Kapitel 2

Strukturelle Rekursion und Induktion

Rekursion ist eine konstruktive Technik für die Beschreibung unendlicher Mengen (und damit insbesondere für die Beschreibung unendliche Funktionen). Induktion ist eine Technik für das Beweisen von Eigenschaften von rekursiv definierten Mengen. Es gibt verschiedene Ausprägungen von Rekursion und Induktion. In diesem Kapitel interessieren wir uns besonders für die einfachsten Varianten von Rekursion und Induktion, die als strukturell bezeichnet werden.

Strukturelle Rekursion ist Ihnen bereits als grundlegende Programmiertechnik vertraut. Als Beispiel betrachten wir zwei Deklarationen in Standard ML:

```
datatype tree = L of int | N of tree * int * tree

fun size (L _)        = 1
  | size (N(t,_,t')) = 1 + size t + size t'

val size : tree -> int
```

Die Werte des Typs `tree` kann man als binäre Bäume auffassen, deren Knoten mit ganzen Zahlen markiert sind. Die Prozedur `size` liefert die Anzahl der Knoten eines binären Baums. Beide Deklarationen erfolgen mithilfe von struktureller Rekursion.

2.1 Rekursive Definition von Mengen

Damit wir nicht zu sehr ins Abstrakte abdriften, diskutieren wir strukturelle Rekursion an dem einfachst möglichen Beispiel: Wir definieren eine Menge N, die isomorph zur Menge $\mathbb{N}$ der natürlichen Zahlen ist.[1] Dabei stellen wir alle natürlichen Zahlen durch reine Mengen dar. Aufbauend auf N werden wir später mit

[1] Die hier definierte Menge N wird in der Literatur oft als ω bezeichnet.

struktureller Rekursion Funktionen definieren, die der Addition und Multiplikation von Zahlen entsprechen.

Wir stellen die natürlichen Zahlen wie folgt durch reine Mengen dar:

0	$\emptyset$
1	$\{\emptyset\}$
2	$\{\{\emptyset\}\}$
...	

Für N wählen wir die Menge, die genau aus den reinen Mengen besteht, die gemäß dieser Vereinbarung natürliche Zahlen darstellen.

Wir zeigen nun, wie wir die Menge N durch strukturelle Rekursion beschreiben können, ohne dabei auf die natürlichen Zahlen $\mathbb{N}$ zurückzugreifen. Wir tun dies, indem wir eine rekursive Konstruktionsvorschrift für die Elemente von N angeben:

1. Die leere Menge ist ein Element von N.
2. Wenn x ein Element von N ist, dann ist die Menge $\{x\}$ ein Element von N.
3. N enthält nur Elemente, die durch diese Regeln konstruiert werden können.

Diese Konstruktionsvorschrift kann man übersichtlich mit zwei *Inferenzregeln* formulieren:

$$\frac{}{\emptyset \in N} \qquad \frac{x \in N}{\{x\} \in N}$$

Mit der ersten Regel kann man die Darstellung der Zahl 0 konstruieren. Mit der zweiten Regel kann man aus der Darstellung einer Zahl n die Darstellung der Nachfolgezahl $n + 1$ konstruieren. Für die Darstellung der Zahl 3 benötigen wir 4 Konstruktionsschritte:

1.	$\emptyset$	mit Regel 1
2.	$\{\emptyset\}$	mit Regel 2
3.	$\{\{\emptyset\}\}$	mit Regel 2
4.	$\{\{\{\emptyset\}\}\}$	mit Regel 2

Die rekursive Definition von N ist so zu verstehen, dass N genau die Objekte enthält, die mithilfe der Inferenzregeln in *endlich vielen* Schritten konstruierbar sind.

Man kann Konstruktionsvorschrift für die Elemente von N auch mit einer rekursiven Gleichung beschreiben:

$$N = \{\emptyset\} \cup \{\, \{x\} \mid x \in N \,\}$$

Die rekursive Definition von N hat zwei wichtige Eigenschaften, die sie als *strukturell rekursiv* auszeichnen:

1. Jedes Objekt in N kann nur mit einer der definierenden Regeln konstruiert werden. Zum Beispiel kann $\emptyset$ nur mit der ersten Regel und $\{\emptyset\}$ nur mit der zweiten Regel konstruiert werden.

2. Die rekursive Regel konstruiert aus einem Objekt $x \in N$ ein größeres Objekt $\{x\} \in N$, das x als echtes Teilobjekt enthält.

Rekursive Konstruktortypdeklarationen in Standard ML können als strukturell rekursive Definitionen von Mengen verstanden werden. Beispielsweise entspricht die Deklaration

```
datatype nat = N | S of nat
```

der rekursiven Gleichung

$$N = \{\emptyset\} \cup \{\, \{x\} \mid x \in N \,\}$$

2.2 Rekursive Definition von Funktionen

Wir wollen jetzt eine Funktion $D \in N \to \mathbb{N}$ definieren, die die Elemente von N auf die durch sie dargestellten Zahlen abbildet. Das gelingt wie folgt:

$$\begin{aligned} &D \in N \to \mathbb{N} \\ &D(\emptyset) = 0 \\ &D(\{x\}) = 1 + D(x) \end{aligned}$$

Diese Definition hat zwei wichtige Eigenschaften, die sie als strukturell rekursiv auszeichnen:

1. Für jede der definierenden Regeln von N gibt es eine entsprechende definierende Regel für D. Das hat zur Folge, dass auf jedes Element von N genau eine der Regeln für D anwendbar ist.

2. Rekursive Anwendungen von D erfolgen nur auf echte Teilobjekte des Arguments. (In der zweiten Regel für D erfolgt die rekursive Anwendung für ein Argument $\{x\}$ auf das echte Teilobjekt x.)

Zusammen garantieren diese Eigenschaften, dass die Regeln eine totale Funktion definieren. Wenn wir die obige Definition als Prozedurdeklaration auffassen, bekommen wir eine Prozedur, die für alle Eingabewerte regulär terminiert.

Als Nächstes definieren wir mithilfe von struktureller Rekursion eine zweistellige Funktion $+ \in N \times N \to N$, die der Addition von natürlichen Zahlen entspricht:

$$\begin{aligned} + &\in N \times N \to N \\ \emptyset + y &= y \\ \{x\} + y &= \{x + y\} \end{aligned}$$

Machen Sie sich klar, dass die zweite definierende Regel für D und $+$ jeweils eine Konstruktionsvorschrift festlegt, mit der Funktionswerte für größere Argumente aus den Funktionswerten für kleinere Argumente erhalten werden. Im Falle der zweistelligen Funktion $+$ läuft die Rekursion über das erste Argument.

2.3 Strukturelle Induktion

Um zu beweisen, dass alle $x \in N$ eine gegebene Eigenschaft A erfüllen, können wir wie folgt vorgehen:

1. Beweise, dass $\emptyset$ die Eigenschaft A erfüllt.
2. Beweise, dass für jedes $x \in N$, dass A erfüllt, $\{x\}$ die Eigenschaft A erfüllt.

Die gerade formulierte Beweisregel bezeichnen wir als *Induktionsregel für N*. Sie kann übersichtlich als Inferenzregel dargestellt werden:

$$\frac{A(\emptyset) \qquad \forall x \in N\colon A(x) \Rightarrow A(\{x\})}{\forall x \in N\colon A(x)}$$

Überzeugen Sie sich davon, das die *Prämissen*

(1) $A(\emptyset)$
(2) $\forall x \in N\colon A(x) \Rightarrow A(\{x\})$

der Induktionsregel direkt aus den definierenden Regeln für N abgeleitet sind.

Machen Sie sich die Gültigkeit der gerade formulierten Induktionsregel für N klar. Falls Ihnen das nicht auf Anhieb gelingt, hilft Ihnen vielleicht die folgende Argumentation. Sei A eine Aussage, für die die oben gezeigten Prämissen (1) und (2) der Beweisregel erfüllt sind. Wir überlegen uns, warum aus den Prämissen die Gültigkeit der Aussage

$$A(\{\{\{\emptyset\}\}\})$$

folgt. Wegen Prämisse (1) gilt:

$$A(\emptyset)$$

Daraus folgt mit Prämisse (2):

$$A(\{\emptyset\})$$

Durch zwei weitere Anwendungen von Prämisse (2) erhalten wir wie gewünscht:

$$A(\{\{\{\emptyset\}\}\})$$

Der Trick liegt also bei Prämisse (2), die für beliebiges $x \in N$ sagt, dass A für $\{x\}$ gültig ist, wenn A für x gültig ist.

Als Nächstes demonstrieren wir die Induktionsregel für N an einem konkreten Beispiel. Wir wollen zeigen, dass

$$\forall x \in N\ \forall y \in N:\ \ D(x+y) = D(x) + D(y)$$

gilt. Dazu verwenden wir die Aussage

$$A(x) \ \stackrel{\text{def}}{=}\ \ \forall y \in N:\ \ D(x+y) = D(x) + D(y)$$

Die Induktionsregel sagen uns, dasss es genügt, die folgenden zwei Eigenschaften zu zeigen (es handelt sich dabei um die mit dem konkreten A instantiierten Prämissen der Induktionsregel):

(1) $\forall y \in N:\ D(\emptyset + y) = D(\emptyset) + D(y)$

(2) $\forall x \in N:\ (\forall y \in N:\ D(x+y) = D(x) + D(y))$
$\quad\Rightarrow (\forall y \in N:\ D(\{x\} + y) = D(\{x\}) + D(y))$

Es ist nicht schwer diese beiden Eigenschaften zu beweisen.

Damit der Schreibaufwand klein und die Übersicht gewahrt bleibt, schreibt man Induktionsbeweise nach einem bestimmten Schema auf, das wir mit dem folgen Beweis demonstrieren.

Proposition 2.3.1 $\forall x \in N\ \forall y \in N:\ \ D(x+y) = D(x) + D(y)$.

Beweis Durch strukturelle Induktion über $x \in N$.

Sei $x = \emptyset$. Dann:

$$\begin{aligned} D(x+y) &= D(\emptyset + y) & \\ &= D(y) & \text{Definition von } + \\ &= 0 + D(y) & \\ &= D(\emptyset) + D(y) & \text{Definition von } D \\ &= D(x) + D(y) & \end{aligned}$$

Sei $x = \{x'\}$. Dann:

$$\begin{aligned}
D(x+y) &= D(\{x'\}+y) \\
&= D(\{x'+y\}) && \text{Definition von } + \\
&= 1 + D(x'+y) && \text{Definition von } D \\
&= 1 + D(x') + D(y) && \text{Induktionsannahme} \\
&= D(\{x'\}) + D(y) && \text{Definition von } D \\
&= D(x) + D(y)
\end{aligned}$$

□

Für jede durch strukturelle Rekursion definierte Menge gibt es eine aus der Definition der Menge ableitbare Induktionsregel. Beispielsweise bekommen wir für die Definition

$$M \;=\; \{\langle\rangle\} \;\cup\; (M \times M) \;\cup\; (M \times M \times M)$$

eine Induktionsregel mit drei Prämissen:

$$\frac{\begin{array}{c} A(\langle\rangle) \\ \forall x, y \in M\colon\; A(x) \wedge A(y) \Rightarrow A(\langle x, y\rangle) \\ \forall x, y, z \in M\colon\; A(x) \wedge A(y) \wedge A(z) \Rightarrow A(\langle x, y, z\rangle) \end{array}}{\forall x \in M\colon\; A(x)}$$

2.4 Wohlfundierte Induktion

Rekursion und Induktion treten in vielen Varianten auf. Die gemeinsame Essenz dieser Varianten lässt sich mit dem Begriff der wohlfundierten Relation formulieren.

Sei X eine Menge. Eine binäre Relation $\succ \subseteq X \times X$ heißt *wohlfundiert*, wenn es keine unendliche Folge $x_1, x_2, x_3, \ldots$ von Elementen von X gibt mit $x_1 \succ x_2 \succ x_3 \succ \cdots$. Ein Paar $(X, \succ)$ aus einer Menge X und einer wohlfundierten Relation $\succ$ auf X bezeichnet man als *wohlfundierte Menge*.

Sei $(X, \succ)$ eine wohlfundierte Menge. Mit der Notation $x \succ y$ verbinden wir die bildhafte Vorstellung, dass x in einem gewissen Sinne größer ist als y. Wohlfundiertheit bedeutet dann, dass es keine unendlichen absteigenden Ketten $x_1 \succ x_2 \succ x_3 \succ \cdots$ gibt. Das Konzept von Wohlfundiertheit ist uns bereits im Zusammenhang mit dem strukturellen Aufbau von mathematischen Objekten begegnet.

Als Beispiel betrachten wir die Menge N. Offensichtlich ist

$$x \succ y \overset{\text{def}}{\Longleftrightarrow} x = \{y\} \ \wedge \ x \in N$$

eine wohlfundierte Relation auf N. Die reflexive und transitive Hülle von $\succ$ entspricht übrigens der kanonischen Ordnung der natürliche Zahlen.

Sei $(X, \succ)$ eine wohlfundierte Menge und $M \subseteq X$. Ein $x \in M$ heißt *minimales Element von M*, wenn es kein $y \in M$ gibt mit $x \succ y$.

Proposition 2.4.1 *Sei $(X, \succ)$ eine wohlfundierte Menge. Dann hat jede nichtleere Teilmenge von M mindestens ein minimales Element.*

Beweis Durch Widerspruch. Sei M eine nichtleere Teilmenge von X, die kein minimales Element hat. Dann existiert zu jedem $x \in M$ ein $y \in M$ sodass $x \succ y$. Da M nach Annahme mindestens ein Element hat, können wir unendliche absteigende Kette konstruieren. Widerspruch. □

Sei $(X, \succ)$ eine wohlfundierte Menge und $A(x)$ eine Aussage für $x \in X$. Dann wird die Inferenzregel

$$\frac{\forall x \in X\colon\ (\forall y \in X\colon x \succ y \Rightarrow A(y)) \ \Rightarrow\ A(x)}{\forall x \in X\colon\ A(x)}$$

als *wohlfundierte Induktion* für X, $\succ$ und A bezeichnet. Die folgende Proposition formuliert die Korrektheit dieser Inferenzregel.

Proposition 2.4.2 (Wohlfundierte Induktion) *Sei $(X, \succ)$ eine wohlfundierte Menge und $A(x)$ eine Aussage für $x \in X$. Außerdem gelte:*

$$\forall x \in X\colon\ (\forall y \in X\colon x \succ y \Rightarrow A(y)) \ \Rightarrow\ A(x)$$

Dann gilt: $\forall x \in X\colon\ A(x)$.

Beweis Durch Widerspruch. Sei M die Teilmenge von X, die alle Elemente enthält, für die A nicht gilt. Wir nehmen an, dass M nichtleer ist. Dann können wir ein minimales Element x_0 von M wählen (Proposition 2.4.1). Da M alle Elemente von X enthält, für die A nicht gilt, folgt

$$\forall y \in X\colon x_0 \succ y \Rightarrow A(y)$$

Also folgt $A(x_0)$ mit der letzten Annahme der Proposition. Widerspruch. □

Übungen

Aufgabe 2.1 (Strukturelle Induktion für N) Seien die Menge N und die Funktion $+$ wie besprochen definiert:

$$N \overset{\text{def}}{=} \{\emptyset\} \cup \{\, \{x\} \mid x \in N \,\}$$

$$\begin{aligned} &+ \in N \times N \to N \\ &\quad \emptyset + y = y \\ &\{x'\} + y = \{x' + y\} \end{aligned}$$

1. Definieren Sie eine Multiplikationsfunktion $\cdot \in N \times N \to N$.
2. Beweisen Sie: $\forall x, y, z \in N\colon \quad (x + y) + z = x + (y + z)$.
3. Beweisen Sie: $\forall x \in N\colon \quad x + \emptyset = x$.
4. Beweisen Sie: $\forall x, y \in N\colon \quad x + y = y + x$.

Aufgabe 2.2 (Strukturelle Induktion für Listen) Sei X eine Menge. Wir definieren die Menge $\mathcal{L}(X)$ der Listen über X durch die folgende rekursive Gleichung:

$$\mathcal{L}(X) \overset{\text{def}}{=} \{\langle\rangle\} \cup (X \times \mathcal{L}(X))$$

Wir führen zwei an Standard ML angelehnte Notationen ein. Für die *leere Liste* $\langle\rangle$ schreiben wir *nil*, und für eine *nichtleere Liste* $\langle x, xr\rangle$ schreiben wir $x :: xr$ (lies x cons xr). Gegeben eine nichtleere Liste $x :: xr$, bezeichnet wir x als den *Kopf* und xr als den *Rumpf* der Liste.

1. Definieren Sie die Menge $\mathcal{L}(X)$ mithilfe von Inferenzregeln.
2. Geben Sie die Induktionsregel für $\mathcal{L}(X)$ an.

Für den zweiten Teil der Aufgabe definieren wir drei Funktionen durch strukturelle Rekursion:

$$|\_| \in \mathcal{L}(X) \to \mathbb{N} \qquad \text{Länge}$$
$$|nil| = 0$$
$$|x :: xr| = 1 + |xr|$$

$$@ \in \mathcal{L}(X) \times \mathcal{L}(X) \to \mathcal{L}(X) \qquad \text{Konkatenation}$$
$$nil @ ys = ys$$
$$(x :: xr) @ ys = x :: (xr @ ys)$$

$$rev \in \mathcal{L}(X) \to \mathcal{L}(X) \qquad \text{Reversion}$$
$$rev(nil) = nil$$
$$rev(x :: xr) = rev(xr) @ [x]$$

Beweisen Sie die folgenden Aussagen:

1. $\forall xs, ys, zs \in \mathcal{L}(X): \quad (xs @ ys) @ zs = xs @ (ys @ zs)$
2. $\forall xs \in \mathcal{L}(X): \quad xs @ nil = xs$
3. $\forall xs, ys \in \mathcal{L}(X): \quad |xs @ ys| = |xs| + |ys|$
4. $\forall xs \in \mathcal{L}(X): \quad |rev(xs)| = |xs|$
5. $\forall xs, ys \in \mathcal{L}(X): \quad rev(xs @ ys) = rev(ys) @ rev(xs)$
6. $\forall xs \in \mathcal{L}(X): \quad rev(rev(xs)) = xs$

Aufgabe 2.3 (Rekursion und Induktion für Bäume) Sei die Menge T wie folgt rekursiv definiert:

$$T \stackrel{\text{def}}{=} \mathbb{Z} \cup (T \times T)$$

1. Beschreiben Sie die Menge T mit zwei Inferenzregeln.
2. Geben Sie die Induktionsregel für T an.
3. Wir stellen uns die Elemente von T als binäre Bäume vor, deren Blätter mit ganzen Zahlen markiert sind. Definieren Sie eine Funktion

   $$s \in T \to \mathbb{N}$$

   die zu einem Baum die Anzahl seiner Knoten liefert.
4. Deklarieren Sie einen Typ `tree`, der der Menge T entspricht.
5. Deklarieren Sie eine Prozedur `size`, die der Funktion s entspricht.

