Friedrich-Alexander-Universität
Technische Fakultät

Lehrstuhl für Informatik 8 (Theoretische Informatik)

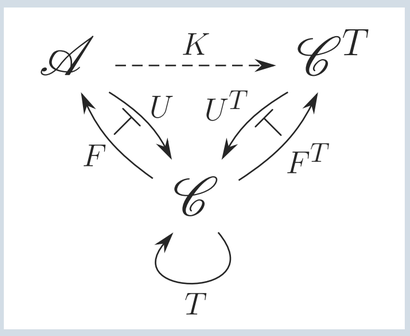
Lehrveranstaltungen WiSe 2026/27

Algebra des Programmierens

Solide und flexible mathematische Grundlagen für effektive Programmierung und Systemsemantik. Kategorien, Funktoren, Algebren und Koalgebren für

- induktive Datentypen: strukturelle Induktion und Rekursion per initiale Semantik
- Automaten & Transitionssysteme: Bisimulation und Systemverhalten per finale Semantik

Wichtig: Dualität von Prozessen und Daten!



Kategorientheorie

Aufbauend auf *Algebra des Programmierens*

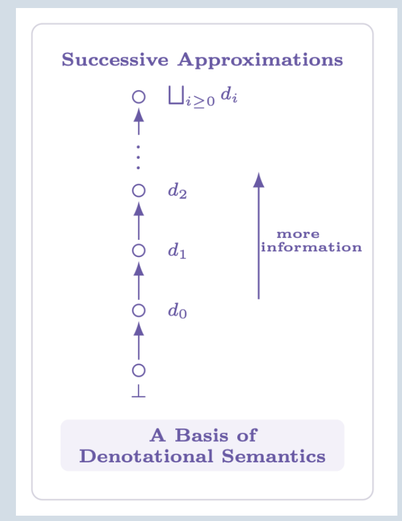
Themen:

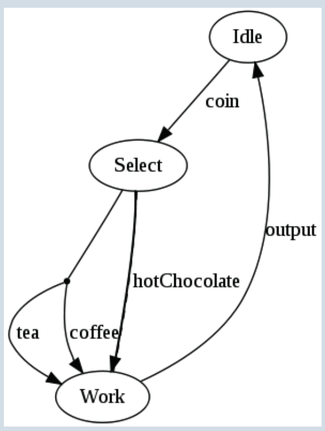
- freie Konstruktionen, universelle Pfeile und adjungierte Funktoren
- Monaden, Eilenberg-Moore- und Kleisli-Kategorien
- Initiale-Algebra/Finale-Koalgebra-Koinzidenz:
  Kanonische Fixpunkte von Funktoren zur Lösung rekursiver Domain-Gleichungen

Domain Theory

Domain theory provides mathematical foundations for information, approximation, and computation.

- partial orders, dcpos, and domains
- approximation, continuity, and least fixed points
- recursive definitions and domain equations
- applications in denotational semantics and functional programming





Theorie der Nebenläufigkeit

Nebenläufigkeit von Prozessen ist traditionell eins der zentralen Probleme der Informatik. Die Bedeutung des Phänomens nimmt in einem Zeitalter verteilter eingebetteter Systeme und massiver Verwendung mobiler Geräte weiterhin zu, aber auch der gute alte UNIX-Prozess fällt bereits in die gleiche Kategorie. In diesem Kurs werden wir Methoden kennenlernen, Prozesse semantisch fundiert zu entwerfen und gegen wohldefinierte Anforderungen wie etwa Liveness und Verklemmungsfreiheit zu verifizieren.

Verifikation von Programmen

Wie können wir uns bei einem Programm sicher sein, dass es auch wirklich korrekt ist? Wir lernen Formalismen für Korrektheitsbeweise von Programmen wie den Hoare-Kalkül kennen, und überprüfen Eigenschaften von Programmen wie Terminierung und Einhaltung von Vor- und Nachbedingungen. Wir setzen dabei einen verifizierten Compiler ein, mit dem ausschließlich beweisbar korrekte Programme kompiliert werden können. Testen ist gut, aber Beweisen ist besser!

```
// Warum ist dieses Sortierverfahren korrekt?
method Sort(a: array<int>)
    modifies a
    ensures IsSorted(a, 0, a.Length)
    ensures multiset(a[..]) == old(multiset(a[..]))
{
    for i := 0 to a.Length
        invariant ???
    {
        for j := 0 to a.Length
            invariant ???
        {
            if a[i] < a[j] {
                a[i], a[j] := a[j], a[i];
            }
        }
    }
}
// /!\ Es ist nicht Bubble-/Insertion-/Selectionsort!
```

Advanced Competitive Programming (Seminar)

Es werden verschiedene Algorithmen und algorithmische Methoden vorgestellt, wie sie im Kontext von Programmierwettbewerben zur Anwendung kommen.

https://www8.cs.fau.de