Uni Algodat Notes

Felix Pojtinger

June 10, 2021

Contents

Uni Algodat Notes

Themen

- Einleitung
  - Zyklus der Informationsverarbeitung
  - Grundbegriffe zu Algodat
  - Eigenschaften von Algorithmen
- Zeitkomplexität
  - Problemgröße des Inputs
  - Schrittzahlfunktionen
  - Testfunktionen
  - Best Case & Worst Case
  - Beispiele
    - Suche im Binärbaum
    - Fibonacci-Sequenz
    - Bubble-Sort
- Applikative Algorithmen
  - Definition Algorithmus
  - Schritte des Auswertung
  - Rekursion
- Imperative Algorithmen
  - Zustandsautomat
  - Seiteneffekt
  - Queries
  - Operations
  - Schritte der Auswertung

  - Sequenz
  - Selektion
  - Iteration
  - Church'sche These
  - Beispiele
    * Fakultät
- Sortieralgorithmen
  - Merge-Sort
    * Zeitkomplexität
  - Quicksort
    * Pivot-Element
    * Zeitkomplexität
- Grammatiken
  - Symbole
  - Produktionsregeln
  - Terminale und Non-Terminale
    * Lexene
      · Identifier
      · Literale
    * Token
      · Operatoren
      · Keywords
  - Techniken beim Compiler-Vorgang
    * Scanning
    * Parsing
    * Interpreter/Compiler
      · Scanner (lexikalische Analyse)
      · Parser (syntaktische Analyse)
    * Startsymbole
- Abstrakte Datentypen
  - ADT
  - Stacks
  - Queues
  - Binärbäume

Paradigm Conversion

I'm too stupid to write proper functional code, so most of the times I "convert" my imperative solutions to functional ones using the following schema I found on the web:

1. Isolate the loop in its own function. Make sure that all captured variables (i.e., variables that are used in the loop, but not declared in the loop) are passed in as parameters.
2. If the loop declares its own counter (e.g., in a for-loop), remove that declaration and make the loop counter another parameter.

3. Replace the loop construct itself:
    1. Replace the loop condition check with an if statement (or if expression, if that's what your language has).
    2. In the failure branch, return.
    3. Move the rest of the loop code into the success branch.
    4. At the end of the success branch, add a recursive call which passes the modified values of the loop counter and all captured variables; this is equivalent to the jump back to the top of the original loop.
4. At the original site of the loop, put in a call to your new recursive function. This is where you'll now provide the initial value of the loop counter.
5. Perform whatever other optimizations seem obvious, as the code at this point will be functional, but probably ugly, probably inefficient, and probably unidiomatic.