Informatik I

28. Ausblick

Bernhard Nebel

Albert-Ludwigs-Universität Freiburg

14.02.2014

Informatik I

14.02.2014 — 28. Ausblick

28.1 Was haben wir gelernt?

28.2 Algorithmusbegriff

28.3 Was geht nicht?

28.1 Was haben wir gelernt?

Was haben wir gelernt?

- Programmieren – jedenfalls ein bisschen
- Python-Programme lesen und verstehen
- Die Informatikperspektive auf die Welt einnehmen können
- Spaß und Begeisterung am Verstehen von (formalen und realen) Problemen
- . . . und am Finden von Lösungen (= Programme schreiben)
- Ein paar Hintergründe haben wir auch kennen gelernt,
- z.B. zur Bedeutung einer Zivilklausel – Do not mention the war!:
  `http://www.youtube.com/watch?v=yf16Lu3xQW0`
  Fawlty Towers

Was wird die Klausur von Ihnen verlangen?

- Es wird nicht schwieriger werden als in der Weihnachtsklausur – aber natürlich kommt der Stoff nach Weihnachten dazu
- Schauen Sie sich die Übungsaufgaben noch einmal an
- Auch die Folien/Aufzeichnungen sollten Sie noch einmal konsultieren
- Bei Unklarheiten: Schauen Sie in Python-Bücher und/oder stellen Sie die Fragen in den beiden angebotenen Fragestunden
- Apropos Unklarheiten: Determiniertheit vs. Determinismus von Algorithmen und Wikipedias Aussagen dazu

28.2 Algorithmusbegriff

Der Algorithmusbegriff in der Informatik-Literatur (1)

Christos Papadimitriou in *Computational Complexity*, 1994:

> *An algorithm is a detailed step-by-step method for solving a problem. But what is a problem? We introduce in this chapter three important examples. . . .*

Robert Sedgewick in *Algorithms*, 2011, 4th ed.:

> *The term algorithm is used in computer science to describe a finite, deterministic, and effective problem-solving method suitable for implementation as a computer program.*

Der Algorithmusbegriff in der Informatik-Literatur (2)

Thomas Ottmann und Peter Widmayer in *Algorithmen und Datenstrukturen*, 2011, 5. Aufl.:

> **Was ist ein Algorithmus?** *Dies ist eine philosophische Frage, auf die wir in diesem Buch kein präzise Antwort geben werden. Dies ist glücklicherweise auch nicht nötig. Wir werden nämlich in diesem Buch (nahezu) ausschließlich positive Aussagen über die Existenz von Algorithmen durch explizite Angabe solcher Algorithmen machen. Dazu genügt ein intuitives Verständnis des Algorithmenbegriffs und die Einsicht, dass sich konkret angegebene Algorithmen etwa in einer höheren Programmiersprache wie Pascal formulieren lassen. Erst wenn man eine Aussage der Art „Es gibt* **keinen** *Algorithmus, der dieses Problem löst“ beweisen will, benötigt man eine präzise formale Fassung des Algorithmenbegriffs.*

Der Algorithmusbegriff und die Turing-Maschine

- Wir
  hatten in der letzten Vorlesung die Church-Turing-These kennen gelernt:

  > *Die Klasse der Turing-berechenbaren Funktionen stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein.*

- Dies kann man verstehen als: Algorithmen (zur Berechnung von Funktionen) im intuitiven Sinne entsprechen im formalen Sinne Turing-Maschinen!
- Aber was sind Turing-Maschinen?
- Turing-Maschinen bestehen aus einem Ein-/Ausgabe-Band, einem Lese-/Schreib-Kopf und einer Zustandsmaschine
- Beispiel: `http://www.youtube.com/watch?v=gJQTFhkhwPA`
- An was erinnert Sie dieses Herumfahren und Ändern von Symbolen auf dem Band?

Algorithmen im engeren und weiteren Sinne

- Algorithmen entsprechen also Turingmaschinen,
- wobei man zeigen kann, dass höhere Programmiersprachen und Turingmaschinen i.w. äquivalent sind.
- Man kann und hat viele Bedingungen gelockert, ohne den Bereich der Turing-Berechenbarkeit zu verlassen:
  - Ein Algorithmus muss nicht immer einen festgelegten Nachfolgezustand / nächsten Schritt haben, sondern die Entscheidung über den nächsten Schritt kann gewürfelt werden (randomisierter Algorithmus).
  - In einem Algorithmus können mehrere nächste Schritte möglich sein, wobei der ausgewählt wird, der zum Erfolg führt (nicht-deterministischer Algorithmus).
  - Bedingt durch randomisierte oder nicht-deterministische Entscheidungen kann ein Algorithmus verschiedene Werte bei gleichen Eingaben ausgeben (im Falle von Suchproblemen, z.B. CSP, oder auch fehlerhaft sein – bei Monte-Carlo-Algorithmen).

Zurück zum Anfang: Algorithmuscharakterisierung am Anfang

Vorschrift zur Durchführung einer Berechnung (Folge von Einzelschritten) mit folgenden Eigenschaften:

Effektivität	Jeder Einzelschritt ist ausführbar.
Determiniertheit	Der nächste Einzelschritt ist stets festgelegt.
Finitheit	Die Vorschrift ist endlich.
Terminierung	Die Berechnung endet nach endlich vielen Einzelschritten – für alle legalen Eingaben.
Generalität	Die Vorschrift kann eine ganze Klasse von Problemen lösen.
Präzision	Die Bedeutung jedes Einzelschritts ist eindeutig festgelegt.

Welche Eigenschaft ist wirklich notwendig?

- Effektivität: Wenn wir einen Schritt nicht effektiv ausführen können, ist es keine Berechnungsvorschrift. Bei einer TM immer klar!
- Determiniertheit: Hier hätte man vielleicht besser *Determinismus* geschrieben, da der Begriff *Determiniertheit* in Wikipedia und anderen Einführungen für eine andere Eigenschaft benutzt wird, nämlich dass Algorithmen Funktionen berechnen.
- Finitheit: Wenn ein Algorithmus eine unendliche Beschreibung haben könnte, könnten wir alle Funktionen über den natürlichen Zahlen berechnen (durch Angabe der Funktionstabelle).
- Terminierung: Endet die Berechnung nicht in endlicher Zeit, können wir das Ergebnis nicht erfahren.
- Generalität: Ist plausibel. Nicht-generelle Algorithmen sind aber einfach nur trivial.
- Präzision: Natürlich muss jeder Schritt klar sein (ist bei einer TM gegeben und fällt eigentlich mit Effektivität zusammen).

Determinismus vs. Determiniertheit

In der englischen Literatur (und der englischen Wikipedia) gibt es die Unterscheidung nicht. In der deutschen Wikipedia finden wir:

> *Darüber hinaus wird der Begriff Algorithmus in praktischen Bereichen oft auf die folgenden Eigenschaften eingeschränkt:*
> *1. Der Algorithmus muss bei denselben Voraussetzungen das gleiche Ergebnis liefern (Determiniertheit).*
> *2. Die nächste anzuwendende Regel im Verfahren ist zu jedem Zeitpunkt eindeutig definiert (Determinismus).*

Das ist völliger Quatsch! Das hat nichts mit praktischen Bereichen zu tun, sondern es handelt sich um die Beschränkungen der engen Sichtweise von Algorithmen.

Weiterer Unsinn

Deutsche Wikipedia:

> *[J]eder deterministische Algorithmus determiniert, während aber nicht jeder determinierte Algorithmus deterministisch ist. So ist Quicksort mit zufälliger Wahl des Pivotelements ein Beispiel für einen determinierten, aber nicht deterministischen Algorithmus, da sein Ergebnis bei gleicher Eingabe und eindeutiger Sortierung immer dasselbe ist, der Weg dorthin jedoch zufällig erfolgt.*

- Random-Quicksort ist nicht nicht-deterministisch sondern randomisiert, also kein Beispiel!
- Außerdem ist Random-Quicksort nicht stabil, d.h. auch nicht determiniert! Es erzeugt zwar immer eine sortierte Sequenz, aber die Elemente mit gleichen Schlüssel können in verschiedener Reihenfolge auftauchen.

28.3 Was geht nicht?

Nicht-Berechenbarkeit und Unentscheidbarkeit

- Ein formaler Algorithmusbegriff ist vor allem deswegen wichtig, weil wir ja auch Unmöglichkeitsergebnisse beweisen wollen.
- Was können wir also z.B. nicht berechnen/entscheiden?
- Gegeben ein beliebiges *Python-Programm* Π, gibt das Programm Π bei jeder Eingabe nach endlicher Zeit ein Ergebnis aus?
- Dies Problem (wie auch jedes ähnliche Problem) ist **unentscheidbar**!
- D.h. wir können **kein** Programm schreiben, um dieses Problem zu entscheiden („ja“ oder „nein“ ausgeben)
- Dies sind alles Dinge, die erst in Info III behandelt werden ...

Fazit

- Python ist eine wirklich tolle Programmiersprache!
- Monthy Python ist eine Comedy-Truppe, die nicht zu überbieten ist.
- . . . und da gibt es die 10 Spitzen-Sketches, wobei ich speziell auch bei dem ersten Platz zustimme:
  `http://www.youtube.com/watch?v=XCnK3B7Ftow`