Kapitel 2

Strukturelle Rekursion und Induktion

Rekursion ist eine konstruktive Definitionstechnik für Mengen und Funktionen. Induktion ist eine Beweistechnik, mithilfe derer man Eigenschaften von rekursiv definierten Mengen und Funktionen beweisen kann. Es gibt verschiedene Ausprägungen von Rekursion und Induktion. In diesem Kapitel interessieren wir uns für Varianten von Rekursion und Induktion, die als strukturell bezeichnet werden.

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 Größe eines binären Baums (also die Anzahl seiner Knoten). 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 am konkreten Beispiel von Listen.

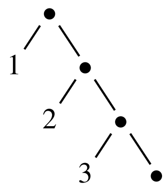
Sei X eine Menge. Wir definieren die Menge $\mathcal{L}(X)$ der *Listen über X* wie folgt:

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

Diese Definition ist rekursiv, da $\mathcal{L}(X)$ auch in der rechten Seite der Gleichung vorkommt. Sie besagt, dass eine Liste über X entweder das leere Tupel $\langle\rangle$ ist, oder ein Paar $\langle x, xs\rangle$, dass aus einem $x \in X$ und einer Liste xs über X besteht. Wir sagen, dass es zwei *Varianten* für Listen gibt (die leere Liste und nichtleere Listen). Hier ist ein Beispiel für eine Liste über $\mathbb{Z}$:

$$\langle 1,\ \langle 2,\ \langle 3,\ \langle\rangle\rangle\rangle\rangle$$

Grafisch lässt sich diese Liste wie folgt darstellen:



Man kann die Menge $\mathcal{L}(X)$ auch durch zwei Inferenzregeln definieren (eine Inferenzregel pro Variante):

$$\frac{}{\langle\rangle \in \mathcal{L}(X)} \qquad \frac{x \in X \qquad xs \in \mathcal{L}(X)}{\langle x, xs\rangle \in \mathcal{L}(X)}$$

Diese Regeln sollte man sich als Konstruktionsregeln vorstellen. Mit der ersten Regel kann die leere Liste konstruiert werden. Mit der zweiten Regel kann aus einem $x \in X$ und einer bereits konstruierten Liste xs über X die Liste $\langle x, xs\rangle$ konstruiert werden. Für die Liste $\langle 1,\ \langle 2,\ \langle 3,\ \langle\rangle\rangle\rangle\rangle$ benötigen wir 4 Konstruktionsschritte:

1.	$\langle\rangle$	mit Regel 1
2.	$\langle 3, \langle\rangle\rangle$	mit Regel 2
3.	$\langle 2, \langle 3, \langle\rangle\rangle\rangle$	mit Regel 2
4.	$\langle 1, \langle 2, \langle 3, \langle\rangle\rangle\rangle\rangle$	mit Regel 2

Die rekursive Definition von $\mathcal{L}(X)$ ist so zu verstehen, dass $\mathcal{L}(X)$ genau die Objekte enthält, die mithilfe der Inferenzregeln in *endlich vielen* Schritten konstruierbar sind.

Wir sagen, dass die Definition von $\mathcal{L}(X)$ mithilfe von *struktureller* Rekursion erfolgt, weil die rekursive Konstruktionsregel aus einer Liste xs eine „größere" Liste $\langle x, xs\rangle$ konstruiert, die xs als echtes Teilobjekt enthält.

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

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

die Definition der Menge

$$T = \mathbb{Z} \uplus (T \times \mathbb{Z} \times T)$$

Die Notation $\uplus$ (Summe von Mengen) wurde in Kapitel 1 definiert.

Für führen noch die aus Standard ML bekannte Notation für Listen ein. Für die *leere Liste* $\langle\rangle$ schreiben wir auch *nil*. Für eine *nichtleere Liste* $\langle x, r\rangle$ schreiben wir auch $x :: r$ (lies x cons r). Gegeben eine nichtleere Liste $x :: r$, bezeichnet wir x als den *Kopf* und r als den *Rumpf* der Liste.

2.2 Rekursive Definition von Funktionen

Wir wollen jetzt eine Funktion $\mathcal{L}(X) \to \mathbb{N}$ definieren, die Listen auf ihre Länge abbildet. Dies gelingt wie folgt:

$$\begin{aligned} |\_| &\in \mathcal{L}(X) \to \mathbb{N} \\ |nil| &= 0 \\ |x :: xr| &= 1 + |xr| \end{aligned}$$

Diese Definition ist rekursiv und folgt der Struktur der Definition der Menge $\mathcal{L}(X)$. Hier sind einige charakteristische Eigenschaften der Definition:

1. Für jede Variante von $\mathcal{L}(X)$ gibt es eine eigene Regel.
2. *Die Regeln sind erschöpfend.* Das bedeutet, dass auf jede Liste mindestens eine Regel anwendbar ist.
3. *Die Regeln schließen sich gegenseitig aus.* Das bedeutet, dass auf keine Liste mehr als eine Regel anwendbar ist.
4. *Die Rekursion ist strukturell.* Da bedeutet, dass rekursive Anwendungen nur auf echte Teilobjekte des Argumentes erfolgen (die zweite Regel enthält eine rekursive Anwendung der Funktion $|\_|$ auf das echte Teilobjekt xr des Argumentes $x :: xr$).

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.

Hier sind zwei weitere Beispiele für die Definition von Funktionen mithilfe von strukturelle Rekursion über $\mathcal{L}(X)$:

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

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

Gegeben eine Konstruktortypdeklaration, kann man die linken Seite der Regeln für eine strukturell rekursive Prozedurdeklaration über dem Konstruktortyp automatisch generieren. Als Beispiel betrachten wir die Deklaration

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

Bis auf die Wahl der Bezeichner `size`, `x`, `t` und `t'` kann die folgende Vorlage für die Deklaration eine Prozedur für Argumente des Typs `tree` automatisch abgeleitet werden:

```
fun size (L x)      = ...
  | size (N(t,x,t') = ...
```

Da nur die zweite Variante von `tree` rekursiv definiert ist, darf nur die zweite Regel für `size` rekursive Anwendungen enthalten. Die rekursiven Anwendungen dürfen zudem nur auf die Teilobjekte t und t' erfolgen.

2.3 Strukturelle Induktion

Sei X eine Menge. Für Listen über X gilt die folgende Induktionsregel:

$$\frac{A(nil) \qquad \forall x \in X\ \forall xr \in \mathcal{L}(X)\colon A(xr) \Rightarrow A(x :: xr)}{\forall xs \in \mathcal{L}(X)\colon A(xs)}$$

Die Prämissen der Induktionsregel entsprechen den Konstruktionsregeln für Listen. Die durch diese Regel formulierte Beweistechnik wird als *Listeninduktion* oder als *strukturelle Induktion* bezeichnet.

Wir wollen uns zuerst von der Korrektheit der Inferenzregel überzeigen. Dazu betrachten wir den Fall $X = \mathbb{Z}$ (Listen über $\mathbb{Z}$) und nehmen an, dass die Prämissen der Induktionsregel für eine Aussageform A erfüllt sind:

(1) $A(nil)$

(2) $\forall x \in \mathbb{Z}\ \forall xr \in \mathcal{L}(\mathbb{Z}):\ A(xr) \Rightarrow A(x :: xr)$

Wir wollen uns nun überlegen, warum aus diesen Prämissen die Gültigkeit der Aussage

$$A(1 :: (2 :: (3 :: nil)))$$

folgt. Wegen Prämisse (1) gilt:

$$A(nil)$$

Daraus folgt mit Prämisse (2):

$$A(3 :: nil)$$

Daraus folgt wieder mit Prämisse (2):

$$A(\ 2 :: (3 :: nil)\ )$$

Daraus folgt wieder mit Prämisse (2):

$$A(\ 1 :: (2 :: (3 :: nil))\ )$$

Der Trick liegt also bei der zweiten Prämisse, die sagt, dass A für eine nichtleere Liste gültig ist, wenn A für den Rumpf der Liste gültig ist.

Wir zeigen jetzt mithilfe von Listeninduktion, dass die Konkatenationsfunktion @ assoziativ ist.

Proposition 2.3.1 *Sei X eine Menge und seien $xs, ys, zs \in \mathcal{L}(X)$. Dann gilt:*

$$(xs@ys)@zs = xs@(ys@zs)$$

Beweis Durch strukturelle Induktion über $xs \in \mathcal{L}(X)$.

Sei $xs = nil$. Dann:

$$\begin{aligned}(xs@ys)@zs &= (nil@ys)@zs \\ &= ys@zs && \text{Definition von @} \\ &= nil@(ys@zs) && \text{Definition von @} \\ &= xs@(ys@zs)\end{aligned}$$

Sei $xs = x :: xr$. Dann:

$$\begin{aligned}
(xs@ys)@zs &= ((x :: xr)@ys)@zs \\
&= (x :: (xr@ys))@zs && \text{Definition von @} \\
&= x :: ((xr@ys)@zs) && \text{Definition von @} \\
&= x :: (xr@(ys@zs)) && \text{Induktionsannahme} \\
&= (x :: xr)@(ys@zs) && \text{Definition von @} \\
&= xs@(ys@zs) && \square
\end{aligned}$$

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:

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

Übungen

Aufgabe 2.1 (Strukturelle Induktion für Listen) Sei X eine Menge und seien die Menge $\mathcal{L}(X)$ und die Funktionen

$$\begin{aligned}
@ &\in \mathcal{L}(X) \times \mathcal{L}(X) \to \mathcal{L}(X) \\
|_| &\in \mathcal{L}(X) \to \mathbb{N} \\
rev &\in \mathcal{L}(X) \to \mathcal{L}(X)
\end{aligned}$$

wie oben definiert. Beweisen Sie die folgenden Aussagen:

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

Dabei dürfen Sie annehmen, dass die Konkatenationsfunktion assoziativ ist:

$$\forall xs, ys, zs \in \mathcal{L}(X): \quad (xs@ys)@zs = xs@(ys@zs)$$

Aufgabe 2.2 (Strukturelle Induktion für N) Sei die Menge N rekursiv definiert:

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

Die Elemente von N können als Darstellungen der natürlichen Zahlen aufgefasst werden:

$$\begin{array}{ll} 0 & \emptyset \\ 1 & \{\emptyset\} \\ 2 & \{\{\emptyset\}\} \\ \ldots & \end{array}$$

Eine Additionsfunktion für N sei wie folgt definiert:

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

1. Definieren Sie die Menge N mithilfe von Inferenzregeln.
2. Definieren Sie eine Multiplikationsfunktion $\cdot \in N \times N \to N$.
3. Geben Sie die Induktionsregel für N an.
4. Beweisen Sie: $\forall x, y, z \in N: \quad (x + y) + z = x + (y + z)$.
5. Beweisen Sie: $\forall x \in N: \quad x + \emptyset = x$.
6. Beweisen Sie: $\forall x, y \in N: \quad x + y = y + x$.