Einführung in die Informatik I (autip)

Dr. Stefan Lewandowski

Fakultät 5: Informatik, Elektrotechnik und Informationstechnik
Abteilung Formale Konzepte
Universität Stuttgart

24. Oktober 2007

Was Sie bis hier her gelernt haben sollten

- Informatik ist mehr als das, was mit Computern zu tun hat
- eine ungefähre Vorstellung der Begriffe Algorithmus und Programm
- der Softwareentwicklungsprozess umfasst weit mehr als nur die Implementierung eines Algorithmus in einer Hochsprache auf dem Rechner
- eine ungefähre Vorstellung der grundlegenden Konzepte von Programmiersprachen
  - Variablen, Typen, Blockkonzepte (bisher in Form von Unterprogrammen)
  - Kontrollstrukturen (Fallunterscheidung, for- und while-Schleifen)
  - Rekursion

Was Sie bis hier her gelernt haben sollten

Mit diesem Wissen sollten Sie bereits in der Lage sein, kurze Programme zu verstehen und diese durch copy-&-modify-&-try-&-error zielgerichtet (d.h., kein blindes Raten) an ähnliche Aufgabenstellungen anzupassen.

Mit dieser ungefähren Vorstellung im Gepäck werden wir nun nochmal an den Anfang zurückgehen und die Begriffe etwas formaler und genauer fassen.

Algorithmen und Programme

Die Definition „Informatik“ als Ganzes aus dem Duden Informatik ist für den Moment hinreichend genau. Die Begriffe Algorithmus und Programm wurden bis jetzt jedoch nur umgangssprachlich verwendet.

Definition Algorithmus und Programm

Was zeichnet einen Algorithmus aus?

- Verarbeitungsvorschrift
- präzise formuliert
- kann von jedem ohne weitere Erläuterungen durchgeführt werden
  - je nach Kontext kann zur Durchführung noch weiteres Wissen notwendig sein – dieses ist dann aber eindeutig und nicht spezifisch für den Algorithmus, z.B. „bilde zu f(x) die erste Ableitung f'(x)“
- enthält der Algorithmus umgangssprachliche Elemente, so müssen diese eindeutig interpretierbar sein
- Endlichkeit der Darstellung

Dies sind für den Moment die wichtigsten Eigenschaften (weitere Forderungen und Eigenschaften sind eher von theoretischem Interesse).

Ein paar Anmerkungen dazu

Zu Punkt 4: Kochrezepte haben i.A. nicht die Eigenschaft eines Algorithmus („eine Prise Salz“, „ein wenig Öl“, . . .).
Zu Punkt 5: Ein Beispiel:

1. merke dir die Zahl 0
2. gebe die gemerkte Zahl aus
3. erhöhe die gemerkte Zahl um 1 und merke dir nun diese (und nur diese) Zahl
4. gehe zu Schritt 2

ist ein Algorithmus zur Ausgabe der natürlichen Zahlen.

1. gebe die Zahl 0 aus
2. gebe die Zahl 1 aus
3. gebe die Zahl 2 aus
4. ⋮

ist hingegen kein Algorithmus.

... und die Laufzeit?

Auf die Frage, ob ein Algorithmus nur endlich lange läuft, kommen wir in einigen Wochen zurück. In der Praxis ist man in der Regel an Algorithmen interessiert, die für jede Eingabe nach endlich vielen Schritten die Berechnung abgeschlossen haben (Ausnahmen sind z.B. Betriebssysteme oder Steuerungssysteme).

Umgangssprachlich können wir Algorithmus mit dem Begriff „eindeutiges Kochrezept“ übersetzen.

Im Allgemeinen gibt es für ein gegebenes Problem mehr als einen Algorithmus (sogar unendlich viele).

Was unterscheidet ein Programm von einem Algorithmus?

- eindeutiger Formalismus einer Programmiersprache
- Bezug auf bestimmte Darstellung der verwendeten Daten
- Schnittstellen zu anderen Programmen (z.B. Betriebssystem) und Hardware
- Ausführbarkeit auf einem Computer

Mit der Umsetzung eines Algorithmus in ein auf einem Computer ausführbaren Programm kommen in der Regel noch die beiden folgenden Eigenschaften bzw. Einschränkungen dazu

- Näherungen (z.B. bei reellen Zahlen)
- endlicher Speicher

Programm vs. Algorithmus

Ein Algorithmus kann in verschiedene Programme umgesetzt werden (verschiedene Programmiersprachen, verschiedene Betriebssysteme, verschiedene Computerhardware). Ein Algorithmus ist somit die abstrakte Formulierung aller Programme, die ihn beschreiben.

Aspekte von Programmen

- lexikalische Einheiten (Bezeichner, Literale, reservierte Wörter, Begrenzer, Trennzeichen, Kommentare)
- Variablen-, Typ-, Block-Konzepte
- Datenstrukturen (integer, boolean, character, ...)
- Operatoren (+, -, *, /, mod, and, or, &, ...)
- Kontrollstrukturen (if, for, while, ...)
- Semiotik (Lehre von Zeichen, Zeichensystemen und Zeichenprozessen)
  - Syntax (formaler Aufbau)
  - Semantik (Bedeutung)
  - Pragmatik (Wechelwirkung mit der „Außenwelt“)

Zur Einführung der Datenstrukturen, Operatoren und Kontrollstrukturen werden wir die Syntax und Semantik definieren. Dazu benötigen wir insbesondere formale Beschreibungsmöglichkeiten für Syntax.

Sprachen zur Beschreibung der Syntax von Sprachen

Der wesentliche Aufbau von Programmen in Ada 95 (und der meisten anderen Hochsprachen) lässt sich relativ leicht beschreiben. Wir stellen hier zwei Möglichkeiten vor

- (E)BNF - (Erweiterte) Backus-Naur-Form
  - wird z.B. auch im Ada Reference Manual verwendet
- Syntaxdiagramme
  - sind etwas anschaulicher, aber in der Praxis auch etwas umständlicher zu handhaben

Hier sollen zunächst nur die wenigen Bausteine der EBNF und der Syntaxdiagramme vorgestellt werden, so dass wir damit die Syntax von Ada 95 formal beschreiben zu können. Welche weitergehenden Eigenschaften die EBNF und Syntaxdiagramme haben, was damit beschrieben werden kann und wo die Grenzen der Möglichkeiten liegen, werden wir später im Semester aufgreifen. Für den Moment reicht erst einmal das Wissen, dass alles, was mit EBNF dargestellt werden kann, auch mit Syntaxdiagrammen möglich ist und umgekehrt.

Einschub zur Motivation „Wozu Formalismen?“
$\rightsquigarrow$ Die Rollenspiel-Metapher

- Programmierer: schreibt Programme (in einer Hochsprache) – hier: Sie!
- Ausführer: – hier: AdaLogo bzw. Übersetzer von Ada 95 (+ Windows/Linux)
  - prüft die Programme auf syntaktische Korrektheit (d.h., er prüft, ob das Programm ein für den Rechner formuliertes „detailliertes, eindeutiges Kochrezept“ ist)
  - führt das Programm (falls es syntaktisch korrekt ist) Schritt für Schritt nach dem „Kochrezept“ aus
- Benutzer: – hier: Sie! (und bei den Übungsaufgaben der Korrektor)
  - lässt den Ausführer Programme ausführen
  - ist dabei für Eingaben und Interpretation der Ausgaben zuständig

Notwendige Kenntnisse für den Programmierer

- Syntax und Semantik der Programmiersprache
- Problemstellung, welche Eingaben sollen zu welchen Ausgaben verarbeitet werden

Notwendige Kenntnisse für den Ausführer

- Zur Überprüfung, ob es sich um ein Programm handelt: Syntax der Programmiersprache
- Voraussetzung: Eindeutige Beschreibungen $\rightsquigarrow$ Formale Darstellung $\rightsquigarrow$ Erweiterte Backus-Naur-Form (EBNF), Syntaxdiagramme
- Zur Ausführung des Programms: Semantik der Programmiersprache

Notwendige Kenntnisse für den Benutzer

- Keine, wenn das Programm entsprechend geschrieben ist

Das Programm aus Sicht des Ausführers

- Eingabe für den Ausführer: hier: eine Textdatei mit dem Programm in AdaLogo bzw. Ada 95
- 1. Aufgabe: Analyse des Programms, Zerlegung des Textes in Lexikalische Einheiten
- 2. Aufgabe: Überprüfung auf formal korrekten Aufbau
- ggf. Rückmeldung an den Programmierer $\rightsquigarrow$ dieser muss sich über die lexikalischen Einheiten bewusst sein, um die Fehlermeldungen des Ausführers verstehen zu können
- 3. Aufgabe: Ausführen des Programms