Wie Mathematik und Informatik im Unterricht voneinander profitieren können

Teil 2: Variation der Problemstellung und Modularisierung

Urs Hauser · Dennis Komm
Giovanni Serafini

Nachdem wir im ersten Teil des Beitrags die *Abstraktion* bzw. die *Variation der Darstellung* besprochen haben, fokussieren wir im zweiten Teil auf die beiden Heurismen *Variation der Problemstellung* und *Modularisierung*.

Variation der Problemstellung

Beispiel (Kombinatorik). In diesem Beispiel legen wir den Fokus auf die *Analogiebildung* als übergeordneten Problemlöseprozess. Der Grundgedanke bei der Analogiebildung besteht darin, eine unbekannte Situation zu analysieren und mit bereits Bekanntem zu vergleichen, um aufgrund gemeinsamer Strukturen vertraute Methoden und Strategien auf die unbekannte Situation zu übertragen.

Ein schachbrettartiges Straßennetz einer Stadt wird als Graph (Quadratgitter) modelliert. Die Knoten sind die Kreuzungen und die Kanten die Straßenabschnitte. Wir suchen alle Routen von A nach B, die keine Umwege beinhalten. Die Fragestellung können wir wie folgt umformulieren: *Wie viele Wege der Länge* 8 *von A nach B gibt es, die keine Umwege beinhalten?*

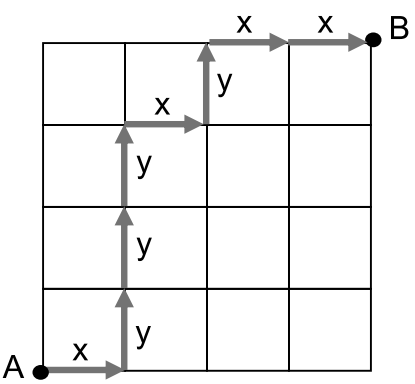

Abb. 1 Straßennetz als Graph

Wir legen das Gitter in ein Koordinatensystem und stellen einen Weg als Folge der Zeichen x für einen Schritt in die x-Richtung und y für einen Schritt in die y-Richtung dar. Der eingezeichnete Weg in Abb. 1 entspricht somit dem Wort

$xyyyxyxx$.

Das Problem der Berechnung aller Wege führen wir also auf das Problem der Berechnung aller Wörter der Länge 8 über dem Alphabet $\{x, y\}$ mit je vier x und vier y zurück. Die Analogie zur Kombinatorik besteht darin, solche Problemstellungen als *Permutationen mit Wiederholungen* zu betrachten. Die Wörter entsprechen den Permutationen der acht Buchstaben (dies sind 8!), wobei x und y aber nur je viermal auftreten können. Durch die Permutation der vier identischen Buchstaben x bzw. y ändert sich das Wort jeweils nicht. Deshalb müssen wir noch durch je 4! Wiederholungen dividieren und erhalten

$$\text{Anzahl Wege} = \frac{8!}{4! \cdot 4!} = 70.$$

https://doi.org/10.1007/s00287-019-01167-0

Urs Hauser
ETH Zürich und PH Luzern,
Zürich, Schweiz
E-Mail: urs.hauser@inf.ethz.ch

Dennis Komm
ETH Zürich und PH Graubünden,
Zürich, Schweiz
E-Mail: dennis.komm@inf.ethz.ch

Giovanni Serafini
ETH Zürich,
Zürich, Schweiz
E-Mail: giovanni.serafini@inf.ethz.ch

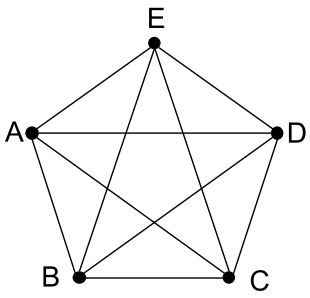

Abb. 2 Geometrische Abstraktion des Händeschüttelns

Beispiel (Kombinatorik). Es befinden sich $n = 5$ Personen auf einer Party, die sich alle zur Begrüßung die Hände schütteln. Wie oft werden die Hände geschüttelt?

Diese Fragestellung bietet sich zur Förderung der Problemlösefähigkeit an. Je nach Vorwissen kann das Problem unterschiedlich gelöst werden.

Man kann das Problem geometrisch abstrahieren und die Teilnehmerinnen und Teilnehmer als Punkte und das Händeschütteln als Verbindungsstrecken von je zwei Punkten darstellen. Übersichtshalber ordnen wir die Punkte in Form eines konvexen Fünfecks an (siehe Abb. 2).

Die Fragestellung reduziert sich nun auf die Frage, wie viele unterschiedliche Verbindungsstrecken die fünf Punkte haben können. Jeder Punkt besitzt vier Verbindungsstrecken, nämlich jeweils eine zu jedem anderen Punkt des Fünfecks. Zeichnet man die Verbindungsstrecken für jeden der Fünfeckpunkte ein, so erhält man $4 \cdot 5$ Strecken. Da wir jede Strecke doppelt gezählt haben, müssen wir die Anzahl noch halbieren und erhalten

$$\frac{4 \cdot 5}{2} = 10.$$

Man kann die obige Frage auch umformulieren. Eine Gerade ist durch zwei disjunkte Punkte eindeutig bestimmt. Wenn wir nach der Lösung suchen, müssen wir die Anzahl der Möglichkeiten bestimmen, aus fünf gegebenen Punkten je eine Zweiergruppe auszuwählen, was auf

$$\binom{5}{2} = \frac{5!}{3!2!} = 10$$

Arten möglich ist.

Man kann ein Händeschütteln auch als Wort der Länge 2 darstellen (zum Beispiel AB oder CA). Alle möglichen Wörter kann man mit einem Baumdiagramm darstellen (siehe Abb. 3). Weil der erste Knoten den Grad 5 und jeder zweite Knoten den Grad 4 hat, erhalten wir für alle Möglichkeiten $5 \cdot 4$, was man durch Zählen der Blätter einfach nachprüfen kann. Da AB und BA der gleiche Händedruck ist, können wir alle symmetrischen Lösungen streichen, und das ist genau die Hälfte. Ein Algorithmus, der die erwähnten Fälle zählt, ist leicht zu programmieren.

Modularisierung

Der modulare Entwurf ist eines der wichtigsten Konzepte in der Informatik. Bei der Entwicklung von komplexen Programmen lässt sich die Komplexität mithilfe einer Zerlegung in einzelne Module (Befehle), die individuell verifiziert werden können, verringern. Das Programm wird dadurch übersichtlicher und strukturierter.

Die Schülerinnen und Schüler können zum Beispiel erleben, wie komplexe Figuren aus einfacheren Bausteinen zusammengesetzt werden können. Sie lernen, solche Figuren zu analysieren und diese in einfache Bausteine zu zerlegen.

Beispiel (Planimetrie). Wir wollen den Flächeninhalt eines regulären n-Ecks mit Seitenlänge s bestimmen.

Das Problem wird in Teilprobleme zerlegt. Da ein reguläres n-Eck aus n kongruenten gleichschenkligen

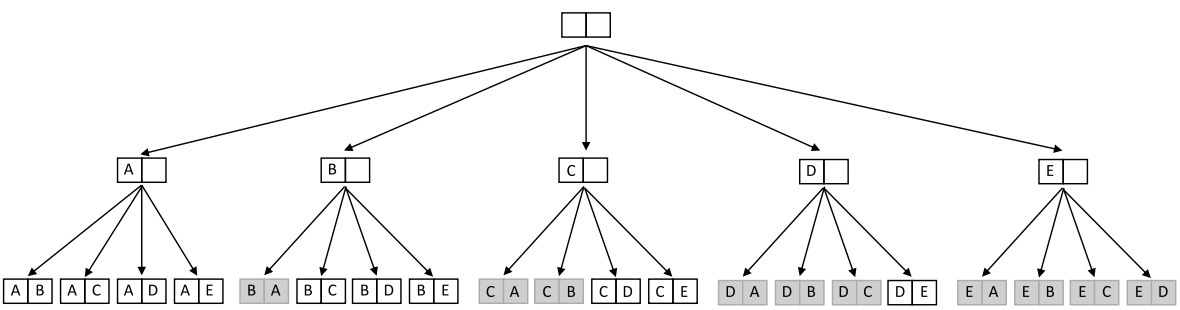

Abb. 3 Anzahl Händeschütteln als Baum

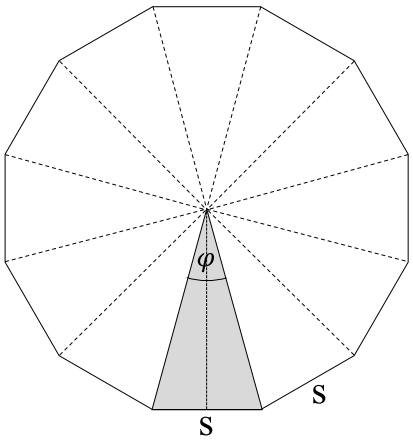

Abb. 4 Bestimmungsdreieck als Teil eines regulären n-Ecks

Dreiecken besteht, können wir die Problemstellung auf die Frage reduzieren, wie groß der Flächeninhalt des gleichschenkligen Bestimmungsdreiecks mit Basisseitenlänge s und dem spitzen Winkel $\varphi = 360°/n$ ist (Abb. 4). Diese Teilaufgabe können Lernende mit trigonometrischen Kenntnissen (ab dem 10. Schuljahr) lösen. Das Erstellen eines Netzes von Dreiecken für Flächen im 3D-Raum (Triangulation) kommt bei der computergestützten geometrischen Modellierung (CAGD) zur Anwendung.

Beispiel (Programmieren). Das Programmieren bietet eine besonders nachhaltige Möglichkeit, den modularen Entwurf zu erlernen. Im folgenden Beispiel verwenden wir TigerJython und die Turtlegrafik [2, 3].

Die Schülerinnen und Schüler sollen ein Programm schreiben, mit dem ein Stern gemäß Abb. 5 gezeichnet wird.

Die Idee beim modularen Entwurf ist, dass die Lernenden die komplexe Figur des Sterns in einfachere Bausteine zerlegen. Dazu entwerfen sie eigene Befehle, mit denen die Figur schrittweise aufgebaut

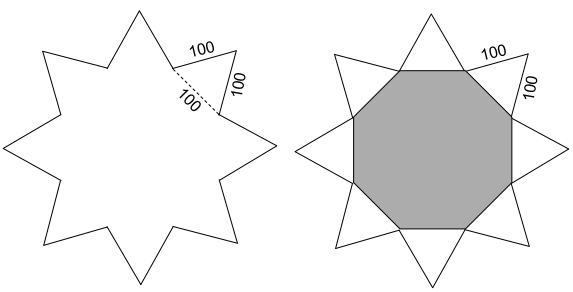

Abb. 5 Stern mit acht Zacken

und gezeichnet wird. Der Stern besteht aus acht Zacken, für die man einen eigenen Befehl `zacke()` erstellen kann.

Um den Stern zeichnen zu können, ist es wichtig, sich über die Position der Turtle vor der Ausführung des Befehls klar zu werden. Dieser führt jedes Mal exakt die gleichen Anweisungen aus. In der Ausrichtung der Turtle liegt die Herausforderung. Wir können uns die Bewegungsrichtung der Turtle im Inneren des Sterns auf einem regulären Achteck vorstellen, bei dem die Drehwinkel bekannt sind. Wir richten die Turtle nach jeder Zacke am Achteck aus. Schließlich können die Lernenden einen eigenen Befehl `stern()` kreieren, der es ihnen ermöglicht, mehrere Sterne zeichnen zu lassen (siehe Programm 1).

Programm 1. Stern mit acht Zacken

```
from gturtle import*
makeTurtle()

def zacke():
    left(60)
    forward(50)
    right(120)
    forward(50)
    left(15)

def stern():
    repeat 8:
        zacke()

stern()
```

Die Schülerinnen und Schüler haben also die Möglichkeit, die *Sprache der Turtle* mit eigenen Befehlen selbstgestaltend und kreativ zu erweitern. Diese Befehle können auch wieder innerhalb anderer Befehle verwendet werden. Sie lernen so intuitiv, was es bedeutet, strukturiert und modular zu programmieren.

In einem nächsten Schritt kann die Aufgabenstellung erweitert werden: *Zeichne einen Sternenhimmel, der aus Sternen wie in Abb. 6 in verschiedenen Größen besteht.*

Mit der Einführung von *Parametern* erreichen die Lernenden eine große Flexibilisierung, indem sie einen Befehl für verschiedene Ausgaben, abhängig vom Parameterwert, verwenden können. Die Lernenden setzen sich mit Parametern und Variablen sowohl im Informatik- wie auch im Mathematikunterricht

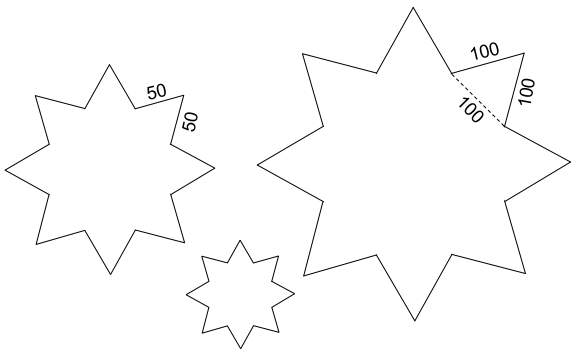

Abb. 6 Ein „Sternenhimmel"

intensiv auseinander. Die Thematisierung der Gemeinsamkeiten und Unterschiede kann in beiden Fächern zu einem tieferen Verständnis der Konzepte beitragen.

Beispiel (Folgen und Reihen). Aus Karten soll ein Kartenhaus gemäß Abb. 7 gebaut werden, wobei wir die Etagen von oben nach unten nummerieren. Wie viele Karten benötigt man für ein Kartenhaus mit n Etagen?

Die Lernenden können zunächst eine Skizze erstellen und die Karten zählen, wobei zu beachten ist, dass es auf der untersten Etage keine Bodenkarten gibt.

Wir möchten aber ein allgemeines Verfahren finden. Bezeichnen wir die Anzahl der Karten eines Kartenhauses mit $n-1$ Etagen mit $a(n)$ und beginnen mit der ersten Etage $a(1)=2$. Für ein Kartenhaus mit zwei Etagen benötigen wir die Karten der ersten Etage plus jene der zweiten Etage, also $a(1)+2\cdot 3-1$ Karten. Für ein Kartenhaus mit vier Etagen benötigt man die Karten der ersten drei Etagen plus jene der vierten Etage, also $a(3)+4\cdot 3-1$. Dabei bestimmen wir die Anzahl der Karten in der vierten Etage, indem wir von den vier vollständigen Dreiecken $(4\cdot 3)$ eine Karte subtrahieren (siehe Abb. 8). Zusammenfassend erhalten wir

$$
\begin{aligned}
a(2) &= a(1)+2\cdot 3-1\\
a(3) &= a(2)+3\cdot 3-1\\
a(4) &= a(3)+4\cdot 3-1\\
&\vdots\\
a(n) &= a(n-1)+n\cdot 3-1
\end{aligned}
$$

als Kartenanzahlen.

Ohne Kenntnis der vorangegangenen Kartenzahlen lässt sich die Summe bei diesem Ansatz nicht

Kartenhäuser und die benötigten Karten *Tabelle 1*

Etagen	Anzahl Karten
1	2
2	7
3	15
4	26

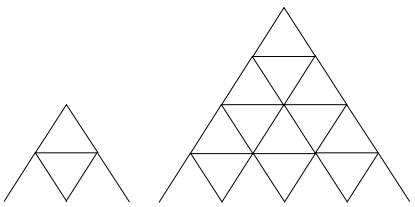

Abb. 7 Kartenhaus mit $n=2$ und $n=4$ Etagen

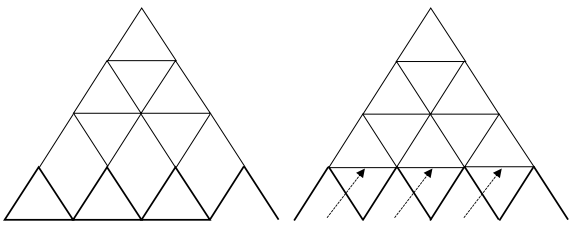

Abb. 8 Berechnung der Kartenzahl der untersten Etage

bilden. Das Problem wird also zurückgeführt auf die Berechnung der Anzahl der Karten eines Hauses mit $n-1$ Etagen etc. Das Konzept nennt man *Rekursion* und es stellt eine Möglichkeit dar, diese Aufgabe zu lösen.

Zur Rekursionsvorschrift $a(n-1)+n\cdot 3-1$ gehört zwingend die Angabe des Anfangswerts $a(1)=2$, damit die Aufgabe eindeutig lösbar ist. Wenn man die Rekursion im Programmierunterricht thematisiert, so lassen sich, neben der konkreten Programmierung, spannende Überlegungen zur Speicherorganisation machen. Programm 2 zeigt, wie die rekursive Berechnung in TigerJython umgesetzt werden kann.

Programm 2. Rekursive Funktion in TigerJyton

```
def a(n):
    if n>1:
        return a(n-1)+3*n-1
    else:
        return 2
```

Es gibt eine zweite Möglichkeit, die Folge der Kartenzahlen *explizit* zu bestimmen. Eine Etage entspricht

der Anzahl ihrer Dreiecke und wir zählen wiederum je drei Karten pro Dreieck. Für zwei Etagen sind dies $3 \cdot (2 + 1) - 2 = 7$ Karten, für vier Etagen $3 \cdot (4 + 3 + 2 + 1) - 4 = 26$ Karten.

Bei n Etagen erhält man also

$$3 \cdot \frac{n \cdot (n+1)}{2} - n = \frac{1}{2} \cdot n \cdot (3n + 1).$$

Beispiel (Bäume und Rekursion). Ein Beispiel zur Anwendung einer rekursiven Funktion ist das Erzeugen von Bäumen.

Gesucht ist ein Programm zur Erzeugung n-stelliger Zahlen mit den Ziffern 0 und 1 (siehe Abb. 9 für $n = 3$).

Die Verzweigung in Baumdiagrammen entspricht der Reduktion einer komplexen Aufgabe zu einfacheren Teilaufgaben. Die Auflistung aller Wörter der Länge n reduzieren wir auf die Auflistung der Wörter der Länge $n - 1$. Die Arbeit von rekursiven Programmen wird gerade mit Bäumen oftmals sehr anschaulich dargestellt. Die am häufigsten verwendete Struktur zur systematischen Speicherung und Verwaltung von Daten entspricht dem Baum [1]. Wie lässt sich ein Baum programmieren? In Programm 3 wird eine Liste mithilfe des rekursiven Aufrufs von `generiere(tiefe)` erzeugt [4].

Programm 3. Generieren aller 3-stelligen Binärzahlen

```
n=3
zahl=[0]*n
def generiere(tiefe):
    for i in range(2):
        zahl[tiefe]=i
        if tiefe==n-1:
            print zahl
        else:
            generiere(tiefe+1)
generiere(0)
```

Diese Aufgabe eignet sich, um das Wachstum der Lösungen mit der Stellenzahl zu thematisieren und einen Link zum Thema *Wachstumsprozesse* in der Mathematik herzustellen. Die Anzahl aller Lösungen wächst mit der Zahl n exponentiell und ist ab einer gewissen Größe nicht mehr berechenbar (für $n = 80$ sind es bereits 2^{80}, also mehr als 10^{24} Lösungen).

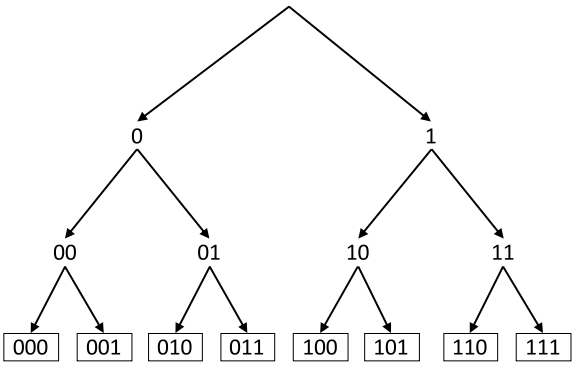

Abb. 9 Baum zur Darstellung aller 3-stelligen Binärzahlen

Als Anwendung der Generierung von Bäumen betrachten wir die folgende Aufgabe: *Wie viele Möglichkeiten gibt es, $k = 8$ (optisch nicht unterscheidbare) Bonbons auf $n = 4$ Kinder zu verteilen? Dabei ist es auch möglich, dass ein Kind oder mehrere Kinder kein Bonbon erhalten.* Die Lösungen sind Folgen von 4 Zahlen, die in der Summe 8 ergeben. Es geht also lediglich darum, wie viele Bonbons die Kinder erhalten. Dies ist das Gleiche wie die Anzahl der Möglichkeiten zu zählen, 8 in 4 Summanden zu zerlegen, wobei die Reihenfolge der Summanden eine Rolle spielt ($4 + 3 + 0 + 1$ ist eine andere Lösung als $1 + 3 + 0 + 4$).

Möchte man alle Lösungen mittels eines Suchbaums darstellen, hat man folgende Möglichkeiten: Weist man jedes Bonbon einem der vier Kinder zu, so entstünde ein 8-stufiger Baum, in dem jeder Knoten den Ausgangsgrad 4 hätte. Insgesamt hätte der Baum somit 4^8 Blätter. Der Baum wächst schnell über alle Grenzen an (bei 50 Kindern und 200 Bonbons hätte der Baum mehr als 10^{339} Blätter).

Wir verfolgen hier eine andere Strategie und entscheiden für jedes Kind, wie viele Bonbons es erhält. Jeder Knoten des 4-stufigen Baums hat einen maximalen Ausgangsgrad von 8. Eigentlich verkleinert sich dieser bei jeder Stufe, denn die Summe aller Bonbons ist ja auf 8 begrenzt.

Wie stellen wir die Lösungen mathematisch dar?

Wir tun dies hier als 4-Tupel (k_1, k_2, k_3, k_4), wobei k_i der Anzahl von Bonbons entspricht, welche das i-te Kind erhält. Eine Möglichkeit ist $(2, 3, 2, 1)$, was wir bildlich mit kleinen Kreisen und unterteilenden Wänden darstellen (siehe Abb. 10 und Abb. 11).

Nun stellen wir (k_1, k_2, k_3, k_4) wiederum als Folge von Nullen und Einsen (als Trennstriche) dar, wodurch wir beispielsweise

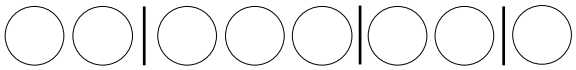

Abb. 10 Repräsentation der Folge (2,3,2,1)

Abb. 11 Repräsentation der Folge (2,3,0,3)

00100010010

erhalten, und suchen nach allen unterschiedlichen Möglichkeiten, die Zeichenfolge anordnen zu können. Für $n = 4$ Kinder und $k = 8$ Bonbons brauchen wir drei Einsen (allgemein $n - 1$) und acht Nullen. Die Anzahl der Zeichenfolgen der Länge 11 mit 8 Nullen beträgt

$$\binom{4-1+8}{8} = \binom{11}{8} = 330$$

und allgemein

$$\binom{n+k-1}{k}.$$

Ein weiteres Beispiel ist die Berechnung der Anzahl aller (maximal) 6-stelligen Zahlen (im Dezimalsystem) mit der einfachen Quersumme 9 (siehe auch Pierhöfer [4]). Hier können wir die $n = 6$ Stellen der Zahl mittels fünf Einsen trennen.

00100010001011

steht dann für die Zahl 233100 oder

11111000000000

für die Zahl 9 und wir erhalten

$$\binom{6+9-1}{9} = \binom{14}{9} = 2002.$$

Die Umsetzung in einem TigerJython-Programm erfolgt wiederum rekursiv nach dem gleichen Prinzip wie im ersten Teilbeispiel von „Bäume und Rekursion“.

Zusammenfassend können wir sagen, dass der Informatik- und Mathematikunterricht die Problemlösefähigkeit und das abstrakte Denken von Lernenden in sehr hohem Maß fördern. Man kann

Programm 4. Generieren aller 6-stelligen Zahlen mit einfacher Quersumme 9

```
n=6
zahl=[0]*n
anz=0
quers=9
d=0

def generiere(tiefe,q):
    global anz
    if tiefe==n-1:
        d=quers-q
        zahl[tiefe]=quers-q
        print zahl
        anz+=1
    else:
        d=quers-q
        for i in range(d+1):
            zahl[tiefe]=i
            generiere(tiefe+1,q+i)

generiere(0,0)
print anz
```

ferner feststellen, dass das neue Fach Informatik einen besonders großen Mehrwert im Fächerkanon darstellt.

Einerseits führt die Informatik den Lernenden die Wichtigkeit der mathematischen Methoden zur Modellierung und Verifizierung vor Augen und stärkt somit die Position der Mathematik. Zusätzlich schafft es der Informatikunterricht, neue Akzente zu setzen und den Lernenden das Lösen von Problemen als kreativen Prozess erfahrbar zu machen und ihnen die Möglichkeit zu geben, die technische Welt zu entdecken, zu verstehen und aktiv mitzugestalten.

Literatur

1. Hromkovič J (2018) Einfach Informatik – Strategien (Begleitband). Klett & Balmer Verlag, Baar
2. Hromkovič J, Kohn T (2018) Einfach Informatik – Programmieren. Klett & Balmer Verlag, Baar
3. Hromkovič J, Kohn T, Komm D, Serafini G (2016) Combining the Power of Python with the Simplicity of Logo for a Sustainable Computer Science Education. In: Proc. of ISSEP 2016. Springer, pp 155–166
4. Pierhöfer H (2009) Bäume und Backtracking, Leitprogramm, ETH