Einführung in die CoCoA-Programmierung

Bisher interaktive Programmierung

Ziel: Erweitere die Funktionalität um selbst erzeugte Zusatzfunktionen.

1. Einige Programmentwicklungserklärungsversuche

(a) Man kann neue Eingabedateien öffnen mit „File“, „New“.
Es erscheint im Eingabebereich ein zweites „Tab“.

(b) In der neuen Datei kann man die eigenen Funktionen eintippen.
Wenn man fertig ist, kann man die neue Funktion mit „Ctrl“ + „Enter“ absenden.
Dann prüft CoCoA die Syntax und wenn die neue Funktion ohne Fehler akzeptiert wird, ist sie einsatzbereit. Austesten im interaktiven Fenster

(c) Zum Schluss kann man den Inhalt des Dateifensters mit „File“, „Save As“ speichern.
Als Dateiname ist *.coc üblich.

2. Define … EndDefine

Neue CoCoA Funktionen werden erzeugt mit

```
Define MeineFunktion(A,I)      „A“, „I“ → Funktionsargumente
  B:=A^I;
  Return B;     Befehl „Return“ zum Zurückgeben des Ergebnisses
EndDefine;

A:=Mat([[0,1],[0,0]]);

MeineFunktion(A,2);  →  Nullmatrix
```

3. Programmablaufsteuerung (Wohin soll ich mich wenden…)

(a) Schleifenbefehl:

```
For I:=1 To 10 Do
  <Befehle>
EndFor;
```

Die Befehle werden 10-mal ausgeführt. Dabei hat die Variable `I` die Werte 1, 2, …, 10.

(b) Schleifenbefehl für Listen:

Sei `L` eine Liste

```
ForEach T In L Do
  <Befehle>
EndForEach;
```

Die Befehle werden für jedes Element in der Liste einmal abgearbeitet. Die Variable `T` enthält jeweils das momentane Element von `L`.

(c) Schleifen mit unbestimmter Wiederholungszahl:

```
While <logische Bedingung> Do
  <Befehle>
EndWhile;
```

Eine logische Bedingung ist eine Variable oder Berechnung die `TRUE` oder `FALSE` ergibt, z.B. `N=5` oder `N<>5`.
Die Befehle werden immer wieder abgearbeitet, solange bis die logische Bedingung den Wert `FALSE` ergibt.

(d) Verzweigungsbefehl:

```
If <logische Bedingung> Then
  <Befehle>
EndIf;
```

Die Befehle werden nur ausgeführt, wenn die logische Bedingung `TRUE` ergibt.

(e) Doppelter Verzweigungsbefehl:

```
If <logische Bedingung> Then
  <Befehle 1>
Else
  <Befehle 2>
EndIf;
```

Die Befehle 1 werden nur ausgeführt, wenn die logische Bedingung `TRUE` ergibt.
Wenn sie `FALSE` ergibt, werden die Befehle 2 ausgeführt.

Beispiel: Der euklidische Algorithmus

```
Define Euklid(A,B)
  If (A=0 And B=0) Then
    Return 0;
  EndIf;

  If A=0 Then
    Return AbsValue(B);
  EndIf;

  If B=0 Then
    Return AbsValue(A);
  EndIf;

  A:=AbsValue(A);
  B:=AbsValue(B);

  If A<B Then
    C:=A;
    A:=B;
    B:=C;
  EndIf;

  While Rest<>0 Do
    Rest:=Mod(A,B);
    A:=B;
    B:=Rest;
  EndWhile;

  Return A;
EndDefine;

Define AbsValue(B);
  If B<0 Then
    Return -B;
  Else
    Return B;
  EndIf;
EndDefine;
```

4. Hinterlistige Listen (Listen Konstruktur)

Sei z.B. `L` eine Liste, etwa `L:=1..100;`

(a) `G:=[N In L | Mod(N,2)=0];`
`N` → neuer Name
`Mod(N,2)=0` → logische Bedingung

→ `G=[2,4,6,…,100]`

(b) `H:=[N^2 | N In L];`

→ `H=[1,4,9,…,10000]`

(c) `I:=[N^2 | N In L And N<=10];`

→ `I=[1,4,9,…,100]`

Beispiel: Eigenwerte

Angenommen es gibt eine Funktion `CharPoly(A)`, die das charakteristische Polynom der Matrix `A` berechnet. Der Grundring sei `K[x]`.

```
Define Eigenwerte(A)
  F:=CharPoly(A);
  L:=Factor(F);

  M:=[P In L | Deg(P[1])=1];
  N:=[P[1] | P In M];

  LL:=[-CoeffOfTerm(1,F) / LC(F) | F in N];

  Return LL;
EndDefine;
```