Überblick über die Lehrveranstaltung

„Einführung in die Informatik I“

im WS 1997/98; Dozent: Lagally

Begleitlektüre: Ludewig & Appelrath, „Skriptum Informatik“, empfohlen

- 14.10.1997: Einführung: was ist Informatik? Organisatorisches
- 16.10.1997: einige Grundbegriffe
  - was ist eine *Definition*: Sprachregelung mit Gültigkeitsbereich
  - *Information* und *Nachricht*
  - *Sprache*: *Syntax* und *Semantik*
  - *Abstraktion*: Unwesentliches weglassen
  - Hausaufgabe: Kapitel „Rechner“ selbst durchlesen
- 21.10.1997: Mengen
  - etwas naive *Mengenlehre*
    - endliche Mengen, Grundoperationen
    - unendliche Mengen: Beispiel natürliche Zahlen
  - *Zeichenvorrat* = *Alphabet* (synonym, ohne Anordnung)
  - *Wörter*, Wortmengen, *freies Monoid*, leeres Wort
  - Halbgruppe, Monoid, *Homomorphismus*
  - *Codierung*, Umkehrbarkeit, Fano-Bedingung, Binärcodierung
  - *formale Sprache* = Wörtermenge
    - Aufzählbarkeit, Entscheidbarkeit
    - Problem: endliche Beschreibung für unendliche Menge
  - Algorithmus: informell
- 23.10.1997: Grammatiken
  - *Chomsky-Grammatiken*, Klassen
  - Klassen 2 und 3: deute NT als *Wortmenge*
  - Chomsky-Grammatiken als Erzeugungsverfahren, *Ableitung*
    - Determinismus, Nichtdeterminismus
    - Entscheidungsverfahren: rate und verifiziere Ableitung

- 28.10.1997: endliche Automaten
  - *Algorithmus*, Definition
  - *endlicher Automat* am Beispiel Zigarettenautomat
    - *Übergangsdiagramm* und *Übergangstafel*
    - Ausgabe: hier nur kursorisch
  - Sprache der *zulässigen Eingaben*: regulär
  - Grammatik aus Tafel und umgekehrt (Beispiel)
  - duale Darstellung: vergiß Zustände, markiere Pfeile
  - *Syntaxdiagramm*, beschreibt Wortmenge; vorerst nichtrekursiv
    - rekursives Einsetzen: → Chomsky 2
    - Abarbeitung: Stapel von Kopien → *Stack-Prinzip*
    - *Kellerautomat*
- 30.10.1997: Funktionen
  - *Funktion* als statische Zuordnung zwischen Mengen
    - Definitionsbereich, Wertevorrat
    - totale und partielle Funktionen
  - Mengen von Funktionen: $\{M \to N\}$
  - λ-Notation: $\lambda x.f(x)$ anonyme Funktion
    - formale Parameter, *Bindung*, lokale Umbenennung
    - weitere Bindungsbeispiele:
      $\{x|E(x)\}$, $x : f(x) = 0$, $\int f(x)\,dx$, $\forall x A(x)$,
      ⌈**var** x: integer; x :=1⌋
- 04.11.1997: Turing-Maschine
  - Algorithmus realisieren durch geeignete *Maschine*
  - *Turing-Maschine*: einfach und allgemein; ineffizient
    - Marsfahrzeug "Pathfinder" als Modell für Turingmaschine
    - formale Definition der TM
  - Beispiel: Subtraktion in *Strichdarstellung*
    - *Übergangsfunktion* hier als Matrix geschrieben
    - weitere Beispiele
  - *Universalität* (behauptet), *Church*-sche These

- 06.11.1997: Programmiersprache
  - Programmieren = *Problemlösen*
    - Problem erkennen, analysieren
    - Lösungsweg suchen, formulieren, realisieren
    - Lösung berechnen
  - *Übersetzen* von Programmiersprachen: *bedeutungsgleich*!
    - *Analysephase*: lexikalisch, syntaktisch, semantisch
  - Einstieg in MODULA 2
  - Scanner: *reguläre Ausdrücke*, Syntaxdiagramme, endl. Automat
  - MODULA: "letter", "number", "string"
    - geschachtelte Kommentare: endlicher Automat genügt nicht!
    - *Kellerautomat*; hier: *Zähler*
- 11.11.1997: Problemlösen
  - Problemlöse-Strategien
    - einfache, kleine Probleme: direkt lösen
    - strukturierte Probleme: zerlegen, Lösungen zusammensetzen
    - *Rekursion*: Teilproblem ist Instanz des ganzen Problems, aber in wohldefiniertem Sinne kleiner: Terminierung ist klar!
  - MODULA: "module", "import", "block", "statement", "call"
    - *Kontextbedingungen*
  - einfaches Beispiel für komplettes Programm (2.1 aus L & A)
  - *Objekt*: Ausgabefläche, potentiell unendlich (aus *Bibliothek*)
    - *Attribut*: Schreibposition
    - *Methoden*: Writeln, WriteString(s: String)
    - Zugriff über *Bibliotheksmodul* InOut
- 13.11.1997: Datentypen, Klassen
  - MODULA: Zahlen, Zeichenketten
  - *Datentyp*: Menge von Werten mit definierten Operationen
  - andere Sprechweise: *Klasse*:
    - Objekte *(Instanzen)* mit Attributen und Methoden
  - Ausblick auf OOP: *Vererbung*

  - MODULA: “integer”, “cardinal”, “real”, “boolean”, “char” jeweils mit definierten Operationen
    * einige *Transferfunktionen*
    * *Überlagerung* bei abs(x)

- 18.11.1997: Ablaufsteuerung
  - MODULA: benannte Konstanten, Gültigkeitsbereich!
  - MODULA: arithmetische Ausdrücke
  - MODULA: Fallunterscheidung mit “case”
  - MODULA: Relationen
  - MODULA: Fallunterscheidung mit “if”, “then”, “else”, “elsif”
  - *Struktogramme*

- 20.11.1997: Syntax-Beschreibungsmechanismen
  - Mechanismen zur Notation von Syntaxregeln: bisher bekannt:
    * Chomsky-Grammatiken
    * Syntaxdiagramme (einfach und auch rekursiv)
  - zusätzlich:
    * BNF (ALGOL 60)
    * *EBNF*: Beziehung zu Syntaxdiagrammen; *reguläre Ausdrücke*

- 25.11.1997: Rekursion; Terminierung
  - *Funktionen*: *statischer* Zusammenhang vs. Algorithmus
    * *berechenbar*: Algorithmus existiert (muß nicht sein!)
  - MODULA: Syntax Funktionsdeklaration
  - MODULA: Syntax Funktionsaufruf
  - Beispiel: Stellenwertdarstellung natürlicher Zahlen
    * Zifferndarstellung zur Basis 16: HexDigit(n: cardinal): char
    * *Vorbedingung*: $0 \leq n < 16$
    * *Nachbedingung*: result $\in \{'0'..'9','A'..'F'\}$
    * definieren die *Korrektheit*; nicht notwendig abgeprüft!
  - Beispiel: bestimme die notwendige Stellenzahl
    * rekursiver Ansatz: Teilproblem ist kleiner!

  - Überlegung zur Korrektheit:
    * benutze den Satz: „jede nichtleere Menge natürlicher Zahlen hat ein kleinstes Element“
    * zeige, daß kein *„kleinster Verbrecher“* existiert, sonst gäbe es einen noch *echt* kleineren
    * das ist ein *versteckter Induktionsbeweis*!
    * das Beweisschema gilt für *alle* rekursiven Programme!
    * man muß nur jeweils eine geeignete *Problemgröße* definieren
  - Einpacken in Bibliotheksmodul: *Schnittstellenbeschreibung*
  - Implementierung kommt erst später.

- 27.11.1997: Variablen
  - Beispiel: Ausgabe einer Hexzahl in ein Feld gegebener Breite, führende Blanks
    * 2 rekursive Aufrufe; nur einer mit *Zählschleife* (ad hoc)
  - Hauptprogramm zur Hex-Umrechnung
  - *Variablen* für Eingabewerte; ReadCard(x)
  - *Modul-Mechanismus* in MODULA: (mit Versions-Überprüfung)
    * “definition module”, “implementation module”,
  - *statische Variablen* im Hauptprogramm
  - *lokale Variablen* in Prozedur
  - *Inkarnationen*, Laufzeitorganisation, *Aktivierungsblöcke*
  - *Gültigkeitsbereich* und *Lebensdauer*
  - *Referenzparameter*

- 02.12.1997: Rekursion
  - Beispiel: ggt(p, q) rekursiv
    * Problemgröße: max(p, q)
  - einige Mechanismen zur Erzeugung neuer Typen: Aufzählungen, Unterbereiche; *skalare Typen* allgemein
  - Beispiel: Türme von Hanoi
    * Regel: bei rekursiven Prozeduren *niemals* den Aufrufbaum betrachten, sondern inneres Exemplar als *black box*!
    * *Problemgröße* jeweils geeignet definieren (natürliche Zahl!)

- 04.12.1997: Zeit- und Speicheraufwand
  - *Aufwandsabschätzung* für Türme von Hanoi
    - Exkurs über *Differenzengleichungen*
    - $\mathcal{O}(f(n))$-Notation (informell)
  - Beispiel: Fibonacci-Zahlen (später in *vielen* Versionen!)
    - rekursiver Ansatz:
    - Aufwand: exponentiell; abschätzen (auch exakte Lösung)
- 09.12.1997: dynamische Programmierung; Tabellen
  - Fibonacci: Definitionsmodul, exportiert MaxArg = 24
  - *Zusicherungen* in der Schnittstelle
  - *Implementierungsmodul*, rekursiv
  - bessere Lösung: *dynamic programming* (Terminus erklären)
    - Prinzip: berechne alle benötigten Teilprobleme in einer geeigneten Reihenfolge, jedes nur einmal; hier: von unten nach oben
    - hier gibt es für die zulässigen Eingabewerte nur endlich viele jemals benötigte Teilprobleme: vorweg berechnen und merken
  - ablegen in *Tabelle*: Array-Konzept
    - Regel: Grenzen bei Indizierung *immer* prüfen, falls man die Einhaltung nicht *statisch* zeigen kann!
  - MODULA: "array"-Syntax
- 11.12.1997: Arrays, Schleifen
  - Beispiele für Array: Verkehrsampel, Dreiervektoren, Matrix als Array von Zeilen; Notation für *Mehrfach-Indizierung*
  - Fibonacci: Ablage der Werte in Tabelle, vorbesetzen bei der *Initialisierung* des Moduls; bei Funktionsaufruf nachsehen: $\mathcal{O}(1)$
  - MODULA: *"while"-Schleife*
    - Fibonacci: berechne Werte von unten nach oben, mit Hilfsfeld
    - Fibonacci: berechne Werte von unten nach oben, ohne Hilfsfeld, mit Umspeichern
  - MODULA: weitere Schleifenkonstrukte: "repeat", "loop", "exit"
  - *Struktogrammdarstellung*

- 16.12.1997: abstrakte Datentypen
  - Frage: woher kommt der Effizienzgewinn der iterativen Lösung? Die Erklärung im Buch (Verwendung von Variablen) stimmt nicht!
    - Idee: benachbarte Werte zusammenfassen
  - Konzept: *abstrakter Datentyp*:
    - Menge von Werten
    - Satz von Operationen
    - Regeln über das Verhalten
    - Realisierung braucht nicht bekannt zu sein!
  - Arten von Moduln (oder *Kapseln*):
    - Funktionsmodul (*keine* internen Objekte außer Konstanten)
    - Datenmodul: abstraktes Objekt *(interner Status)*
    - Typenmodul: Menge von abstrakten Objekten *gleichen Typs*
  - Beispiel: ADT Zahlenpaar; Operationen Teil1, Teil2, Komb
    - *Term-Algebra* ist treues Modell
    - Fibonacci: Annahme: Wert vom Typ Paar kann Funktionsresultat sein (stimmt leider nicht in MODULA!)
    - Rekursionsgleichung für benachbarte Paare ist *einstufig*
    - rekursive Lösung hat linearen Aufwand!
  - Ausweg in MODULA: Resultate als *var-Parameter*; Hilfsvariablen nötig, Eleganz der funktionalen Schreibweise geht verloren
    - intern wird das ohnehin so gemacht, aber dafür sind eigentlich Compiler da!
    - es ist immer noch derselbe Algorithmus, daher der lineare Aufwand!
- 18.12.1997: Prozedurtypen
  - *Prozedurtypen* in MODULA
    - Beispiel: numerische Integration, Rechteckregel, Trapezregel
- 23.12.1997: Wiederholung
  - Gültigkeitsbereich und Lebensdauer:
  - großes Beispiel aus L & A
  - (abfällige) Bemerkungen zum Programmierstil im Beispiel

- 08.01.1998: Sets, Arrays, Strings
  - *Komplexe Datentypen* in MODULA:
  - *Sets*, BITSET, INCL, EXCL
  - *Arrays* (schon teilweise besprochen)
  - Verwendung von *Zeichenreihenkonstanten*
- 13.01.1998: Records
  - Komplexe Datentypen in MODULA:
  - *Records*, *Selektoren*
    - "with"-Statement
  - *Varianten*
  - *Gültigkeitsbereiche* bei Records
  - Beispiel: Verwaltung von Studentendaten (wird nicht fertig)
    - Datentypen, Verbunde für Datum und Personendaten
    - *Vergleichsoperationen* für Datum und Namen
    - Resultattyp: VglTyp = (kl, gl, gr)
    - *lexikographischer Vergleich* von Strings
- 15.01.1998: Pointers
  - *Dynamische Speicherverwaltung*
  - *Zeigertypen* frei eingeführt
  - *Halde*: ALLOCATE, DEALLOCATE, NEW, DISPOSE
  - Hinweis: Zeiger sind *keine festen Adressen*! Arithmetik unzulässig, nur Zuweisung, Vergleich und *Dereferenzierung* sind definiert
  - Begriff *Garbage-Collection* (keine Algorithmen)
- 20.01.1998: lineare Listen
  - ADT *Liste*
    - funktionale Spezifikation
    - Term-Algebra als Modell
    - Definitionsmodul
  - *opaker Typ*; motiviert mit festem Platzbedarf
  - Listen-Realisierung als *Geflecht*, rekursiv

    - ∗ *kopierende* Realisierung
    - ∗ iterative Realisierung, *Schleppzeigertechnik*
    - ∗ Kopieren mit *Reißverschlußtechnik*

- 22.01.1998: Stacks
  - Grundoperationen mit linearen Listen
  - eigene *Freilistenverwaltung*
  - Standard-Typschema *Stack* (generisch!)
    - ∗ funktionale Spezifikation
    - ∗ Term-Algebra als Modell
    - ∗ Realisierung als *verkettete Liste*, funktional
    - ∗ dasselbe prozedural mit *Durchgangsparameter*
  - welche Operationen mit verketteten Listen gehen mit $\mathcal{O}(1)$?
- 27.01.1998: Binärbäume
  - Standard-Typschema *Binärbaum* (generisch!)
    - ∗ funktionale Spezifikation
    - ∗ Term-Algebra als Modell
    - ∗ Realisierung als Geflecht
    - ∗ *Durchwanderung*: prefix, infix, postfix
  - Beispiele: *Strukturbaum* einer Formel
    - ∗ prefix gibt Funktions-Termdarstellung
    - ∗ infix gibt Infix-Notation
    - ∗ postfix gibt Operationenfolge für *UPN-Rechner*
- 29.01.1998: Suchbäume
  - Standard-Typschema *Suchbaum* (generisch!)
    - ∗ funktionale Spezifikation
    - ∗ Term-Algebra als Modell
    - ∗ Infix-Durchwanderung gibt sortierte Folge
    - ∗ *Entartungsfall*: *Balancierung* nötig; keine Algorithmen
    - ∗ Realisierung als Geflecht
  - *Suchen*, rekursiv
  - *Entrekursivierung* ad hoc

- 03.02.1998: Schlangen
  - Standard-Typschema *Schlange* (generisch!)
    - funktionale Spezifikation
    - Term-Algebra als Modell
    - Realisierung als Geflecht, prozedural
- 05.02.1998: endliche Schlange; Binärsuche
  - Schlange endlicher Maximallänge
    - Realisierung als *Ringliste* im Array
    - Realisierung als Geflecht, prozedural
  - *Binäres Suchen* in Tabelle, allgemeines Suchschema
- 10.02.1998: Suchen, allgemein
  - *Suchen* in allgemeinen Datenstrukturen *(Zustandsmodell)*
    - rekursiver Ansatz
    - iterativer Ansatz
  - *Aushängen* aus einem Suchbaum
- 12.02.1998: Ergänzungen
  - allgemeine Diskussion: Datenstrukturen mit *Bearbeitungszustand* (*Aufsetzpunkt*, vgl. "cursor" in Eiffel)
  - Darstellung *allgemeiner Bäume* und *Wälder*
    - *äquivalenter Binärbaum*, Durchwanderungen analog
  - *Gerichtete Graphen*; Darstellungen:
    - Liste von Knoten und Kanten
    - *Adjazenzmatrix*
    - Liste der Vorgänger bzw. Nachfolger je Knoten
  - *binäre Relationen* als gerichtete Graphen
  - *ungerichtete Graphen* als Sonderfall
  - Markierungsverfahren: *Tiefensuche* und *Breitensuche*; *Gerüst*
    - Labyrinth-Algorithmus (als Anekdote)
- Ende WS 1997/98

19. Mai 1998, Lagally