Übung 1: Einführung

http://people.f4.htw-berlin.de/~hebold/htw/pka/exercises/einf%C3%BChrung.pdf

Schreiben Sie in Java jeweils ein Programm, das die Funktionen

1. $x + y$
2. $x * y$
3. x^y

als int-Werten rekursiv berechnet

```
public static int add(int x, int y) {
  if (y == 0) { return x; }
  return 1 + add(x, y-1);
}

public static int multiply(int x, int y) {
  if (y == 0) { return 0; }
  return x + multiply(x, y-1);
}

public static int pow(int x, int y) {
  if (y == 0) { return 1; }
  return x * pow(x, y-1);
}
```

Schreiben Sie in Java Programme zu Berechnung der Fakultät mit dem Datentyp BigInteger.

Collatz-Problem Definition:
$f : \mathbb{N} \to \mathbb{N}$ mit $f(1) = 1$, n gerade: $f(n) = n/2$,n ungerade: $f(n) = 3n + 1$
$F : \mathbb{N} \to \mathbb{N}$ mit $F(1) = f(1) = 1$, $F(n) = F(f(n))$
Collatz-Problem: Ist F für jedes $n \in \mathbb{N}$ definiert, d.h. $\forall n \in \mathbb{N} \; \exists \; F(n) \in \mathbb{N}$?

```
public static int f(int x) {
    if (x == 1) { return 1; }
    else if (x % 2 == 0) { return x / 2; }
    return 3 * x + 1;
}
public static int F(int x) {
    if (x == 1) return 1;
    return F(f(x));
}
public static int Flength(int x, int c) {
    if (x == 1) return c;
    return Flength(f(x), c + 1);
}
public static int Flength(int x) {return Flength(x, 1);}
```

Relationen, mit $R \subseteq M \times M$

Reflexivität:	$(x, x) \in R$
Symmetrie:	$(x, y) \in R \Rightarrow (y, x) \in R$
Antisymmetrie:	$(x, y) \in R, (y, x) \in R \Rightarrow x = y$
Asymmetrie:	$(x, y) \in R \Rightarrow (y, x) \notin R$
Transitivität:	$(x, y) \in R, (y, z) \in R \Rightarrow (x, z) \in R$
Funktion:	bijektiv = surjektiv (rechtstotal, isOnto) + injektiv (linkseindeutig, isOneOne)

Mengen: $A = \{3, 4\}$, $B = \{\{3, 4\}\}$
Welche Behauptung stimmt?

1. $A = B$
2. $A \subseteq B$
3. $A \subsetneq B$
4. $|A| = |B|$

---

1. Nein, da unterschiedlich mächtig. siehe 4.)
2. Nein, da gelten muss: $\forall x \in A : x \in B$, aber hier: $3, 4 \notin B$
3. A keine echte Teilmenge von B, da gelten muss:
   $A \subset B \wedge A \neq B \Rightarrow 1.) \wedge \neg 2.)$, 1. ist falsch
4. Nein, da $2 \neq 1$

Das cartesische Produkt zweier Mengen A und B ist wie folgt definiert:
$A \times B = \{(x, y) : x \in A \wedge y \in B\}$
Prüfen Sie, ob das kartesische Produkt assoziativ ist, d.h. ob für Mengen X,Y,Z gilt: $(X \times Y) \times Z = X \times (Y \times Z)$

Nein, da andere Struktur

Welcher der folgenden Ausdrücke ist korrekt?

1. $0 \in \emptyset$
2. $0 = \emptyset$
3. $0 \subseteq \emptyset$
4. $\{0\} \subseteq \emptyset$

---

1. Nein, die leere Menge hat keine Elemente
2. Nein, Zahlen sind keine Menge
3. Nein, siehe 2.)
4. Nein, siehe 1.)

Sei X eine Menge endlicher Größe und 2^X die Potenzmenge von X.
Welches Ergebnis liefern:

1. $|2^X \cup X|$
2. $|2^X| \cup |X|$

Beispiel: Potenzmenge von $\{a, b\} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$

1. $|X| = 2^{|X|}$
2. Widerspruch, Vereinigung von Zahlen, nicht Mengen

Übung 2: Konzepte Abstraktion

http://people.f4.htw-berlin.de/~hebold/htw/pka/exercises/konzepte-Abstraktion.pdf

Nennen Sie mindestens 3 Gründe für Abstraktion.

---

1. Wiederverwendbarkeit von allgemeinen Problemlösungen
2. Klassifizieren von Problemen, erkennen der Struktur
3. Allgemeine Lösung zu detaillierten Problemen ($\approx$ Kompression)
4. Vereinfachung, reduzieren auf gemeinsame Eigenschaft

Inwiefern wird beim Programmieren ganz generell abstrahiert?

---

1. OOP $\Leftrightarrow$ reelle Welt, internes Modell als Abstraktion
2. Abstraktion von Problembeispielen auf Programmcode
3. Programme $\Leftrightarrow$ Prozessen (vgl. Debuggen - ständiger Kontextwechsel)

Inwiefern wird bei der **strukturierten** Programmierung abstrahiert?

---

1. if, for, while-Konstrukte ersetzen Sprungbefehle
   Beispiel: labels mit goto (?)

Inwiefern wird bei der **prozeduralen** Programmierung abstrahiert?

---

1. parametrisierte Prozeduren ersetzen alle Werte-Kombinationen beim Aufruf
   Beispiel: f(x) $\widehat{=}$ f(1), f(2), ... f(n)

Inwiefern wird bei der **modularen** Programmierung abstrahiert?

---

1. statische Werte, d.h. kein Zustand
2. Blackbox, Implementierung unbekannt
   Beispiel: sort(array[] field) sortiert ohne, dass wir wissen wie.

Inwiefern wird bei der **objekt-orientierten** Programmierung abstrahiert?

---

1. Vererbung, generischen Datentypen
   Beispiel: A extends B
   (A Konkretisierung von B $\leftrightarrow$ B Generalisierung/Abstraktion von A)
2. Polymorphismus, dynamisches Binden
   Beispiel:

Inwiefern wird bei der **funktionalen** Programmierung abstrahiert?

---

1. Nur Funktionen und Rückgabewerte, keine Datentypen, bzw. Objekte
   Beispiel:
2. Überladen von Funktionen
   Beispiel:

Inwiefern wird bei der Programmierung abstrakter Datentypen abstrahiert?

---

1. Lists, Arrays als Abstraktion, vgl. Gruppe (Math.)
   Beispiel: new ArrayList<Integer>().get(0)

Beim Abstraktionskonzept wird auf verschiedenen konkreten Objekten mit einem Namen referiert, wobei die Besonderheiten unberücksichtigt bleiben - von diesen wird abstrahiert.
Bei der Konkretisierung wird umgekehrt einem Namen ein bestimmtes konkretes Objekt zugeordnet - der Name wird gebunden.
Wann erfolgt im Rahmen der Programmierung die Konkretisierung, d.h. die Bindung eines Namens?

Bei der Zuweisung wird der Typ und Name konkret festgelegt: Zur Laufzeit
Beispiel:

```
// Abstraktion:
List x = new ArrayList();
// Konkretisierung:
void doSth(List x) {ArrayList y=(ArrayList)x;}
```

Imperative Programmierung:
Von was wird durch einen Variablennamen abstrahiert?

Vom Wert, da nur die Referenz auf den Wert benutzt wird
Beispiel: int x = 3; x=x+4;

Imperative Programmierung:
Von was wird durch Pointer abstrahiert?

Von der Speicheradresse
Beispiel

Imperative Programmierung:
Von was wird durch eine Initialisierung `int i=42` abstrahiert?

Von der Speicherdarstellung
Beispiel: Big Indian/ Little Indian(?)

Imperative Programmierung:
Von was wird durch eine Zuweisung abstrahiert?

Von allen verschiedenen Zuweisungsoperatoren
Beispiel

Imperative Programmierung:
Von was wird durch

1. if-Abfrage
2. for-Schleife `for(int i=0;i<a;i++) block`
3. while-Schleife

abstrahiert?

---

goto + label
Beispiel

Imperative Programmierung:
Von was wird durch eine Prozedur (void) abstrahiert?

Von der Implementierung
Beispiel

Imperative Programmierung:
Von was wird durch eine Funktion (non-void) abstrahiert?

Von der Implementierung
der Rückgabewert abstrahiert von der Funktion
Beispiel

Imperative Programmierung:
Von was wird in C und C++ und Java durch den abstrakten Datentyp Array abstrahiert?

C und C++: Von Pointern, Beispiel:

Java: Von Referenzen, Beispiel:

```
{"a", "b", "c"}.get(0)
```

Imperative Programmierung:
Funktionen werden in C, C++ und Java durch Aufrufe zur Laufzeit konkretisiert. Signatur und Methode sind die Abstraktionen. Zusätzlich bietet C++ Funktionen mit default Parametern. Was bedeuten diese für Abstraktion und Konkretisierung?

???

Imperative Programmierung:
Von was wird in C++ durch eine inline-Funktion abstrahiert?

Wie Makros als Textersetzung ohne Stack, jedoch wie Funktion mit Auswertung

Objektorientierte Programmierung:
Von was wird in Java durch eine Referenz `Type ref` abstrahiert?

`Type`: Für alle Objekt-Typen und deren Ableitung von Type
`ref`: Abstraktion der Objekte, aber nicht vom Objekt selbst

Objektorientierte Programmierung:
Die meisten objektorientierter Sprachen verfügen über primitive Datentypen wie z.B. int. Warum haben diese primitiven Datentypen aus der Warte des Abstraktionskonzepts einen Sonderstatus?

Call-by-Value: Passen direkt in Referenzspeicherbereich, keine Kapselung notwendig

Objektorientierte Programmierung:
Was wird durch die ausschließliche Verwendung von Klassen, Objekten und Referenzen, d.h. durch die Streichung der primitiven Datentypen, im Sinne des Abstraktionskonzepts erreicht?

Kontinuität

Objektorientierte Programmierung:
C++ und Java kennen die Möglichkeit des **overriding**. Inwiefern handelt es sich um eine Abstraktion? D.h. von welchen konkrete Elementen wird abstrahiert?

Von der Implementierung der Objekt-Methode

Objektorientierte Programmierung:
Mehrfachvererbung ist in Java bei Klassen nicht zugelassen.

1. Nennen Sie eine Begründung im Rahmen des Abstraktionsprinzips.
2. Wieso ist Mehrfachvererbung bei Interfaces zugelassen?
3. Wie löst C++ die genannten Probleme?

---

1. Overriding von Methoden ist somit eindeutig
2. Es steckt keine konkrete Implementierung dahinter
3. Durch die Reihenfolge der Vererbung

Objektorientierte Programmierung:
Inwiefern handelt es sich bei der Definition von superclasses um eine Abstraktion?

---

1. Verallgemeinerung mit weniger Eigenschaften
2. Allgemeingültige Klasse für alle Unterklassen
3. Generalisierung Richtung Oberklasse

Funktionale Programmierung:
Inline-Funktionen sind Teil der meisten funktionalen Sprachen.
Beschreiben Sie im Rahmen des Abstraktionskonzepts das Problem mit rekursiv definierten (inline-) Funktionen.

Nur 1 Rückgabewert
Beispiel:

Funktionale Programmierung:
Inwiefern kann man sagen, dass in rein funktionalen Sprachen auf einer höheren Stufe der Abstraktion programmiert wird?

Funktionen wie Werte veränderbar, daher nur Werte zu verarbeiten und keine Prozeduraufrufe vorhanden.
Beispiel:

Funktionale Programmierung:
In rein funktionalen Sprachen sind Funktionen als Parameter und Rückgabewerte von Funktionen zugelassen. Inwiefern wird dadurch eine höhere Stufe der Abstraktion erreicht, als Paradigmen bei Sprachen, die dieses Feature nicht haben?

---

1. Asynchroner Ablauf, Event-basiert / Ereignis-gesteuert
2. Dynamischer Kontrollfluss, Callbacks
3. Skalierbarkeit, da bekannter Gültigkeitsbereich

Funktionale Programmierung:
Ist es möglich, durch diese Erweiterung Probleme zu lösen, die in imperativen Sprachen nicht gelöst werden können?

Nein, trotzdem aber andere Ansätze möglich und besserer Umgang mit Parallelität

Übung 3: Paradigmen

http://people.f4.htw-berlin.de/~hebold/htw/pka/exercises/konzepte-Paradigmen.pdf

Das von-Neumann-Rechnerkonzept (auch von-Neumann-Architektur) zählt zur archetypischen Realisierung des imperativen Programmierparadigmas. Warum?

Imperative Konzept $\widehat{=}$ Befehlsorientiert
Fetch, Execute-Zyklus

Die Turing-Maschine realisiert ebenfalls das imperativen Programmierparadigma. Warum?

Jeder Zustand verknüpft über Befehle, vgl. Überführungsfunktion

Wiese wird vom von-Neumann-Rechner**konzept** aber von der Turing-**Maschine** gesprochen?

Konzept: Abstraktion
Maschine: Konkrete Idee (auch wenn so nicht realisierbar)

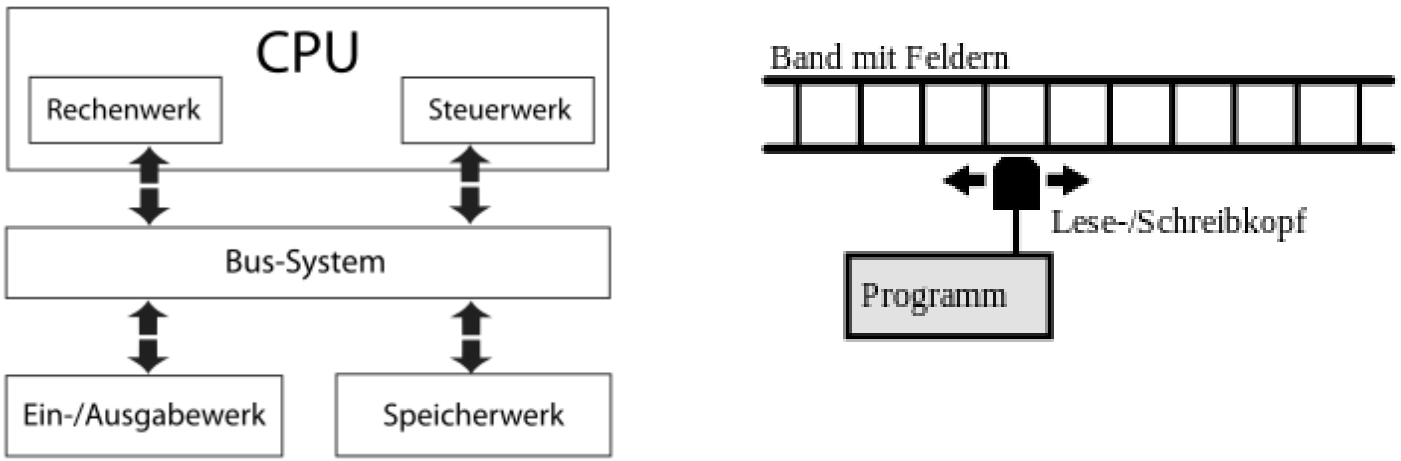

Abbildung : Von-Neumann, Turing-Maschine

Im Zusammenhang mit dem Neumann-Rechnerkonzept ist die Rede vom von-Neumann-Flaschenhals, wenn Nachteile des Konzepts genannte werden.

---

a) Was ist darunter zu verstehen?
Alle Befehle müssen durch den Bus

---

b) Gibt es eine vergleichbare Problematik für die Turing-Maschine?
Schreib/Lesekopf kann nur entweder schreiben oder lesen

Nennen Sie wenigstens einen konzeptionellen Unterschied zwischen von-Neumann-Rechnerkonzept und Turing-Maschine.

Von TM ausgehend:

1. Daten und Programme liegen **nicht** im selben Speicher
2. keine Nummerierung auf dem Band
3. keine Sprungadressen
4. kann nur 1 Feld gehen pro Befehl

Setzt die Turing-Maschine das von-Neumann-Rechnerkonzept um?

Nein, weil

1. Bei TM: Daten $\neq$ Programme
2. TM hat keine Sprungadresse

oder **Ja** mit Einschränkungen (s.o)

Wie könnte das Paradigma der strukturierten Programmierung in das von-Neumann-Rechnerkonzept integriert werden?

Überwachen, bzw. Regeln der Sprunganweisungen.
D.h. Begrenzter Bereich z.B. bei if-Anweisungen

Wieso verletzt das Konzept der lokalen static-Variablen in C das Paradigma der funktionalen Programmierung?

Funktionsausgabe nur abhängig von Eingabe. D.h. bei gleicher Eingabe gleiche Ausgabe.

```
int f(int i) {
  // Ausfuehrung bei Objekt-Init,
  // nicht bei Methodenaufruf
  static int x = 0;
  x++;
  return x+i;
}
```

Wieso verletzen Pointer in C das Paradigma der funktionalen Programmierung?

Pardigma der f. Programmierung: Funktionsausgabe nur abhängig von Eingabe. D.h. bei gleicher Eingabe gleiche Ausgabe.

```
int f(int *i) {
  // Veraendern der Speicheradresse und
  //somit der Eingabe
  *i = 1234
  ...
}
```

In Java gibt es mit dem Collection-Framework eine Reihe von sogenannten Container-Klassen. Welches objektorientierte Programmierparadigma verletzen Objekte z.B. der Klassen ArrayList oder Vector?

Es werden Referenzen gespeichert. D.h. die Datenkapselung ist verletzt.

```
Class Dummy{int value;}
...
Dummy example = new Dummy()
ArrayList<Dummy> list = new ArrayList<Dummy>()
list.add(example)

// Zugriff auf value via:
example.value
list.get(0).value
```

Wie müsste das Funktionskonzept in C beschränkt bzw. erweitert werden, damit es nicht zu Verletzungen des Paradigmas der funktionalen Programmierung kommen kann?

---

1. kein static und Pointer
2. keine Systemaufrufe

Das funktionale Programmierparadigma, das die referentielle Transparenz der Variablen fordert, wird in C durch die Zuweisung verletzt. Wie müssten die Regeln für die Verwendung der Zuweisung geändert werden, damit die Zuweisung in das funktionale Konzept passt?

Welcher Art von Anweisung entspräche die veränderte Zuweisung dann?
TODO

Die referentielle Transparenz sorgt dafür, dass Programme in rein funktionalen Sprachen problemlos nebenläufig abgearbeitet werden können. Erklären Sie den Zusammenhang an einem Beispiel. Erklären Sie an einem Beispiel den Zusammenhang fehlender referentieller Transparenz und Problemen bei nebenläufig ausgeführten Programmen.

Objektorientierte Sprachen kennen sogenannte inline-Funktionen. Wieso sind inline-Funktionen in objektorientierten Programmiersprachen implementiert? Sind inline-Funktionen in rein funktionalen Programmiersprachen sinnvoll? Welches Problem ergibt sich aus inline-Funktionen im Rahmen einer rein funktionalen Sprache?

Angenommen in C würden innerhalb von parametrisierten Makros Zuweisungen nicht mehr zugelassen. Inwiefern verletzen Makros dann trotzdem weiterhin Paradigmen der funktionalen Programmierung?

Übung 5: Induktion

http://people.f4.htw-berlin.de/~hebold/htw/pka/exercises/algorithmen-Induktion.pdf

Was bedeutet es, wenn die Verankerung bei n=a, also z.B. n=5 bewiesen wird, aber nicht für kleinere Werte?

Bewiesen erst ab n=5 und aufwärts, bzw. Voraussetzung erst ab dann beweisbar anwendbar.

Kann man aus der Allgemeingültigkeit von φ schließen, dass $\varphi(0)$ und $\varphi(n) \Rightarrow \varphi(n^+)$ gelten?

Durch 0 und den Nachfolger schließt man auf die Allgemeingültigkeit, daher ist dieser Satz falsch... hier sind Prämisse und Konklusion vertauscht.

Angenommen φ wird für 0 bewiesen, ist für ein $n = a > 0$ ungültig und $\varphi(n) \Rightarrow \varphi(n+1)$ kann wiederum gezeigt werden. Was besagt das für die Induktion?

Darf die bewiesene Verankerung im Induktionsschritt verwendet werden?

Induktionsschritt ist die Verallgemeinerung, die Verankerung wird mit konkreten Werten angewendet. Daraus folgt: Nein, darf man nicht.

Das Schema der vollständigen Induktion lautet:
Gilt für eine Menge A

1. $0 \in A$
2. $x \in A \Rightarrow x^+ \in A$

$\models \mathbb{N} \subseteq A$
oder
Gilt für ein auf $\mathbb{N}$ definiertes Prädikat φ

1. $\varphi(0)$
2. $\varphi(x) \Rightarrow \varphi(x+)$

$\models \forall x(\varphi(x))$
Geben Sie wenigstens 3 Möglichkeiten der Verallgemeinerung (Abstrahierung) an.

---

1. beliebige Startwerte, statt 0
2. mehrere Startwerte/Verankerungen
3. andere Nachfolgerfunktion
4. $\mathbb{N}$ oder Menge A kann ersetzt werden
5. Prädikat φ kann ersetzt werden

Übung 5: λ-Kalkül

http://people.f4.htw-berlin.de/~hebold/htw/pka/exercises/algorithmen-lambdaCalculus.pdf

Das λ-Kalkül unterscheidet zwei Arten von Ausdrücken: Auswertungen und Abstraktionen. Benennen Sie für jeden der Ausdrücke dessen Art und dann innerhalb des Ausdrucks gebundene Variablen und Rumpf bzw. Funktionsargument und Funktion.

1. $\lambda a.(a \quad \lambda b.(b \quad a))$
2. $\lambda x.\lambda y.\lambda z.((z\ x) \quad (z \quad y))$

Auswertung: in Klammern, hat Argumente
Abstraktion: hat *keine* Argumente, $\sim$ Funktion Achtung: Nicht verwechseln mit Rumpf, der auch in Klammern stehen kann.

1. Abstraktion, da nicht in Klammern
   Funktion: $\lambda a.f$
   Rumpf: $(a \quad \lambda b.(b \quad a))$
   Gebundene Variablen: a (was ist mit b?)
2. wie 1)
   Funktion: $\lambda x.\lambda y.\lambda z.f$
   Rumpf: $((z\ x) \quad (z \quad y))$
   Gebundene Variablen: x, y, z

Das λ-Kalkül unterscheidet zwei Arten von Variablen: gebundene und freie. Benennen Sie für jeden der folgenden Ausdrücke diese.

1. $\lambda x.\lambda y.(\lambda x.y \quad \lambda y.x)$
2. $\lambda x.(x \quad (\lambda y.(\lambda x.x \quad y) \quad x))$

Alle Variablen sind gebunden.

Werten Sie folgende λ-Ausdrücke aus:

1. $((\lambda x.\lambda y.(y \quad x) \quad \lambda p.\lambda q.p) \quad \lambda i.i)$
2. $((\lambda x.\lambda y.\lambda z((x \quad y) \quad z) \quad \lambda f.\lambda a.(f \quad a)) \quad \lambda i.i) \quad \lambda j.j)$

Der Ausdruck wird als Wert in dem/der **Symbol/Variable** ersetzt, festgelegt durch λ**Symbol**. Beachte die Klammern um den jeweilige funktion, die erst die Auswertung erlaubt.

1. $((\lambda \mathbf{x}.\lambda y.(y \quad \mathbf{x}) \quad \underline{\lambda p.\lambda q.p}) \quad \lambda i.i) \Rightarrow ((\lambda y.(y \quad \lambda p.\lambda q.p)) \quad \lambda i.i)$
   $((\lambda \mathbf{y}.(\mathbf{y} \quad \lambda p.\lambda q.p)) \quad \underline{\lambda i.i}) \Rightarrow (\lambda i.i \quad \lambda p.\lambda q.p)$
   $(\lambda \mathbf{i}.\mathbf{i} \quad \underline{\lambda p.\lambda q.p}) \Rightarrow \lambda p.\lambda q.p$
2. Lösungsweg fehlt: $\lambda j.j$

Wieso wird das abstrakteste λ-Kalkül als **typfrei** bezeichnet?

Arbeit nur auf Symbolen, reine Textersetzung

Bei λ-Kalkül–Ausdrücken wird von Auswertung und Abstraktion gesprochen. Erklären Sie an einem Beispiel, in wiefern bei λ-Kalkül–Ausdrücken abstrahiert und konkretisiert wird.

Mit welchen Programmierkonzept aus C ist das typfreie λ-Kalkül vergleichbar?

1. Makros
2. Templates
3. Funktionen
4. Pointern

Im Zusammenhang mit der Auswertung von λ-Ausdrücken kann es zu Namenskonflikten kommen, die mit Hilfe der sogenannten α Konvertierung gelöst werden. Erklären Sie an einem Beispiel die Problematik. Wie wird das Problem konkret gelöst?

Die Auswertung von Ausdrücken wird als β–Reduktion bezeichnet.
Welches Problem tritt hier auf? (Beispiel!)

Die einzige Möglichkeit im λ-Kalkül eine Wiederholung zu formulieren basiert auf dem sogenannten Fixpunktsatz. Was ist damit gemeint? Wieso lösen Fixpunkte das Problem der Wiederholung?

Tafelanachrieb?

Übung 7: λ-Kalkül - Fortsetzung

http://people.f4.htw-berlin.de/~hebold/htw/pka/exercises/algorithmen-lambdaCalculus_cont.pdf

Definieren Sie im λ–Kalkül die

1. die Null (=0)
2. die Nachfolgefunktion succ
3. eine beliebige Zahl

---

1. $\lceil 0 \rceil = \lambda x.x$
2. $\lceil n+1 \rceil = \lambda x.((x \perp) \quad \lceil n \rceil)$
3. $\lceil n \rceil$ =beliebig häufig anwenden von 2.) ?

Definieren Sie im λ–Kalkül die primitiv rekursiven Funktionen:

1. 0
2. N
3. P_n^m
4. S^{n+1}

Wie lautet der Fixpunkt von:

1. not
2. succ

Info: `https://de.wikipedia.org/wiki/Fixpunkt-Kombinator`

Definieren Sie im λ-Kalkül das primitiv rekursive Funktionsschema **R**

Beschreiben Sie den Unterschied von $=$ (Gleichheit) und $\equiv$ (Identität) an einem Beispiel.

Gleichheit: Äquivalenz; kann auch Behauptung sein und soll sich logisch ergeben.
Identität: Definition linker Seite durch rechte Seite, vgl.
$A := B, A =_{def} B$ oder hier $A \equiv B$ für Definition von B zu A.

Bei der Definition der Substitution werden Funktionen auf zwei Arten miteinander verknüpft. Nennen Sie die beiden Arten. Beschreiben Sie den Unterschied an einem Beispiel.

Komposition, Mehrstelligkeit (Currying) Currying = $(\ldots(f(a_1)a_2)\ldots a_n) = f(a_1, a_2, \ldots, a_n)$

Funktionen können durch Komposition oder Currying verknüpft werden.

1. Beschreiben Sie den Unterschied an einem Beispiel.
2. Welche der beiden Verknüpfungsarten entspricht einer Funktionsdefinition in Java?

Java: immer mehrstellige Funktionen, Gegenteil von funktionalen Sprachen