich weiß aber nicht, ob der Sprung zu `noset` tatsächlich im Falle einer leeren Ergebnismenge ausgeführt wird.

```
$jobtext .= <<"XxX";  
download set  
if error jump noset  
jump ende  
  
:noset  
write "Leider Kein Resultat!" newline  
:ende  
XxX
```

Ein Unterprogramm, das den Job abschickt...

```
&launchjob();  
warn "\nwaiting for result...\n";
```

...und ein Unterprogramm, das das Ergebnis abwartet: Jede Zeile des Ergebnisses ist dann ein Element des Arrays `@result`

```
@result = &waitjob();
```

Wieder nur ein Demo: Damit der Schirm nicht überläuft, leiten wir das Suchergebnis in eine „Datei“, hinter der sich das MS-DOS Systemkommando `more` verbirgt:

```
open (PIPE, "|more");  
print PIPE @result;  
close (PIPE);
```

Trennen und Wiederholen der Schleife...

```
print "\n-----\n\n";  
redo endlessly;  
}
```

Jetzt das Schließen des Sockets: Wir haben (um Overhead zu vermeiden) die gesamte Zeit nur *einen* Socket benutzt. Das schafft uns maximal schnelle Beantwortung aufeinanderfolgender Jobs, hält aber natürlich auf dem Zielsystem eine Instanz des Avanti-Prozessmoduls `procav` am leben.¹

¹unter gewissen Webservern habe ich beobachtet, daß es Probleme gibt, mehrere Kommunikationsschritte mit Avanti über einunddenselben Socket laufen zu lassen. Evtl. ist es in der Praxis also sicherer, den Aufruf von `TCPCconnect` und das `close` in das Innere von Schleifen zu verlagern und den entsprechende Performance-Verlust in Kauf zu nehmen.

```

close (S);
die "Bye\n";
## ;-)

```

Hier das Unterprogramm zum Verbindungsaufbau: 4949 ist der (Standard-)Port von Avanti, localhost der Name dieser Maschine (immer!). Die Einbindung des Bibliotheksmoduls Socket.pm via use Socket ist hierhin geraten, sie ermöglicht vor allem den Zugriff auf die symbolischen Konstanten AF_INET etc.

```

sub TCPConnect {
#TCP-IP Verbindungen herstellen:
    $port = 4949;
    $them = 'localhost';
    use Socket;
    $sockaddr = 'S n a4 x8';

    $hostname = 'localhost';
    ($name, $aliases, $proto) = getprotobyname('tcp');
#    ($name, $aliases, $port) = getservbyname($port, 'tcp')
#        unless $port =~ /\d+$/;
    ($name, $aliases, $type, $len, $thisaddr) =
        gethostbyname($hostname);

```

Hier könnte eine geringfügige Verbesserung nicht schaden, damit auch bereits numerisch angegebene Internetadressen korrekt verarbeitet werden, d.h. der Aufruf der Funktion gethostbyname ist zu umgehen und mittels split und pack wäre der vierbyttige Wert für \$thataddr anders zu konstruieren.

```

    ($name, $aliases, $type, $len, $thataddr) =
        gethostbyname($them);
    $this = pack($sockaddr, AF_INET, 0, $thisaddr);
    $that = pack($sockaddr, AF_INET, $port, $thataddr);

```

Jetzt haben wir die Initialisierungsdaten beisammen und erzeugen den Socket (oder eine Fehlermeldung)

```

socket(S, PF_INET, SOCK_STREAM, $proto)
    || die "socket: $!";

```

Nach erfolgreicher Erzeugung wird sofort die Verbindung mit dem Server aufgenommen:

```

if(! connect(S, $that)){
    &ServerMustBeDown;
    exit;
}

```

Diese Anweisungen stellen sicher, daß maximal große Datenpakete an den Server übertragen werden (sonst erfolgt etwa nach jeder Zeile eine Übermittlung, was nicht besonders ökonomisch ist).

```
select(S); $| = 0; select(STDOUT);
```

Dies ist derzeit für den Windows-Server Avanti-W ein muß, für die UNIX-Variante aber verboten. Herr Hoepfner versprach baldige Vereinheitlichung.

```
print S "SendEndOfReply";  
}
```

```
### Auftrag abschicken  
sub launchjob {  
    local ($_);
```

Es wird der Text mit dem Job aus \$jobtext in seine Zeilen zerlegt, diese um Kommentare bereinigt, der Virtuelle Aufruffpfad davorgesetzt und Datenbankname, User und Password dahinter. Alles zusammen wird auf den Socket geschrieben, d.h. an den Server übermittelt.

```
local (@jobtext) = split (/\n/, $jobtext);  
print S "& $PfadParam\n";  
foreach (@jobtext) {  
    ($_) = split (/\s*\/\//, $_);  
    print S "$_\n";  
};  
print S "\@ DB=$Datenbank ID=$user/$passwd\n";  
print S "AVANTI:EOJ\n";
```

Hiermit wird kurzzeitig die oben eingestellte maximale Bufferung der Übertragung aufgehoben, man könnte die fünf Kommandos auch in ein Unterprogramm schreiben, das sollte dann flush heißen!

```
select(S); $| = 1; print S ""; $| = 0; select(STDOUT);  
}
```

```
### Auftragsergebnis einsammeln  
sub waitjob {
```

Solange eine Zeile über den Socket einlesbar ist, wird diese an das Array @Input angehängt.

```
local(@Input);  
while (<S>) {  
# bis das Ende-Signal der Antwort empfangen wird:  
# EOR (=End of Reply)  
    next if /^$/;
```

Hier unterscheiden sich Sockets etwas von „normalen“ Dateien: Wenn zu einem Zeitpunkt keine Daten anliegen, wissen wir nicht, ob nicht später weitere nachfolgen, da durch dieses Programm und Avanti zumindest theoretisch simultan gelesen und geschrieben werden kann. Würde Avanti von sich aus bereits ein Close oder Shutdown machen, würde dieses Programm das natürlich merken, Avanti tut dies aber nicht und erhält die Verbindung aufrecht, was wir in der allgemeinen Schleife im Hauptprogramm ja auch ausnutzen. Also teilt uns Avanti über den festen Text AVANTI: EOR (d.h. End of Result) mit, daß es das Übertragen abgeschlossen hat und nichts mehr kommt, wir also mit der abschließenden Verarbeitung der empfangenen Daten beginnen dürfen.

```

        last if /AVANTI: EOR/;
        push (@Input, $_);
    };
    return(@Input);
}

# Meldung, falls der Server unten ist
sub ServerMustBeDown {
    print "Der Server fuer die Datenbank $Datenbank scheint"
        . " nicht aktiv zu sein.\n",
        "Versuchen Sie es bitte später noch einmal\n";
}

```

Dies war jetzt also das kleine Programm, das Avanti ständig abfragt. Das Register für die Suchbegriffe ist natürlich fest eingestellt, zum Abfragen einer Datenbank ist es aber fast brauchbar (im Lesesaal würde man es aber vielleicht nicht einsetzen wollen;-)

3 Der Soundex-Algorithmus

Was man wissen muß, ist eigentlich nur, daß ein Modul dafür Bestandteil der Standard-Perl-Bibliothek ist. Mittels

```
use Text::Soundex;
```

am Anfang unseres Perlprogramms und dem Wissen, daß dieses Modul eine Funktion `soundex()` zur Verfügung stellt, die gleichermaßen für einzelne Worte wie für Arrays funktioniert. Es liefert also `soundex("Heilbronn")` den Wert H416.

Bevor wir nun im nächsten Abschnitt daran gehen, auch auf der Seite der Datenbank das Register PHO zu erzeugen, einige Worte zum Soundex-Algorithmus.

Der Kern des Moduls besteht aus folgenden wenigen Zeilen Code (Autor und Quelle bitte in `soundex.pm` nachsehen). Diese operieren auf `$_`, das ein einziges, bereits in Großbuchstaben umgewandeltes Wort enthält, zum Beispiel „Wolfenbuettel“
Zunächst den ersten Buchstaben merken, hier also „W“.

```
( $f ) = /^(.)/;
```

Jedem Buchstaben ist ein (amerikanischer!) Lautwert zugeordnet, Vokale (und ähnliches) bekommen den Wert 0, die anderen Buchstaben einen Wert zwischen 1 und 6, im Beispiel erhalten wir nun „0041051003304“

```
tr/AEHIOUWYBFPVCGJKQSXZDTLMNR/00000000111122222222334556/;
```

Der Lautwert des ersten Buchstabens kommt nach \$fc, also „0“

```
( $fc ) = /^(.)/;
```

Der Lautwert des ersten Buchstabens wird vom Anfang der in Codes umgewandelten Kette (evtl. mehrfach) entfernt, das ergibt im Beispiel dann „41051003304“

```
s/^$fc+//;
```

Jede Folge gleicher Lautwerte wird auf ein Vorkommen reduziert! Jetzt haben wir nur noch „410510304“.

```
tr//cs;
```

Jetzt werden noch alle Vokale ganz ausgeblendet, in unserem Beispiel bleibt „415134“

```
tr/0//d;
```

der erste Buchstabe im Klartext vorangestellt und sichergestellt, daß das Resultat mindestens vier Zeichen hat, hier nun „W415134000“

```
$_ = $f . $_ . '000';
```

und schließlich das Ergebnis auf die ersten vier Zeichen (d.i. ein Buchstabe gefolgt von drei Ziffern) reduziert, das Ergebnis im Beispiel ist dann „W415“;

```
s/^(.{4}).*/$1/;
```

Wir sehen also, daß dieser Algorithmus die gesamte Sprache auf $26 * 6 * 6 * 6 = 5616$ mögliche Soundex-Codes abbildet und daher nicht besonders gut sein wird. Durch eine bessere (aus Deutsche zugeschnittene?) Lauttabelle und eine Verlängerung der Codes kann er durchaus noch verbessert werden, es handelt sich aber prinzipiell eigentlich nur um ein instruktives Beispiel aus Knuths „Art of computer programming“, viele bessere Algorithmen sind seitdem entwickelt worden!

4 Räuber's Schiller

Für die geplante phonetische Suche mittels Soundex-Codes müssen wir nun diese Codes auf irgendeine Weise in die Register der Datenbank bringen, um sie dort finden zu können. Prinzipiell haben wir nun viele Möglichkeiten:

1. Wir bringen sie garnicht in die Datenbank, sondern machen einen Volltextexport der Datenbank, bilden dann extern von jedem Wort das Soundex-Äquivalent und zeigen im Fall von Treffern den Datensatz an. Dies ist natürlich extrem unökonomisch und hat zudem nichts mit Avanti zu tun.

2. Wir lassen uns von Avanti zu jedem zu suchenden Wort einen Registerausschnitt des Anfangsbuchstabens liefern und bilden für jedes im Registerausschnitt vorkommende Wort im Perlskript das Soundex-Äquivalent, um vergleichen zu können.
3. Wir bilden den Soundex-Algorithmus mit Mitteln der Exportsprache nach und integrieren ihn zwecks Produktion zusätzlicher Schlüssel in die Indexparameterdatei. Dies ist in diesem einfachen Fall bestimmt möglich.
4. (Spekulativ) Wir programmieren den Soundex-Algorithmus in C nach und integrieren ihn über die (längst vergessene) C-Schnittstelle in INDEX.EXE.
5. Wir programmieren den Soundex-Algorithmus in C nach und integrieren ihn über die C-Schnittstelle in PRESTO.EXE: Bei der Eingabe einer Kategorie werden die Soundex-Äquivalente aller Kategorien errechnet und in eine neue Zusatzkategorie gelegt.
6. Wir machen einen Volltextexport und legen die Soundex-Äquivalente aller Worte des Datensatzes in eine Zusatzkategorie, die wir mit Update wieder in die Datenbank zurückmischen. Diese Zusatzkategorie wird verstichwortet.
7. Wir machen einen Volltextexport der vorhandenen Datenbank und produzieren aus den Soundex-Äquivalenten der Worte in den Datensätzen Dateien `ii...`, die wir dann mit QRIX.EXE in den Index integrieren. Dieser Methode gebe ich in der Praxis den Vorzug, da sie auch für Volltextindexierungen (der Abstracts in Katalogisaten oder digitaler Volltexte der katalogisierten Werke) gut geeignet ist.
8. Wir machen einen Volltextexport der vorhandenen Datenbank und erzeugen neue, zusätzliche Datensätze mit den Soundex-Äquivalenten, die wir dazuindexieren.

Die ersten beiden Möglichkeiten passen nicht auf unseren oben programmierten Client, die letzten zwei, obwohl die einzig realistischen meiner Meinung nach, haben den Nachteil, daß die produzierten Soundex-Äquivalente bei einer Änderung der zugrundeliegenden Allegro-Datensätze nicht mit aktualisiert werden. Die Vorgehensweise ist also ersteinmal nur für statische Datenbanken geeignet, die nicht interaktiv weiterbearbeitet werden können. Die mittleren (also bis auf die ersten und letzten beiden) haben nicht nur den Nachteil der geringen Verbreitung der C-Schnittstelle, sondern sie fallen auch stärker unter die Einschränkung, daß ein Allegro-Datensatz nur 500 Schlüssel produzieren darf.

Nur aus Gründen der Beispielhaftigkeit und vor allem der etwas unüblichen Nutzung der neuen Schiller-Räuber-Funktionalität verfolgen wir den letzten der aufgezählten Wege:

Zunächst also die Exportdatei, wir nennen sie `E-PHO.APR`, sie kann ihre Herknunft von den Standardexporten `I-1.APR` oder `E-1.APR` nicht verleugnen:

```
zl=0      Zeilenlänge unbegrenzt (kein Umbruch)
ks=4      Beginn des Ausgabertextes nach der Kategorienummer
```

```

ke=" "  Kategorie-Ende = Code 0
as=""  Aufnahme-Start: Hierarchiekennung + 0
      Hauptaufnahme      : Code 01 als Startzeichen
      Unteraufnahme Stufe 1:      02 ...
ae=13 10 Aufnahme-Ende: Carriage Return / Line Feed
      bei hierarchischer Aufnahme: nur am Ende
      ---- Anweisungsteil -----
i4=1   Stammsatzersetzungen machen!

/0..   Viel (codiertes, Nummern, Signaturen) ausblenden
/30.
/32.
/71
/76
/77
/87.
/88.
/89.
/90.
/91.

      Stammsätze ausblenden
#3n +# Z #zz 0
#4n +# Z #zz 0
#6n +# Z #zz 0
#8n +# Z #zz 0

#hi +#98z Z #zz 0      % hierarchieuebergreifend!
#t{h0}
#00 y0 p"00 y" #zz 0  % Originalidentnummer mit 'y' davor
                    % nach #00
#t{0 "96 "}          % Alle Worte (auch von hierarchisch
                    % gespeicherten Unterbänden)
                    % in *eine* #96 bringen

#98z
##      Pauschalexport: alle Kategorien hintereinander
      ausgeben, jeweils mit ke abschließen
##
      Ein paar (viel zu grobe) Zeichenumsetzungen.
p !/@ =32      fast alle Sonderzeichen (und Ziffern!)
      als Worttrenner nehmen!

p _ 32
p [ 1

```

```

p l 1
p ¬ 1

p ä "ae"
p ö "oe"
p ü "ue"
p ß "ss"
p Ä "Ae"
p Ö "Oe"
p Ü "Ue"

```

Export mit diesen Parametern wird uns also eine Menge von Sätzen liefern, die aus Kategorien #00 mit y-Identnummern und #96 mit Stichworten bestehen. Dies können wir nun hervorragend mit folgendem Perlskript nachbearbeiten:

```

#!perl
use Text::Soundex;
#$soundex_nocode = 'Z000';

```

Für jeden Datensatz...

```
while ( <> ) {
```

Zeilenende (Satzende) abschneiden

```
    chop;
```

Mehrfachleerzeichen (davon hat uns der Export Massen erzeugt!) zusammenfassen

```
    s/\s+/ /g;           # mehrfachleerzeichen
```

Leerzeilen kann es (Stammsätze etc.) auch geben, diese müssen ausgeblendet werden:

```
    next unless $_;
```

Das Einfügen der vier Zeichen x gibt uns Daten im .ALD-Format, der `split()` liefert uns einen Teildatensatz „vor“ #96 und „danach“.

```
    s/^\x01/\x01xxxx/;      # mache ald!
    ($id, $words) = split (/^\x0096 /);
```

Der Anfang des Datensatzes wird ausgegeben:

```
    print $id;
```

Zerlegen in Worte...

```
    @words = split(//, $words);
```

...umwandeln in Soundex-Äquivalente...

```
@new = soundex (@words);
```

... und wiederzusammensetzen zu einem Text.

```
$new = join (" ", @new);
```

vorige drei Zeilen kürzer:

```
# $new = join (" ", soundex(split(/ /, $words)));
```

Ausgabe des restlichen Teils des Datensatzes:

```
print "\x0096 $new\x00\n";  
}
```

Fertig.

Was nun noch fehlt, ist ein bisschen Code für die Indexparameter.

Zum einen müssen die in die neue Kategorie #96 gebrachten Sound-Äquivalente indexiert werden, weil wir nur mit AVANTI über symbolische Registernamen zugreifen, nehmen wir den Bereich 'PH_' im Register 7. Die abschließende „I“-Zeile manifestiert den symbolischen Registernamen PHO.

```
ak=96 " "+Ç
```

```
#-Ç
```

```
#u1 y0 p" | 7PH "          Phonetik-Eintrag
```

```
#+#
```

```
I PHO 7PH.
```

Zum anderen müssen ja auch die neuen Datensätze mit den phonetischen Äquivalenten in Beziehung zu den Originalsätzen gesetzt werden! Wenn wir uns an die Allegro-Lösung für das Schiller-Räuber-Problem erinnern, so fällt uns auf, daß diese Lösung asymmetrisch ist: „Schiller“ ist der in der *Hauptaufnahme* enthaltene Verfasser, „Räuber“ ein in *einer Unteraufnahme* enthaltenes Titelstichwort. Die sogenannte Plus-Suche erweitert nun den Treffer zu „Schiller“ um alle Unteraufnahmen, so daß Kombination mit den Treffern zu „Räuber“ zu einem gemeinsamen Treffer, nämlich dem Band, führt. Über Avanti werden wir bei unserer Suche nach den phonetischen Codes zunächst einmal nur die Datensätze mit diesen finden, die Ausweitung & im Sinne der Plus-Suche muß uns dann die eigentlichen Aufnahmen liefern. Insofern ist klar, daß wir die frisch generierten Sätze mit der phonetischen Information in #96 als *Hauptsätze* der schon längst vorhandenen Titelsätze auffassen müssen. Für die technische Realisierung müssen wir die Formulierung umkehren: Die Titelsätze deklarieren sich als Untersätze der zukünftigen phonetischen Sätze, diese werden aber dieselbe Identnummer nur mit vorangestelltem 'y' bekommen, wir haben also kein Problem, folgende Zeilen in die .API zu integrieren (y-Sätze selbst sollen dabei natürlich nicht Untersätze von nie existierenden yy-Sätzen werden):

ak=00+ü

i7=9

```
#-ü                               SR-Zusatzeintrag (falsch herum!)
#u1 y0 I4,y p"y" X9
#+#
```

Anders als im Beispiel von „Schiller“ und „Räuber“ werden unsere phonetische Suchen immer nur „Hauptsatz“-begriffe suchen und miteinander verknüpfen; die Ausweitung wird uns also stets, da nie durch „Untersatz“-begriffe eingeschränkt, als Resultat phonetische Sätze *und* die Titelsätze liefern. Hier wird also noch gefiltert werden müssen. Aus Gründen der Ökonomie des Beispiels verlegen wir dieses Filtern nicht in die Exportparameterdatei, sondern modifizieren gleich den avanti-Job.

5 Integration der neuen Funktionalität

Zunächst eine kleine Stapeldatei, um die Datenbank zu generieren:

```
: phony.bat: Produktion der "phonetischen" Zusatzdateien
:1. allgemeine Setzungen
set -k=A
set -L=ger
set -d=d:\avanti-w\avdemo
set -P=d:\coop\a-prg
set -b=cat
:2. Isolation der Worte mit Allegro
%-P%\srch -f6 -ee-pho/%-b%_99.raw -d%-d%\%-b%_1 -m0
:3. Ersetzen der Worte durch die phonetischen Aequivalente
perl phony.pl<%-b%_99.raw>%-D%\%-b%_99.ald
rem del %-b%_99.raw
:4. Neuaufbau der Datenbank
%-P%\index -f70 -@1 -e%-b%/%-d% -d*%-d%\%-b%_ -n0 -m0 -v0 -h0
%-P%\index -fil -@2 -e%-b%/%-d% -d*%-d%\%-b%_ -n0 -m0 -v0 -h0
```

Jetzt fehlt als Abschluß noch der Umbau des ursprünglichen Perlskripts in folgenden Punkten:

1. Berechnen des phonetischen Codes der Suchbegriffe
2. Umstellung des Suchkommandos für Avanti auf das neue Register PHO und &-Kommandos
3. Herausfiltern der phonetischen Sätze aus den von Avanti gelieferten Resultaten.

Dazu gehen wir wie folgt vor:

1. Wir ergänzen am Anfang des Skripts eine Zeile

```
use Text::Soundex;
```

2. Wir ersetzen im Skript die Zuweisung an @patterns durch

```
@patterns = soundex (split (/ /, $query));
```

und einige Zeilen weiter die Konstruktion des Avanti-Find-Befehls durch:

```
$jobtext .= "f PHO &".join (" AND &", @patterns)."\n";
```

Zu beachten ist hierbei nicht nur das geänderte Register (TIT zu PHO) sondern auch die an zwei Stellen ergänzten &, die Avanti mitteilen, daß die Suche im Sinne der Schiller-Räuber-Problematik automatisch auszuweiten ist. Die zu suchenden Begriffe A123 und B456 erfordern somit die Suchanweisung f PHO &A123 AND &B456.

3. Das Filtern der #96er-Sätze aus dem Resultat erfordert, daß wir in der Formulierung des Avanti-Jobs die kurze Form download set durch eine explizit programmierte Schleife ersetzen:

```
$jobtext .= <<"XxX";
get first
if error jump noset
:loop
if #96 jump next
download
:next
get next
if ok jump loop
```

Damit sind die Modifikationen am Client beendet.