Gliederung des Kapitels 1.1

1.1 Algorithmen und Sprachen

~~1.1.1 Darstellung von Algorithmen~~
~~1.1.2 Grundlegende Datenbereiche~~
1.1.3 Realisierte Abbildung
1.1.4 (Künstliche) Sprachen
1.1.5 Grammatiken
1.1.6 BNF, Syntaxdiagramme
1.1.7 Sprachen zur Beschreibung von Sprachen
1.1.8 Übungsaufgaben

Abschnitt 1.1.3 soll Ihnen folgende Inhalte vermitteln:

Zu jedem Programm π gehört eine Abbildung $f_\pi: E \to A$. Hierbei ist E die Menge der zulässigen Eingabedaten und A die Menge der Ausgabedaten des Programms. f_π heißt die von π realisierte Abbildung.

Zur Menge aller korrekten Programme gehört die Menge der von diesen Programmen realisierten Abbildungen $\wp$. Diese Menge beschreibt alles, was unsere Programme leisten können. In diesem Abschnitt lernen Sie einige Eigenschaften dieser Menge.

1.1.3 Realisierte Abbildungen

Programme verändern Daten, d.h., sie bilden Eingabedaten auf Ausgabedaten ab. Genauer: Sei π ein korrekt aufgebautes Programm und seien E und A die Mengen seiner Eingabedaten bzw. der Ausgabedaten. Dann ordnet π jeder Eingabe e die zugehörige Ausgabe a zu, sofern π bei Eingabe von e nach endlich vielen Schritten anhält; anderenfalls wird e die Ausgabe "undefiniert" zugeordnet.

Definition 1.1.3.1: Die von einem Programm mit den Eingabe- und Ausgabemengen E und A realisierte (partielle) Abbildung $f_\pi: E \to A$ ist definiert durch

$$f_\pi(e) = \begin{cases} a, & \text{sofern } \pi \text{ bei Eingabe von e anhält und} \\ & \text{hierbei die Ausgabe a liefert,} \\ \perp & (=\text{"undefiniert"}) \text{ sonst.} \end{cases}$$

Beispiel (= Beispiel 1.1.2.2):

```
program fakultaet1 is
var  ergebnis, i, n: natural;
begin   read (n);  ergebnis := 1;
        for i := 1 to n do ergebnis := ergebnis*i od;
        write (ergebnis)
end
```

Die Eingabemenge E ist hier **N0** (n ist vom Typ natural), die Ausgabemenge A ist ebenfalls **N0**. Das Programm "fakultaet1" terminiert für jede Eingabe und berechnet die Fakultät der Eingabe. Also lautet die realisierte Abbildung:

$$f(n) = n! = \begin{cases} 1 \cdot 2 \cdot \ldots \cdot n, & \text{für } n > 0, \\ 1, & \text{für } n = 0. \end{cases}$$

Beispiel :

```
program q is
var  a1, a2, a3: character;
begin   read (a1); read (a2); read (a3);
        if a1 = a2 then a2 := a3 fi;
        a3 := a1; write (a1); write (a2); write (a3)
end
```

Die Eingabemenge E ist hier $\hat{A}^3$ (die Variablen a1, a2 und a3 sind vom Typ character), die Ausgabemenge A ist ebenfalls $\hat{A}^3$. Das Programm "q" terminiert für jede Eingabe und gibt stets drei Zeichen aus, von denen das erste und dritte übereinstimmen. Die realisierte Abbildung lautet also:

$$f_q(x,y,z) = \begin{cases} (x,z,x), & \text{falls } x = y, \\ (x,y,x), & \text{falls } x \neq y. \end{cases}$$

Definition 1.1.3.2:

$\wp_{E,A} = \{f: E \to A \mid \text{Es gibt ein Programm } \pi \text{ mit } f_\pi = f\}$

heißt die Menge der von Programmen realisierbaren oder berechenbaren Abbildungen von E nach A.

Man beachte, dass man die Mengen E und A nicht beliebig wählen kann. Denn es dürfen nur solche Mengen benutzt werden, die als Eingabe- oder Ausgabemengen von Programmen zulässig sind. Dies sind Mengen, die aus Folgen von Elementen aus den elementaren Wertebereichen bestehen, wobei die reellen Zahlen durch die Menge der dezimal dargestellten Zahlen mit endlich vielen Nachkommastellen ersetzt werden müssen. Wir präzisieren dies nun.

1.1.3.3: Menge aller elementarer Daten in Programmen:

D = {false, true} ∪ **Â** ∪ **Z** ∪ **Ř** mit

Ř = { z | Es gibt natürliche Zahlen k≥1 und n≥1 und eine Basis b ≥ 2, so dass $z = z_{n-1}z_{n-2} \ldots z_1z_0 \cdot z_{-1}z_{-2} \ldots z_{-k}$ oder $z = -z_{n-1}z_{n-2} \ldots z_1z_0 \cdot z_{-1}z_{-2} \ldots z_{-k}$ gilt und jedes z_i eine der Ziffern von 0 bis b-1 ist}.

Alle Elemente in **D** sind genau voneinander zu unterscheiden. Ist z.B. die ganze Zahl "zwei" gemeint, so wird sie als 2 (oder binär als 10) notiert, ist die reelle Zahl "zwei" gemeint, so stellen wir sie durch 2.0 dar. Ist dagegen die Ziffer "zwei" (also als Zeichen) gemeint, so schreiben wir '2'. (Grundsätzlich schließen wir einzelne Zeichen, um sie von Bezeichnern unterscheiden zu können, zwischen zwei Apostrophen ein.)

Einschub: Was Sie in der Mathematik lernten:

Eine Abbildung f: M → N ordnet jedem a ∈ M höchstens ein Element f(a) ∈ N zu. M heißt Vorbereich oder Urbildbereich (engl.: domain), N heißt Nachbereich oder Bildbereich (eng.: codomain).

Wird hierbei jedem a ∈ M genau ein Element f(a) ∈ N zugeordnet, so heißt die Abbildung total.

Falls es möglich sein kann (aber nicht muss), dass manchen kein Element a ∈ M durch die Abbildung zugeordnet wird, so spricht man von einer partiellen Abbildung. Insbesondere ist jede totale Abbildung auch eine partielle Abbildung.

noch Einschub

Statt "Abbildung" sagt man oft "Funktion", auch wenn dieser Begriff in der Mathematik auf Abbildungen beschränkt ist, deren Vor- und Nachbereiche Zahlenmengen sind.

Die Menge der Elemente
Def(f) = {a ∈ M | es gibt ein b ∈ N mit f(a) = b} ⊂ M
heißt Definitionsbereich von f.
f: M → N ist genau dann total, wenn Def(f) = M gilt.

Die Menge der Elemente
f(M) = {b ∈ N | es gibt ein a ∈ M mit f(a) = b} ⊂ N
heißt Bild von f oder Bild von M unter f.

Die Menge der Abbildungen von M nach N bezeichnet man mit N^M oder mit Abb(M,N) = {f | f: M → N} = N^M.

(Achten Sie bei Büchern genau darauf, ob jeweils partielle oder totale Abbildungen gemeint sind!)

noch Einschub

Eine Abbildung f: M → N heißt injektiv genau dann, wenn für alle a, b ∈ M mit a ≠ b gilt: f(a) ≠ f(b).

Eine Abbildung f: M → N heißt surjektiv genau dann, wenn es zu jedem c ∈ N mindestens ein a ∈ M mit f(a) = c gibt. f ist genau dann surjektiv, wenn f(M) = N gilt.

Eine Abbildung f: M → N heißt bijektiv genau dann, wenn sie injektiv und surjektiv ist.

Wenn es eine bijektive Abbildung f: M → N gibt, dann gibt es zu jedem c ∈ N genau ein a ∈ M mit f(a) = c. Setze daher $f^{-1}(c) = a$. Dies definiert eine bijektive Abbildung f^{-1}: N → M. Diese Abbildung f^{-1} heißt die Umkehrabbildung oder die Inverse zu f. Bijektive Abbildungen besitzen also stets eine Inverse, die ebenfalls bijektiv ist.

noch Einschub

Wir führen folgende Begriffe für Mengen M ein: "endlich", "unendlich", "abzählbar", "überabzählbar", "aufzählbar", "beschränkt". Mit #M oder |M| bezeichnen wir die Anzahl der Elemente der Menge M.

M ist endlich ⇔ M ist leer oder es gibt eine natürliche Zahl n und eine Bijektion f: M → {1, ..., n}.
(In diesem Falle ist #M = n.)

M ist unendlich ⇔ M ist nicht endlich.

M abzählbar (unendlich) ⇔ Es gibt eine Bijektion von M zu den natürlichen Zahlen.

M ist höchstens abzählbar ⇔
M ist endlich oder M ist abzählbar unendlich.

noch Einschub

M ist überabzählbar ⇔ M ist nicht "höchstens abzählbar".

M ist aufzählbar ⇔ M ist leer oder es gibt einen Algorithmus, der jeder natürlichen Zahl n ein Element der Menge M zuordnet, und zu jedem Element von M gibt es mindestens eine hierdurch zugeordnete natürliche Zahl.

Der Begriff "beschränkt" ist ein relativer Begriff; er bezieht sich auf die jeweils betrachtete Umgebung. Hierzu muss man *vorher* eine Schranke k festlegen. *Bei abzählbaren Zahlenmengen* lautet eine solche Definition dann beispielsweise:
M ist beschränkt ⇔ Es gibt eine vorab festgelegte natürliche Zahl k, so dass M nicht mehr als k Elemente besitzt.

Eine beschränkte Teilmenge von **Z** ist endlich, aber aus der Eigenschaft "endlich" folgt nicht "beschränkt", da man im Allgemeinen die Schranke k nicht vorab festlegen kann.

noch Einschub: *Beispiele*

Die leere Menge ∅ ist endlich. Sie besitzt #∅ = 0 Elemente.

Für jede natürliche Zahl n ist die Menge {1, 2, ..., n} endlich. Die Menge der natürlichen Zahlen **N** = {1, 2, 3, ...} bzw. **N0** = {0, 1, 2, 3, ...} und die Menge der ganzen Zahlen **Z** = { ..., -2, -1, 0, 1, 2, 3, ... } sind abzählbar unendlich und selbstverständlich aufzählbar.

Die Menge der rationalen Zahlen **Q** = {n/m | n ganze Zahl, m positive ganze Zahl, n und m teilerfremd} ist abzählbar unendlich und zugleich aufzählbar (wie zeigt man dies?).

Die Menge der reellen Zahlen **R** und die Menge der komplexen Zahlen **C** sind überabzählbar (Beweis über das Cantorsche Diagonalverfahren, vgl. Mathematikvorlesungen).

Die Menge

D = {false, true} ∪ **Â** ∪ **Z** ∪ **Ř** mit
Ř = { z | Es gibt natürliche Zahlen k≥1 und n≥1 und eine Basis b ≥ 2, so dass $z = z_{n-1}z_{n-2} \dots z_1z_0 \cdot z_{-1}z_{-2} \dots z_{-k}$ oder $z = -z_{n-1}z_{n-2} \dots z_1z_0 \cdot z_{-1}z_{-2} \dots z_{-k}$ gilt und jedes z_i eine der Ziffern von 0 bis b-1 ist}.

ist eine abzählbar unendliche Menge.

(**Warum?** Können Sie eine Abzählung, also eine bijektive Abbildung zu den natürlichen Zahlen angeben? Beachte: für jedes z ist die Länge der Darstellung ist endlich, aber nicht beschränkt.)

Die Eingabe in ein Programm ist nun nichts anderes als eine Folge von Elementen aus **D**. Wir definieren die "Menge der Folgen".

Definition 1.1.3.4: Es sei M eine Menge. Die Menge aller endlichen Folgen von Elementen aus M

$M^* = \{a_1 a_2 \dots a_n \mid n \geq 0 \text{ und } a_i \in M \text{ für } i = 1, 2, \dots, n\}$

heißt Wortmenge oder Menge der Wörter oder freies Monoid über M. Die Elemente von M* heißen Wörter oder Folgen.

Für ein Wort $u = u_1 u_2 \dots u_n$ heißt $n = |u|$ die Länge des Wortes. Das Wort der Länge 0 heißt leeres Wort und wird mit ε bezeichnet.

Auf M* ist die Operation "Konkatenation" definiert:
Für zwei Wörter $u = u_1 u_2 \dots u_n$ und $v = v_1 v_2 \dots v_m$ ist
$u\,v = u_1 u_2 \dots u_n v_1 v_2 \dots v_m$.
Diese Operation ist assoziativ und sie besitzt ε als Einheit, d.h., u ε = ε u = u für alle u ∈ M*. Offenbar gilt $|u\,v| = |u| + |v|$.
Speziell sei u^k (*u hoch k*) die k-fache Konkatenation eines Wortes u mit sich, d.h.: $u^0 = \varepsilon$ und $u^{k+1} = u\,u^k$ für alle k ≥ 0.

D*, die Menge der Wörter über **D**, ist ebenfalls eine abzählbare Menge (warum?).

In ein Programm werden Elemente vom Datentyp Boolean, natural, integer, real oder character eingegeben, also Elemente aus **D**. Ebenso werden solche Elemente aus **D** vom Programm ausgegeben. *Jedes* Programm π realisiert somit eine partielle Abbildung f_π: **D*** → **D***.

Kein Programm π schöpft hierbei ganz **D*** aus, vielmehr sind der Definitionsbereich von f_π und das Bild unter f_π in der Regel eine der "üblichen Mengen", also etwa f_π: **Â*** → **Â*** oder f_π: $\mathbf{Z}^3 \to \mathbf{Z}^2$ usw.

Allgemein lässt sich dann die Menge aller von Programmen realisierbarer Abbildungen wie folgt definieren:

Definition 1.1.3.5:

℘ = {f | Es gibt ein Programm π mit f = f_π: **D*** → **D***}

ist die Menge der von Programmen realisierbaren Abbildungen. Man nennt diese Menge auch die Menge der von Programmen berechenbaren Funktionen oder die Menge der partiell rekursiven Funktionen (den Begriff "rekursive Funktion" verwendet man vor allem dann, wenn die Vor- und Nachbereiche Zahlenmengen sind).

Die Mengen aus Definition 1.1.3.2
$\wp_{E,A} = \{f: E \to A \mid \text{Es gibt ein Programm } \pi \text{ mit } f_\pi = f\}$
lassen sich als Teilmengen von ℘ auffassen, da alle Eingabe- und Ausgabemengen E und A Teilmengen von **D*** sind.

Die Menge ℘ gilt als eine der "großen Entdeckungen" des 20. Jahrhunderts. Sie bildet die Obermenge dessen, was man mit Programmen beschreiben und realisieren, bzw. was man mit Algorithmen erreichen kann. Sie hat interessante Eigenschaften.

Insbesondere ist die Menge ℘ abgeschlossen gegen gewisse Operationen. So gilt beispielsweise: Wenn f, g ∈ ℘ sind, dann ist auch die Hintereinanderausführung f∘g ∈ ℘ [f∘g(x) := f(g(x)) für alle x]. Wenn f ∈ ℘ ist, dann ist auch dessen "Iterierte" $f^i = \underbrace{f \circ f \circ f \circ \dots \circ f}_{\text{i-mal}} \in \wp$.

Wenn f, g ∈ ℘ und b ein Boolescher Ausdruck sind, dann ist auch cond(b,f,g) ∈ ℘ mit cond(b,f,g)(x) = if b(x) then f(x) else f(x) fi für alle Argumente x. (*usw. usw., siehe künftige Vorlesungen*)

Die Menge **D*** gilt für alle Programme. Jedes Programm π hat jedoch genau eine (durch die Datentypen der Variablen in den Eingabeanweisungen) definierte Menge E_π seiner Eingabedaten und ebenso genau eine (durch den Datentyp der Variablen oder Ausdrücke in den Ausgabeanweisungen) definierte Menge von Ausgabedaten A_π; beide Mengen E_π und A_π sind Teilmengen von **D*** und nur mittels Vereinigungen und Hintereinanderschreiben aus den Mengen **B, N0, Z, Ř** und **Â** aufgebaut.

Folglich gibt es zu jedem Programm π eine Eingabemenge E_π , die genau die Menge aller Folgen von Eingabedaten definiert, die von dem Programm π korrekt vollständig eingelesen werden. Wir sagen, das Programm π terminiert immer, wenn π für alle Eingabedaten aus E_π nach endlich vielen Schritten anhält (ggf. mit Fehlerabbruch).

Beispiel: Folgendes Programm ist immer terminierend:

```
program h is
var  n, i: integer; a1, a2: character;
begin  read (a1);
       i := 1;
       if a1 = 'J'  then read (n); a2 := 'A'
                    else read (a2); n := 2 fi;
       while n > 0 do  i := i+i; n := n-1 od;
       write (a1); write (a2); write (i)
end
```

Die Eingabemenge $E_h \subset$ **D*** dieses Programms h lautet
E_h = **Â Z** ∪ **Â Â**,
d.h., korrekte Eingaben bestehen aus einem Zeichen gefolgt von einer ganzen Zahl oder aus zwei aufeinanderfolgenden Zeichen.

Analog lautet die Ausgabemenge A_h dieses Programms
A_h = **Â Â Z**.

Beispiel, Fortsetzung:

Die Eingabemenge E_h ist zu unterscheiden von der Menge derjenigen Eingaben, für die das Programm ohne Abbruch arbeitet. Gibt man in obiges Programm beispielsweise 'J' 'J' ein, so bricht das Programm mit einem Fehler ab, da es nach Einlesen des Zeichens 'J' versucht, eine Zahl einzulesen, aber in der Eingabe statt dessen ein Zeichen vorfindet. Die Menge, für die h ohne Fehlerabbruch arbeitet, lautet:

$K_h = \{xy \mid x = 'J' \in$ **Â** und $y \in$ **Z**$\} \cup \{xy \mid x,y \in$ **Â** und $x \neq 'J'\}$
$= \{'J'\}$ **Z** $\cup$ (**Â**$\setminus\{'J'\}$) **Â**.

Diese Menge der fehlerfrei bearbeiteten Eingaben K_h lässt sich im Allgemeinen nicht exakt bestimmen (siehe später; diese Menge ist generell "nicht entscheidbar").

Geben Sie die realisierte Funktion f_h an! ■

Definition 1.1.3.6:

$\Re = \{f \mid$ Es gibt ein Programm π, das immer terminiert, mit $f = f_\pi\}$

ist die Menge der von stets anhaltenden Programmen realisierbaren Abbildungen. Man nennt diese Menge auch die Menge der von Programmen total berechenbaren Funktionen oder die Menge der total rekursiven Funktionen.

Die in ℜ liegenden Abbildungen sind total in dem Sinne, dass sie für jede Eingabe zu einem Ende kommen entweder durch einen Fehlerabbruch, weil die Datentypen bei der Eingabe nicht stimmen oder ein Fehler bei der Bearbeitung geschieht (z.B. Division durch 0), oder durch Erreichen des Endes des Programms.