Kurze Einführung in die `KASH3`-Syntax

Vorlesung Kodierungstheorie Sommersemester 2006

Im folgendem soll eine kurze Einführung in die `KASH3`-Syntax gegeben werden. Dies geschieht anhand von Beispielen, an denen die Befehle und zu beachtende Dinge erläutert werden.

1 Erste Schritte in `KASH3`

`KASH3` ist ein Interpreter um sich der `KANT V4`-C-Bibliotheken zu bedienen. `KANT V4` ist ein auf algebraische Zahlentheorie spezialisiertes Computeralgebra-System, was an der TU Berlin von der `KANT`-Gruppe entwickelt wurde.
`KASH3` startet man, indem in die Shell der Befehl `kash3` eingeben wird. Es erscheint dann ein Banner und ein Eingabeprompt `kash`. Verlassen kann man `KASH3` wieder mit `kash> quit;`. Nach dem Aufruf von `KASH3` gelangt man in einen Interpreter, d.h. es können Befehle eingegeben und sofort ausgeführt werden. `KASH3`-Programme werden mit `Read("Dateiname");` eingelesen. Mit dem Befehl `LogTo("Dateiname");` wird alles, was nach diesem Befehl ausgegeben wird, genauso in der Datei "Dateiname" gespeichert. Mit Eingabe von `LogTo();` wird dieser Vorgang beendet. Führt man speicherintensive Berechnungen in `KASH3` durch, so kann man `KASH3` mit `kash3 -m n` aufrufen, wobei `n` die Anzahl der Bytes ist, die für `KASH` reserviert werden sollen.

Die Syntax von `KASH3` ist der von der Programmiersprache Pascal teilweise recht ähnlich. Eine erste Besonderheit ist, daß `KASH3` sowohl imperative als auch funktionale Elemente enthält. Wir werden dies später an geeigneten Beispielen sehen.

Folgende Wörter oder Befehle sind in `KASH3` reserviert und können nicht als Variablen benutzt werden:

`and`, `do`, `else`, `end`, `fi`, `for`, `function`, `if`, `in`, `local`, `mod`, `not`, `od`, `or`, `repeat`, `return`, `then`, `until`, `while`, `quit`.

Ansonsten kann jedes Wort als Variable benutzt werden, wie z.B.

```
kash> text := "Hello world";
Hello world
```
oder auch
```
kash> a := 3;
3,
```

wobei := der Zuweisungsoperator ist. Folgt auf einer Eingabe ein Semikolon, so versucht `KASH3` den Befehl auszuführen. Ein wichtiges Merkmal von `KASH3` ist, daß man kein Typmanagement benötigt. Um ein kleines Beispiel zu geben, wird erst der Begriff Liste in `KASH` erklärt. Eine Liste in `KASH3` ist eine endliche angeordnetes Tupel von beliebigen Elementen. Ein einfaches Beispiel ist

```
kash> L := [1..10000];
[1..10000].
```

Hier ist `L` eine Liste, die aus den ersten 10000 ganzen Zahlen besteht. Man kann durch `L[k]` auf das k-te Element von L zugreifen, sofern die Liste L auch soviel Element enthält:

```
kash> L[312];
312.
```

Ein anderes Beispiel für eine Liste ist

```
K := [1, "HALLO", ZX, i-> i*2, , [] ];
[ 1, "HALLO", Univariate Polynomial Ring in X over Integer Ring ,
function ( i ) ... end, , [] ].
```

Hier haben wir unterschiedliche Objekte in der Liste: eine ganze Zahl, ein ASCII-String, ein Polynomring,eine Funktion, eine Leerstelle in der Liste L und eine leere Liste. Möchte man den Polynomring aus der Liste K lesen, so schreibt man

```
kash> PolyRing := K[3];
Univariate Polynomial Ring in x over Integer Ring
```

und KASH3 hat dann den Polynomring in der Liste L unter der Variable PolyRing gespeichert. Mit einer Definition einer Variablen ist also der Typ automatisch festgelegt. Listen sind jedoch keine Mengen: So erhält man durch Eingabe von

```
kash> L := [3,3,3,4,4,5,6];
[ 3, 3, 3, 4, 4, 5, 6]
```

eine Liste mit Elementen, die mehrfach auftauchen. KASH kennt aber auch Mengen im mathematischen Sinne. Der Befehl hierfür ist Set. Für obiges Beispiel ergibt sich dann

```
kash> L := Set([3,3,3,4,4,5,6]);
[ 3, 4, 5, 6].
```

Möchte man wissen, ob das Element 5 in der Menge L liegt, so gibt man

```
kash> 5 in L;
```

ein und erhält

```
TRUE.
```

Als nächstes wollen wir eine Funktion in KASH definieren. Das folgende Beispiel soll eine Liste von Zahlen in eine Liste von Buchstaben eines vorgegebenen Alphabetes umwandeln. Die Buchstaben der Worte kommen aus einem Alphabet A, das wir als Liste definieren:

```
kash> A := ['A','B','C','D','E','F','G','H','I','J','K','L','M',
            'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'];
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

Um zu sehen wie man Funktionen definiert soll folgendes Beispiel dienen:

```
NumbersToText := function(L,alphabet)
   return(List(L, i-> alphabet[i+1]));
end;
```

Der Funktionsname ist `NumbersToText` und als Parameter erhält die Funktion eine Liste `L` von ganzen Zahlen und eine Alphabet `alphabet`. Als nächstes kommt eine sogenannte Listenfunktion. Der Befehl `List` verlangt eine Liste und eine Funktion, hier also `L` und `i -> alphabet[i+1]`. Die Funktion `i -> alphabet[i+1]` bildet eine Zahl `i` auf das `i+1`-te Element im Alphabet ab. Sinnvollerweise sollten also die Einträge der Liste `L` nur um eins verminderte Positionen von Elementen des Alphabets `alphabet` sein. Der Befehl `List` geht alle Elemente der Liste `L` durch, bildet sie bezüglich der vorgegebenen Abbildung ab und gibt eine neue Liste zurück. Diese besteht dann aus den Bildern von `i-> alphabet[i+1]`. Dabei wird die Reihenfolge eingehalten, d.h. in der zurückgegebenen Liste ist das `i`-te Element das Bild vom `i`-ten Element der Liste `L`. Schließlich wird die so erhaltene Liste mit `return` zurückgegeben. Das bedeutet, wenn man den Aufruf und die Zuordnung

```
kash> text := NumbersToText([7,0,11,11,14], A);
```

macht, so ordnet `KASH` der Variable `text` das Ergebnis des Aufrufes der Funktions `NumbersToText` zu, nämlich den Wert

```
kash> text;
"HALLO"
```

`KASH` unterscheidet zwischen globalen und lokalen Variablen. Das nächste Beispiel soll deutlich machen, was eine lokale und was eine globale Variable ist:

```
Vigenere := function(block, key,alphabet)
local S,B,Cipher;
   S := TextToNumbers(key,alphabet);
   B := TextToNumbers(block, alphabet);
   Cipher := NumbersToText(List([1..Length(S)],
             i-> (S[i]+B[i]) mod Length(alphabet)), alphabet);
   return(Cipher);
end;
```

Zunächst haben wir wieder einen Funktionsnamen `Vigenere`. Die Funktion verlangt die Parameter `Block, key` und `alphabet`. Als lokale Variablen werden nun `S,B` und `Cipher` gewählt. Dies macht man durch den Befehl `local` deutlich, dem eine Aufzählund lokaler Variablen folgt. Das bedeutet, daß diese Variablen ihre Gültigkeit nur innerhalb der Funktion `Vigenere` haben. Gibt man z.B. folgendes in die Shell ein:

```
kash> S;
```

so gibt `KASH`

```
Error, Variable: 'S' must have a value
```

zurück. Wurde aber vorher

```
kash> S := 3;
```

eingegeben, so würde man dann

```
kash> S;
3
```

erhalten. Global heißt eine Variable, wenn sie wie oben definiert wurde, also nicht innerhalb einer Funktion. Innerhalb der Funktion `Vigenere` besitzt die Variable `S` den innerhalb der Funktion zugewiesenen Wert, also hier `S := TextToNumbers(key,alphabet)`. Beim Verlassen der Funktion hat `S` wieder seinen alten Wert `3`. Würde `S` nicht in der Aufzählung der lokalen Variablen auftauchen, so hat `S` auch innerhalb der Funktion den selben Wert wie außerhalb. Wie man sieht können auch innerhalb einer Funktion wieder Funktionen aufgerufen werden. Allerdings müssen diese vorher definiert werden.

2 Funktional versus imperativ

Als nächstes werden wir einige weitere Operatoren auf Listen kennenlernen. Gleichzeitig wird immer eine imperative Variante angegeben.

Der Operator `Filtered` verlangt als Parameter eine Liste `L` und eine boolsche Funktion, z.B. die Funktion `i -> IsPrime(i)`, wobei `IsPrime(n) = false` ist wenn `n` keine Primzahl war, ansonsten wird `true` zurückgegeben. Filtered gibt eine neue Liste zurück, die aus allen Elementen `k` der Liste `L` besteht, für die `IsPrime(k)=true` gilt.

```
kash> Filtered([1..50], i-> IsPrime(i));
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47 ].
```

Wir erhalten also eine Liste der Primzahlen $n \leq 50$. Die imperative Variante wäre hier

```
kash> for xx in [1.. 50] do
    if IsPrime(xx) then
       Print(xx," ");
    fi;
 od;
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

Möchte man nun wissen, ob in der Menge `[2342..2444]` sich irgendeine Primzahl befindet, so gibt man folgendes ein:

```
kash> ForAny([2342..2444], i-> IsPrime(i));
true.
```

```
kash> xx :=2341; b := false;
2341
false
kash> repeat
   xx := xx+1;
   b  := IsPrime(xx);
   until(b or xx > 2443);
kash> b;
true
```

Wir wissen also jetzt, daß sich in der Liste `[2342..2444]` mindestens eine Primzahl befindet. Die kleinste bekommen wir mit

```
kash> First([2342..2444], i-> IsPrime(i));
2347
```

oder

```
kash> repeat
  xx := xx+1;
  b  := IsPrime(xx);
  until(b or xx > 2444);
kash> xx;
2347
```

Als nächstes möchten wir wissen, ob in der Liste `K := [2,4,10,12,18,20,22]` nur gerade Zahlen vorkommen, also

```
kash> L := [2,4,10,12,18,20,22];
[ 2, 4, 10, 12, 18, 20, 22 ]
kash> ForAll([2,4,10,12,18,20,22], j-> j mod 2 =0);
true

kash> while xx <= Length(L) do
   b := (L[xx] mod 2 =0);
   xx := xx+1;
 od;
kash> b;
true
```

Eine weitere wichtige Listenoperation ist `ListSplit(L,n)`, welche eine gegebene Liste `L` in Teillisten der Länge `n` aufteilt. Ist nun `n` kein Teiler der Listenlänge `Length(L)` und gilt `Length(L)=qn + r` mit $|$`r`$|$ `<` `n` und `q` $\in \mathbb{N}$, so gibt `KASH` `q` Listen der Länge `n` und eine Liste der Länge `r` zurück:

```
kash> A1 := "1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ)(,.=!?ab
                                     cdefghijklmnopqrstuvwxyz ";;
kash> ListSplit(A1,15);
[ "1234567890ABCDE", "FGHIJKLMNOPQRST", "UVWXYZ)(,.=!?ab",
                      "cdefghijklmnopq", "rstuvwxyz " ]
```

Hierbei unterdrückt das Doppelsemikolon hinter der ersten Eingabe die Ausgabe von `A1`. Benötigt man das n-fache kartesische Produkt einer gegebenen Teilmenge A, so erhält man dies durch die Eingabe von `Cartesian(A,..,A);`, wobei `A` n-mal innerhalb der runden Klammern erscheint. Bei zu großem `A` kann es jedoch sein, daß `KASH` zuviel Speicher belegt und abgebrochen wird. Um dies zu vermeiden, benutzt man obigen Befehl `ListSplit`. Wendet man diesen auf `A` an und teilt dadurch`A` z.B. in Teillisten `A1,A2,A3`, so brechnet man jeweils das kartesische Produkt von `Cartesian(Ai,Aj,Ak);` (`i,j,k` $\in \{1,2,3\}$).

3 Chinesischer Restsatz und endliche Körper

Als nächstes wollen wir den chinesischen Restsatz anwenden. Deshalb soll hier kurz erläutert werden, wie man in `KASH` damit arbeiten kann.

Seien z.B. Zahlen $z_1 = 17$, $z_2 = 25$ und $z_3 = 133$ vorgegeben. Der Chinesische Restsatz besagt nun, daß für $n = z_1 \cdot z_2 \cdot z_3$ die Tatsache

$$\mathbb{Z}/n\mathbb{Z} \cong \mathbb{Z}/z_1\mathbb{Z} \times \mathbb{Z}/z_2\mathbb{Z} \times \mathbb{Z}/z_3\mathbb{Z} \tag{1}$$

gilt. Wenn man von ein Element in dem Produktring auf der rechten Seite von (1) das zugehörige Element von $\mathbb{Z}/n\mathbb{Z}$ berechnen möchte, kann man dies mit dem Hauptsatz der simultanen Kongruenzen tun. Dies bedeutet also, wenn man z.B. für ein Element $(6, 24, 99)$ des Produktringes ein $\alpha \in \mathbb{Z}/n\mathbb{Z}$ berechnen möchte, so tut man das in `KASH` mit

```
kash> ChineseRemainder([17,25,133],[6,24,99]);
20049
```

d.h. also das gesuchte Element in $\mathbb{Z}/56525\mathbb{Z}$ war $\alpha = 20049$ mit $\alpha \equiv 6 \mod 17$, $\alpha \equiv 24 \mod 25$ und $\alpha \equiv 99 \mod 133$.

Im RSA-Verfahren ist ist ein sogenannter RSA-Modul n vorgegeben, wobei $n = pq$ das Produkt zweier ungerader Primzahlen p und q ist. Ist z.B. $p = 309811$ und $q = 4000037$, so berechnet man die Ordnung der multiplikativen Gruppe von $\mathbb{Z}/n\mathbb{Z}$. Diese ist $(p-1)(q-1) = t = 1239251153160$. Ist nun ein $b \in \mathbb{N}$ mit $\mathrm{ggT}(b, t) = 1$, so läßt sich ein $a \in \mathbb{N}$ berechnen mit $x^{ab} \equiv x \mod n \; \forall x \in \mathbb{Z}/n\mathbb{Z}$. Ist z.B. $b = 32321$ so berechnet man a in KASH mit

```
kash> IntXGcd(32321,1239251153160);
[ 1, [ 11349226241, -296 ] ]
```

und erhält damit Zahlen $a, c \in \mathbb{Z}$ mit $at + cb = 1$. Hierbei ist $a = 11349226241$ und $c = -296$, was man durch

```
kash> 11349226241*b -296*t;
1
```

verifiziert. Oftmals ist es erforderlich mit endlichen Körpern zu rechnen. Ein Endlicher Körper `k` wird in `KASH` z.B. durch

```
kash> k:= FF(2,8);
Finite field of size 2^8
```

erzeugt, wie man ihn z.B. bei AES benötigt. Die schon bekannten Körper $\mathbb{Z}/p\mathbb{Z}$ für eine vorgegebene Primzahl p erzeugt man mit `FF(p,1)`. Um Elemente direkt angeben zu können, benutzt man

```
kash> k := FF(31847,2);
Finite field of size 31847^2
kash> a := FFGenerator(k);
w.
```

Man kann dann alle $31847^2 - 1$ von Null verschiedenen Elemente durch Potenzen von a erzeugen, d.h. $k^\times = \langle a \rangle = \{a^k \mid k = 1, \ldots, p^2 - 1\}$.

Für Polynome aus einem Körper k benutzt man statt `IntXGcd` die Funktion `PolyXGcd`. Ist z.B. $p = 31847$, so sei $k := \mathbb{Z}/p\mathbb{Z}$. Nun definiert man sich ein Polynom in $k[T]$ wie folgt:

```
kash> k := FF(p,1);
Finite field of size 31847
kash> kT := PolyAlg(k,"T");
Univariate Polynomial Ring in T over GF(31847)
kash> f := Poly(kT, [1,202,2,0,452333,1]);
T^5 + 202*T^4 + 2*T^3 + 6475*T + 1
```

und um zu überprüfen, aus welcher Polynomalgebra f nun kommt gibt man

```
kash> PolyAlg(f);
Univariate Polynomial Ring in T over GF(31847)
```

ein. Ist g ein zweites Polynom

```
kash> g := Poly(kT, [1,203,45095090,1]);
T^3 + 203*T^2 + 31585*T + 1
```

so kann man Elemente $\alpha, \beta \in k[T]$ berechnen mit $\alpha f + \beta g = 1$, nämlich durch

```
kash> PolyXGcd(f,g);
[ 1, 27363*T^2 + 8628*T + 4864,
  4484*T^4 + 18735*T^3 + 27737*T^2 + 25276*T + 26984 ]
```

wobei $\alpha = 27363T^2 + 8628T + 4864$ und $\beta = 4484T^4 + 18735T^3 + 27737T^2 + 25276T + 26984$ ist. Dies verifiziert man durch

```
kash> (27363*T^2 + 8628*T + 4864)*f +
                (4484*T^4 + 18735*T^3 + 27737*T^2 + 25276*T + 26984)*g;
1.
```

Der chinesische Restsatz für Polynome aus $k[T]$, wobei k ein Körper ist, angewandt auf f und g wäre hier z.B.

```
kash> h := ChineseRemainder([f,g],[T,T^2+2*T]);
30798*T^7 + 7150*T^6 + 19282*T^5 + 28693*T^4 + 9470*T^3 + 28606*T^2 +
   21484*T + 9167
```

was man wieder durch

```
kash> h mod f;
T
```

und

```
kash> h mod g;
T^2 + 2*T
```

verifiziert.