Tocas

Claus Diem

30. April 2019

1 Was ist Tocas und wofür brauchen wir es?

Laut der deutschsprachigen Wikipedia ist ein Computeralgebrasystem (CAS) ([Wiki-CAS]) ein Computerprogramm, das der Bearbeitung algebraischer Ausdrücke dient.

Es löst nicht nur mathematische Aufgaben mit Zahlen (wie ein einfacher Taschenrechner), sondern auch solche mit symbolischen Ausdrücken (wie Variablen, Funktionen, Polynomen und Matrizen).

Diese Beschreibung ist gibt eine klassische Idee von Computeralgebrasystemen wider.

Entsprechend heißt das entsprechende Forschungsgebiet oftmals auch Symbolisches Rechnen.

Diese Beschreibung gibt allerdings nicht wirklich wider, was man als Mathematiker gerne hätte: Man hätte gerne ein System, das einen beim "Mathematik-machen" unterstützt.

Nun denken Mathematiker nicht in "Ausdrücken", die sie "umformen" (auch wenn sie dies auch ab und zu tun), sondern in abstrakten Objekten. Man hätte also gerne ein System, bei dem man bei der Benutzung das Gefühl hat, man "mache Mathematik".

Wenn wir Mathematik machen, denken wir über Objekte, und in diesem Sinne ist ein objektorientierter Zugang sicherlich ein Schritt in die richtige Richtung. Daneben haben wir bei mathematischen Objekten oftmals die Idee eines Typs im Kopf. Beispielsweise haben wir Ringe und als Spezialfälle beispielsweise Ganzzahl-Restklassenringe, Polynomringe, Körper oder endliche Körper. Ferner haben wir in Ringen Ringelemente, mit welchen wir dann rechnen können. Dies legt einerseits nahe, dass die Idee von Typen relevant sein sollte, wobei Typen spezialisiert werden können.

Eine naheliegende Umsetzung ist: Der Typ der Instanz einer Klasse ist die Klasse selbst.

Es gibt abstrakte Klassen, wie beispielsweise eine Klasse Ring oder eine Klasse RingElement, die nicht instanziierbar sind. Aus solchen abstrakten Klassen werden dann (möglicherweise über mehrere Schritte) instanziierbare Klassen abgeleitet, wie beispielsweise eine Klasse GanzzhalRestklassenring oder eine Klasse GanzzahlRestklassenringElement.

Dies kann sehr gut in python3 umgesetzt werden. Konkret hat hier (im Gegensatz zu python2) jedes Objekt hat wie beschrieben einen Typ, der gleich der entsprechenden Klasse ist. (Das Kommando zur Ausgabe des Typs lautet type.) Ferner gibt es eine sogenannte abstrakte Basisklasse (abstract base class, ABC), aus der abstrakte Klassen abgeleitet werden können.

Tocas steht für "Typenorientiertes Computeralgebrasystem". Es besteht im Moment aus einigen grundlegenden Klassen in python3. Ich habe dies vor Beginn des Sommersemesters implementiert.

Die zugrundeliegenden Ideen sind (natürlich !) nicht neu. Sie sind in meinen Augen zuerst im Computeralgebrasystem Magma verwirklicht worden. Dieses System ist im Sinne von Sprachdesign nicht objektorientiert, die Idee von Objekten, mit denen man hantieren kann, ist aber nichtsdestoweniger in besonderem Maße verwirklicht.

Die erste Version dieses Systems stammt von 1993. Inzwischen ist das System enorm groß. Wie bei Computeralgebrasystemen üblich, gibt es keinen Compiler, das System ist aber nichtsdestoweniger extrem schnell. Die Bedienung erfolgt über ein Terminal. Magma wurde und wird an der Universität Sydney entwickelt. Es ist nicht frei erhältlich, aber es gibt einen "Magma Calculator", der online verfügbar ist.

Seit 2005 gibt es ein großes, freies Konkurrenzprojekt zu Magma: sage. Dieses System integriert verschiedene schon vorher existierende Compueralgebrasysteme. Das System selbst ist weitgehend auch in python implementiert. Hierzu gibt es, im Gegensatz zu Magma, ein sogenanntes Notebook für die Eingabe und auch graphische Ausgaben; dies ist das Jupyter-Notebook, das man auch gut für python und somit auch für Tocas verwenden kann.

Warum dann ein neues System und nicht Magma oder sage? Nun, der Grund ist ganz einfach: Diese Systeme sind zu gut. Mit Tocas haben Sie einen Ausgangspunkt, mit dem Sie mit vertretbarem Aufwand einige einfache Algorithmen selbst implementieren können und dann Ihr Projekt angehen können. In Magma oder sage wäre alles schon vorhanden.

Selbstredend können (und sollten Sie) sich diese System aber auch mal anschauen. Es gibt auch spezielle python-Bibliotheken für mathematische Themen. Solche Bibliotheken sollten Sie für die Hausaufgaben und das Projekt nicht von sich aus benutzen.

Wenn Ihnen etwas sinnvoll erscheint, sollten Sie mich fragen, Sie sollten dabei aber davon ausgehen, dass ich die Benutzung ablehnen werde.

2 Zu python

Viele von Ihnen sind sicherlich python schon gut vertraut. Ich möchte nur ein paar Aspekte erwähnen, die für mich eine gewisse Hürde dargestellt haben und die vielleicht Probleme bereiten könnten.

2.1 Technisches

Erstmal das rein “Technische”: Wie schon geschrieben benötigen wir python3.

Ich benutze linux in der ubuntu-Distribution. Es gibt ein Paket python, das vielleicht sogar schon vorinstalliert ist. Hiermit habe ich aber Probleme. Denn: Ich mache immer gerne “drag and drop” von einem Editor aus ins python-Terminal. Und das funktioniert hier bei mehreren Zeilen nicht gut. Gerade die wichtigen Einrückungen scheinen oftmals fehlerhaft zu werden, was dann zu Fehlern führt.

Die bessere Variante ist ipython. Dies funktioniert bei mir sehr gut. Wenn wir gerade bei ipython sind: Sie sollten sich auch Jupyter anschauen. Das Notebook ist praktisch. Besonders schön ist, dass man ausführbaren Beispielcode mit Kommentaren erstellen kann. Es gibt ein paar gute Editoren für python-Code. Einer ist emacs, ein anderer ist geany. In letzerem sieht der Code noch besser und die Unterstützung sieht erstmal besser aus. Man muss aber aufpassen: Einrückungen sollten mit Leerzeichen gemacht werden und nicht mit Tabs. In emacs wird das optimal unterstützt, in geany aber nicht. Hier muss man automatische Einrückungen wieder löschen. (Ich weiß nicht, ob es auch eine Option hierzu gibt.)

Es gibt auch Umgebungen für python, darunter die eclipse-Variante liclipse. Diese Software ist allerdings nicht frei. Nach einer Testperiode von 30 Tagen muss man 80 Dollar bezahlen.

2.2 Einbinden von Tocas

Tocas ist darauf ausgelegt, dass man damit “rumbasteln” kann. Zum Einbinden der Module sollte man in das Verzeichnis gehen und dann

```
from Tocas import *
```

ingeben.

Dies importiert die Definitionen in den Standardnamensraum, d.h. “ohne Punkt-Namen”.

Dies ist natürlich nicht der normale Weg, um Module in python einzubinden. Normalerweise schreibt man in ein Verzeichnis eine init-Datei mit Namen __init__.py. Das Verchnis ist dann ein Paket. So ein Paket importiert man mit:

```
import Verzeichnisname *
```

oder

```
from Verzeichnisname import *
```

Hierfür müssen dann aber – so scheint es mir jedenfalls – alle internen import-Kommandos geändert werden, z.B. von

```
from modulX import *
```

zu:

```
from .modulX import *
```

(Man beachte den Punkt!)

Leider funktioniert die punktierte Variante nicht, wenn man ein Modul von innerhalb eines Verzeichnisses aufruft. Hier musste ich also eine Entscheidung treffen und habe mich für die "bastel"-Variante "ohne Punkt" entschieden.

Sie können dies aber leicht ändern, indem Sie bei den import-Statements die Punkte einfügen.

2.3 Die Bedeutung von Variablen in python

Die Sprache python ist so designed, dass es einfach ist, "schönen" Code zu schreiben. Die Einstiegshürde ist auch niedrig, man kann einfach loslegen. Mit einem Aspekt sollte man sich allerdings vertraut machen: Die Bedeutung von Variablen in python.

Eine grundlegende Idee von python ist: "Alles ist ein Objekt." Das was man normalerweise als "Variablen" bezeichnet, sollte man sich dann als mögliche Namen von Objekten vorstellen. In den Worten der python-Dokumentation ([Python-Doc], Abschnitt 4.2, "naming and binding":

> *Names* refer to objects. Names are introduced by name binding operations.

Ein Beispiel dazu. Der Code

```
L = [“a”,”b”,”c”]
M=L
M[2] = “d”
L
```

führt zu: ['a', 'b', 'd']

Dies bedeutet auch: Wenn man ein Objekt kopieren will muss man dies explizit formulieren. Das passende Kommando dazu ist deepcopy. (Das Kommando muss man allerdinngs deutlich weniger oft verwenden, als man zunächst denken würde. In Tocas habe ich es gar nicht verwendet.)

Standardmäßig sind einige Objekte unveränderbar. So gibt es neben Listen, die veränderbar sind, auch die unveränderbaren Tupel. Aber Instanzen von neu geschaffenen Klassen sind standardmäßig veränderbar. Man kann allerdings das Verändern auch unterbinden, indem man __setattr__ überschreibt. In Tocas gibt es eine abstrakte Klasse EinfrierbaresObjekt. Wenn man von dieser Klasse ableitet, kann man sie “einfrieren” und wieder “auftauen”. Um Fehlerquellen zu vermeiden und weil es mir auch inhaltlich sinnvoll erscheint, habe ich alle Klassen in Tocas von dieser Klasse abgeleitet.

3 Designkriterien

Ich gebe einige Designkriterien an. Diese sollten Sie dann auch beachten.

- Das Typenkonzept von python3 wird beständig verwendet.
- Vererbung wird so weit wie möglich benutzt.
- Es gibt nicht nur Objekte “zum Rechnen”, sondern auch “Rechenbereiche”, z.B. gibt es nicht nur Ringelemente sondern auch Ringe. Die Rechenbereiche treten als Umgebungen von Elementen auf. (Da Tocas im Wesentlichen nur Klassen für Ringe und Ringelemente umfasst, ist dies nur hierfür implementiert, nämlich mittels der Methode umgebung von RingElement.)
- Jede Klasse hat einen Gleichheitsoperator (“==”), basierend auf einer rechnerisch einfachen Isomorphie. Wenn für zwei Objekte O1 und O2 die Bedingung O1 == O2 erfüllt ist, schreibe ich im Folgenden: O1 und O2 gelten als gleich.

4 Die Klassen und Funktionen von Tocas

Es sollen nun die Klassen und Funktionen angegeben werden, die Tocas bietet.

In python3 kann man abstrakte Klassen durch Ableitung von der Klasse ABC definieren. Eine abstrakte Klasse ist dann eine Klasse, die mindestens eine abstrakte Methode hat. So eine Klasse ist dann nicht instanziierbar. Die Klassen in Tocas sind entweder abstrakt oder instanziierbar.

Es folgt eine Auflistung der Klassen und Funktionen unter Angabe der entsprechenden Moduln.

- Abstrakte Klassen
  - MeinABCObjekt in AbstrakterAnfang
  - EinfrierbaresObjekt in AbstrakterAnfang
  - Ring in AbstrakteRinge
  - RingElement in AbstrakteRinge

  Instanziierbare Klassen
  - RingTupel in AbstrakteRinge

– Ganzzahlring in AbstrakteRinge
– GanzzahlringRestklassenring in GanzzahlRestklassenringe
– GanzzahlRestklassenringElement in GanzzahlRestklassenringe
– Polynomring in Polynomringe
– PolynomringElement in Polynomringe
– Homomorphismus in Homomorphismen
– KringelHomomorphismus in Homomorphismen

Fehlerklassen

– InvertierungsFehler in AbstrakteRingen

Funktionen

– TypBeschreibung in AbstrakterAnfang
– ZweiAdisch in AbstrakteRinge

Es folgt eine Beschreibung der Klassen und Funktionen. Die abstrakten Klassen beschreibe ich eher informal, die instanziierbaren Klassen und Funktionen strukturierter.

Abstrakte Klassen

MeinABCObjekt

Diese abstrakte Klasse ist von der "abstract base class" ABC abgeleitet.

Sie erfüllt nur einen Zweck:

In python gibt es unterschiedliche Ausgabemethoden, wenn man einerseits "nur" den Objektnamen eingibt oder andererseits das Drucke-Kommando print benutzt. Mit dieser Klasse wird die Methode für das Erstere (repr) auf die Methode für das Zweitere (str) umgeleitet. Damit werden die Ausgaben also identisch.

Alle folgenden Klassen sind von dieser Klasse abgeleitet, und so sollten Sie auch verfahren.

EinfrierbaresObjekt

In python gibt es sowohl *mutuable* als auch *immutable objects*. Immutuable sind insbesondere Zahlen und Objekte vom Typ tuple. Objekte von selbst kreierten Klassen sind jedoch standardmäßig mutuable. Es gibt keine Standardmethode (im allgemeinen Sinn des Wortes), um aus eine Klasse für immutuable objects zu implementieren.

Mit dieser Klasse wird etwas Ähnliches (für unsere Zwecke Besseres) bereitgestellt: Wenn man eine Klasse hiervon ableitet und self._frier() angibt, wird das Objekt “eingefroren”, indem die setattr-Methode überschrieben wird. Das einzige noch funktionierende Schreibkommando ist dann self._tau(), mit dem man das Objekt wieder “auftauen” kann.

Wiederum sind alle folgenden Klassen von dieser Klasse abgeleitet.

Ring

Es ist nur ein rudimentärer Gleichheitstest implementiert. Als Attribute sind vorgesehen: null und eins für die Null und die Eins des Rings. Es gibt eine abstrakte Methode: element. Dieses soll zu einer noch näher zu bestimmbaren Information ein Ringelement aus der Instanz (also dem Ring) liefern.

Man kann ein Objekt als als Funktion aufrufen: R(info) ist identisch zu R.element(info).

RingElement

Es sind Methoden zur Ausgabe, zum Gleichheitstest, zur Arithmetik implemtiert oder als abstrakte Methoden angegeben. Es ist ein (Instanz-)Attribut vorgesehen: ring. Diese soll auf den Ring verweisen, aus welchem die Instanz (das Ringelement) liegt.

Die folgenden Instanzmethoden sind implementiert:

- Methoden zum Drucken von Elementen (für Details siehe Code).
- Ein rudimentärer Gleichheitstest.
- Eine Methode umgebung. Diese gibt das Attribut ring zurück, d.h. den Ring, in welchem die Instanz (das Ringelement) liegt.
- Einige Methoden für Arithmetik. Hier werden die “normalen” Operatoren immer zuerst auf die “rechts-Operatoren” (z.B. radd) geleitet. Hiermit kann links dann auch eine ganze Zahl stehen und die Operation der Form e * f für ein Ringelement e und ein anderes Element f kann noch “bei ” implementiert werden.

Für Fehler beim Invertieren (mit der hier noch abstrakten Methode invers) ist die Fehlerklasse InvertierungsFehler vorgesehen.

Ferner gibt es die statischen Methoden test zum Testen, ob es sich um ein Ringelement im mathematischen Sinn handelt, und ring, um den entsprechenden Ring zu erhalten. Der Grund für diese Methoden ist: Hiermit werden Fallunterscheidungen für ganze Zahlen vermieden. Denn leider sind ganze Zahlen nicht abgeleitet von RingElement.

5 Instanziierbare Klassen

In den folgenden Beschreibungen wird "links" in der Instanziierung immer ein Name angegeben. Dies geschieht nur, um später darauf zu verweisen.

Bei den Eingaben gebe ich an, welchen Typ diese haben sollen. Dies mache ich in der Form

> Objekt : Typ

wenn Objekt genau den angegebenen Typ Typ haben soll. Dies entspricht type(Objekt) == Typ.

Manchmal wird nur verlangt, dass der Typ von Objekt vom angegebenen Typ abgeleitet sein soll. Dann schreibe ich:

> Objekt : abgeleitet von Typ

Dies entspricht isinstance(Objekt,Typ) == true.

Wenn nötig gebe ich noch Einschränkungen an.

Ganzzahlring

Hiermit kann ein Ganzzahlring instanziiert werden. Eine Instanz sollte Z genannt werden, im Code wird auch schon eine solche Instanz erzeugt.

Elternklasse. Ring

Instanziierung.

> Z = Ganzzahlring()

Attribute. keine

Instanzmethoden.

- "==": Es ist Z == Y genau dann, wenn Y auch eine Instanz der Klasse ist.

Verwendung als Funktion. Siehe Ring.

Statische Methoden.

- ext_ggt Mit dieser Methode ist der erweiterte Euklidische Algorithmus implementiert. (Der Grund, warum ich dies als statische Methode und nicht als freistehende Funktion implementiert habe, ist: Sie sollen eine entsprechende Methode implementieren, und diese sollte dann eine statische Methode von Polynomring sein.)

Aufruf.

ext_ggt(a,b) , a,b : int

Hinweis. Im Gegensatz zu GanzzahlRestklassenringElement und PolynomringElement ist keine Klasse GanzzahlringElement implementiert. Der Grund ist, dass die ganzen Zahlen in python bereits implementiert sind. Aufgrund der Tatsache, dass es diese Elemente nicht gibt, muss man ab und an eine Fallunterscheidung machen.

GanzzahlRestklassenring

Mit dieser Klasse sind Ganzzahl-Restklassenringe implementiert.

Elternklasse. Ring

Instanziierung.

R = GanzzahlRestklassenring(n) , n : int ($n \geq 2$)

Hier ist dann n der Modulus des Rings.

Attribute. null (die Null), eins (die Eins) und modulus (Modulus n)

Instanzmethoden.

- "==": Es ist R == S genau dann, wenn S auch eine Instanz von GanzzahlRestklassenring ist und denselben Modulus hat.
- element

  R.element(a) , a : int oder GanzzahlRestklassenringElement

  ruft GanzzahlRestklassenringElement(a,R) auf und erzeugt das Bild der Zahl oder der Restklasse a im Ring R.
- zufaellig:

  R.zufaellig()

  erzeugt ein zufälliges Element aus R.

Verwendung als Funktion. wie bei Ring.

Statische Methoden. keine

GanzzahlRestklassenringElement

Hiermit sind Elemente aus Ganzzahl-Restklassenringen implementiert.

Elternklasse. RingElement

Instanziierung.

$$\begin{aligned}\mathsf{e} = \quad & \mathsf{GanzzahlRestklassenringElement(a,R)} \quad , \\ & \mathsf{a} : \mathsf{int} \text{ oder } \mathsf{GanzzahlRestklassenringElement} \quad , \\ & \mathsf{R} : \mathsf{GanzzahlRestklassenring}\end{aligned}$$

oder

$$\begin{aligned}\mathsf{e} = \quad & \mathsf{GanzzahlRestklassenringElement(a,n)} \quad , \\ & \mathsf{a} : \mathsf{int} \text{ oder } \mathsf{GanzzahlRestklassenringElement} \ , \ \mathsf{n} : \mathsf{int} \ (\mathsf{n} \geq 2)\end{aligned}$$

Hiermit wird die von a definierte Restklasse in R oder in einem neu instanzierten Moduloring erzeugt. Wenn a selbst eine Restklasse (und keine ganze Zahl) ist, muss n (oder der Modulus von R) ein Teiler des Modulus der Restklasse a sein.

Attribute. ring und wert mit den offensichtlichen Bedeutungen. (modulus ist kein Attribut, aber ein Attribut des Attributs ring.)

Instanzmethoden

- "==": Es ist e == f genau dann, wenn die Ringe zu e und f als gleich gelten und die Werte gleich sind.
- Methoden für Arithmetik (Die Multiplikation ist durch dobule-and-add im Restklassenring implementiert, das ist wesentlich effizienter als zuerst Multiplikation in ganzen Zahlen und dann Reduktion.)
- Die Methode umgebung von RingElement.

Verwendung als Funktion. nein

Statische Methoden. keine

Polynomring

Hiermit sind Polynomringe (in einer Variablen) implementiert.

Elternklasse. Ring

Instanziierung.

$$\mathsf{P} = \mathsf{Polynomring(R)} \quad , \quad \mathsf{R} : \text{abgeleitet von } \mathsf{Ring}$$

oder

$$\begin{aligned} \mathsf{P} = \quad & \mathsf{Polynomring(R,variablenname)} \quad , \\ & \mathsf{R} : \text{abgeleitet von } \mathsf{Ring}, \mathsf{variablenname} : \mathsf{str} \end{aligned}$$

Erzeugt wird der Polynomring (in einer Variablen) über dem Ring R. Wenn kein Variablenname angegeben wird, wird dieser gleich 'x' gewählt. Der Name ist nur für die Ausgabe relevant.

Attribute. null, eins, variable, variablenname mit den offensichtlichen Bedeutungen und basisring mit P.basisring = R

Instanzmethoden.

- "==": Es gilt R == S genau dann, wenn S auch eine Instanz von Polynomring ist und die Basisringe vergleichbar sind. Der Variablenname ist hierbei irrelevant.
- element

  $$\mathsf{R.element(koeffizienten)} \quad , \quad \mathsf{koeffizienten} : \mathsf{RingTupel}$$

  erzeugt eine Instanz von PolynomringElement, und zwar das Element in R mit Koeffiziententupel koeffizienten.
- monom:

  $$\mathsf{R.monom(exp)} \quad , \quad \mathsf{exp} : \mathsf{int}\ (\mathsf{exp} \geq 0)$$

  erzeugt auch eine Instanz von PolynomringElement, und zwar $\mathsf{x}^{\mathsf{exp}}$.

Verwendung als Funktion. Siehe Ring.

Statische Methoden. keine

PolynomringElement

Hiermit sind Elemente von Polynomringen implementiert.

Elternklasse. RingElement

Instanziierung.

e = PolynomringElement(koffizienten,polyring)
koeffizienten : RingTupel
polyring : abgeleitet von Polynomring

Das Ergebnis ist das Ringelement zum Koeffiziententupel koeffizienten.

Attribute. ring als Ring polyring; grad, der Grad eines Elementes; basisring als Basisring von polyring; koeffizienten als Koeffiziententupel (mit Typ RingTupel).

Methoden.

- "==": Es ist e == f genau dann, wenn die Polynomringe als gleich gelten und die Koeffiziententupel als gleich gelten.
- Methoden für die Arithmetik

Hinweis. Die Koffizienten werden immer mittels des Tupels koeffizienten abgespeichert.

Dies bedeutet: Für ein Monom x^n müssen n Nullen gefolgt von einer Eins gespeichert werden. (Dies nennt man "dichte Darstellung", im Gegensatz zur "dünnen Darstellung", in welcher nur die nicht-Null-Einträge mit Monom und Koeffizient abgespeichert werden.)

Aufruf als Funktion. nein

Statische Methoden. keine

RingTupel

Hiermit sind Tupel von Elementen aus demselben Ring implementiert.

Der wesentliche Grund für diese Implementierung ist: Bei Listen ist "+" als Konkatination implentiert, die anderen arithmetischen Opreationen entsprechen auch nicht dem Rechnen mit Tupeln von Ringelementen.

Elternklasse. EinfrierbaresObjekt

Instanziierung.

e = PolynomringElement(koeffizienten) ,
koeffizienten : tupel oder liste

Hier muss koeffizienten eine Liste oder ein Tupel von Ringelementen von vergleichbaren Ringen sein. Oder

e = PolynomringElement(koeffizienten,R) ,
koeffizienten : tupel oder liste, R : Ring

Hier müssen die Koeffizienten in koeffizienten kanonisch in den Ring R abbildbar sein.

Oder:

e = PolynomringElement(ringelement,n)
ringelement : abgeleitet von RingElement oder int , n : int

Das Ergebnis ist das Tupel mit konstantem Eintrag ringelement und Länge n.

Attribute. ring und laenge mit den offensichtlichen Bedeutungen; koeffizienten, dies ist eine Liste (list) von Koeffizienten.

Instanzmethoden.

- "==" definiert über koeffizientenweisen Vergleich
- Methoden für die Arithmetik
- Methoden für die üblichen Operatoren für eckige Klammern für Tupel / Listen (lesen und setzen)
- auslaufende_nullen_loeschen: Hiermit werden (wie der Name sagt) auslaufende Nullstallen einer Instanz gelöscht. (Dies wird von PolynomringElement benutzt.)

Aufruf als Funktion. nein

Statische Methoden.

- zusammenfuegen Hiermit werden zwei Tupel mit Elementen aus vergleichbaren Ringen zusammengefügt.

Homomorphismus

Auch die Homomorphismen zwischen Ringen kann man als Objekte betrachten. Mit dieser Klasse kann man solche Objekte instanziieren. Natürlich kann man die Homomorphismen dann auch auf Ringelemente anwenden.

Das Design folgte diesen Prinzipien:

Oftmals gibt es kanonische Homomorphismen zwischen Ringen. Wenn dies der Fall ist, sollten die Angabe von Quelle und Ziel ausreichen, um so einen zu instanziieren. Insbesondere sollte für als gleich geltende Ringe hiermit der kanonische Isomorphismus erzeugt werden. Für Polynomringe wird auf dieses Resultat zurückgegriffen:

Für einen Polynomring $R[x]$, einen weiteren Ring S, einen Ringhomomorphismus $\varphi : R \to S$ und ein Element $a \in S$ gibt es genau einen Ringhomomorphismus $\psi : R[x] \to S$ mit $\psi|_R = \varphi$ und $\psi(x) = a$.

Hierbei wird noch diese Idee verfolgt: Wenn dann der Homomorphismus ψ kanonisch ist, kann er auch weggelassen werden.

Elternklasse. EinfrierbaresObjekt

Instanziierung.

hom = Homomorphismus(quelle,ziel) ,
quelle, ziel : abgeleitet von Ring

hom = Homomorphismus(quelle,ziel,element)
quelle, ziel : abgeleitet von Ring,
element : abgeleitet von RingElement

hom = Homomorphismus(quelle,ziel,element,basishom)
quelle, ziel : abgeleitet von Ring,
element : abgeleitet von RingElement,
basishom: abgeleitet von Homomorphismus

Hierbei müssen die offensichtlichen Kompatibilitäten gelten: element muss ein Element aus einem zu ziel als gleich betrachteten Ring sein; basishom muss als Quelle einen zum Basisring von quelle passenden Ring haben und als Ziel einen zu ziel passenden. (Dies wird strikt überprüft. Wenn's nicht passt, muss vielleicht mit einer Verkettung gearbeitet werden.)

Attribute. quelle, ziel, element, basishom mit den offensichtlichen Bedeutungen.

Instanzmethoden.

- "==": Definiert über Vergleich der Attribute.
- anwenden: hom.anwenden(e) liefert das Bild von e unter dem Homomorphismus.
- "*": Für einen weiteren Homomorphismus hom2 ist hom * hom2 die Verknüpfung von hom mit hom2. Dies ist dann vom Typ KringelHomomorphismus.

Verwendung als Funktion. hom(e) ist identisch mit hom.aufruf(e)

Statische Methoden. keine

KringelHomomorphismus

Hiermit ist die Verkettung von zwei Homomorphismen implementiert. Das Ergebnis ist besteht aus beiden Homomorphismen zusammen mit der Verkettung. Für die Anwendung auf ein Element werden diese dann hintereinander ausgeführt.

Mit anderen Worten: Die Verkettung wird nicht aufgelöst. Dies führt zu einem sehr schwachen Gleichheitsbegriff.

Elternklasse. Homomorphismus

Instanziierung.

h = KringelHomomorphismus(hom1,hom2)
hom1, hom2 abgeleitet von Homomorphismus

Hier muss selbstredend die Quelle von hom1 zum Ziel von hom2 passen.

Attribute. quelle, ziel, hom1, hom2 mit den offensichtlichen Bedeutungen

Instanzmethoden.

- "==": Definiert über Vergleich der Attribute
- anwenden: Die Ausführung der Verkettung auf ein Element
- "*": Die Verkettung

Verwendung als Funktion. hom(e) ist identisch mit hom.aufruf(e)

Statische Methoden. keine

Funktionen

typ_beschreibung

Für eine Instanz einer Klasse wird der Name der Klasse als String zurückgegeben. Dies ist sehr ähnlich zur Ausgabe von type, ist nur ein wenig schöner.

Aufruf.

typ_bschreibung(O) , O ein beliebiges Objekt.

zwei_adisch

Mit dieser Funktion wird eine nicht-negative ganze Zahl in einen 01-String umgewandelt, der die Zahl im 2-er System beschreibt.

Aufruf.

zwei_adisch(n) , n : int

Literatur

[Python-Doc] Python 3.6.5, The Python Language Reference, Python Software Foundation

[Wiki-CAS] Artikel "Computeralgebra" in der deutschsprachigen Wikipedia, abgerufen am 15.4.2018