Kapitel 5

Fixpunktoperatoren

Rekursion ist eine wichtige Definitionstechnik. Im Rahmen dieser Vorlesung haben wir bereits eine Vielzahl von rekursiven Definitionen gesehen. Dabei wurden überwiegend strukturelle Rekursionen verwendet.

Aus mathematischer Sicht stellt sich die Frage, welche Bedingungen ein rekursiver Definitionsversuch erfüllen muss, damit er zulässig ist. Wenn man dieser Frage nach geht, stellt man fest, dass sie sich nicht ohne weiteres beantworten lässt.

Mithilfe sogenannter Fixpunktoperatoren gelingt es, auch komplexe rekursive Definitionen auf einfache mathematische Konzepte zurückzuführen. Daher gehören Fixpunktoperatoren zur mathematischen Grundausstattung der Theoretischen Informatik.

Wir erproben die neuen Techniken mithilfe von zwei Fallstudien. Zuerst analysieren wir die rekursive Definition von Mengen mithilfe von Inferenzregeln. Ein Ergebnis unserer Analyse ist die Formulierung einer Beweistechnik, die als Regelinduktion bezeichnet wird.

Die zweite Fallstudie betrifft die rekursive Definition von Funktionen. Mithilfe von Fixpunktoperatoren ist es möglich, auch solche Definitionen zuzulassen, deren Rekursion nicht für alle Argumente terminiert.

Die Analyse von Rekursion mithilfe von Fixpunktoperatoren bereitet Anfängern zunächst erhebliche Verständnisprobleme. Das liegt zum Teil daran, dass sie eine völlig neue Sicht auf Rekursion eröffnet.

In Standard ML benutzen wir Rekursion, um Prozeduren und Typen zu definieren. Standard ML ist so konstruiert, dass rekursive Definitionen mathematisch gesehen immer zulässig sind.

Lesematerial

[Winskel, Kapitel 4.1, 4.4 und 5.4]

5.1 FP-Format und R-Strukturen

Sei X eine Menge und $f \in X \to X$ eine Funktion. Dann kann man versuchen, ein $x \in X$ als die Lösung der rekursiven Gleichung

$$x \stackrel{\text{def}}{=} f(x)$$

zu definieren. Wir sagen, dass ein solcher rekursiver Definitionsversuch im *FP-Format* vorliegt. Die Funktion f nennen wir das *Funktional* des Definitionsversuchs. Natürlich stellt sich die Frage, ob es ein x mit

$$x = f(x)$$

gibt (*Existenz*), und ob es eindeutig bestimmt ist (*Eindeutigkeit*). Wenn es genau ein x mit $x = f(x)$ gibt, dann ist die rekursive Definition ohne weitere Nebenbedingungen zulässig.

Sei $f \in X \to X$ eine Funktion. Ein $x \in X$ heißt *Fixpunkt von* f genau dann, wenn $x = f(x)$.

Das gerade vorgestellte Definitionsschema hat das Problem, dass die in der Praxis auftretenden Funktionale f typischerweise mehrere Fixpunkte haben. Dieses Problem lösen wir, indem wir zusätzlich einen sogenannten Fixpunktoperator festlegen, der für jedes Funktional f genau einen Fixpunkt auswählt. Das führt uns zum Begriff der R-Struktur.

Definition 5.1.1 *Ein* R-Struktur *ist ein Tripel* $\langle X, F, \mathit{fix}\rangle$ *wie folgt:*

1. X *ist eine Menge.*
2. $F \subseteq X \to X$.
3. $\mathit{fix} \in F \to X$ *so dass* $\forall f \in F: \ \mathit{fix}(f) = f(\mathit{fix}(f))$.

Die Funktion fix bezeichnen wir als den Fixpunktoperator der R-Struktur.

Sei $\langle X, F, \mathit{fix}\rangle$ eine R-Struktur und $f \in F$. Dann können wir eine rekursive Definition

$$x \stackrel{\text{def}}{=} f(x)$$

mithilfe des Fixpunktoperators der R-Struktur auf die nicht-rekursiven Definition

$$x \stackrel{\text{def}}{=} \mathit{fix}(f)$$

zurückführen. Wir werden sehen, dass dieses einfache Modell eine Vielzahl von rekursiven Definitionen erklären kann.

5.2 Ein Fixpunktoperator für Standard ML

Wir zeigen jetzt, dass man in Standard ML einen Fixpunktoperator definieren kann, mit dem sich rekursive Prozedurdeklarationen auf nicht-rekursive Deklarationen zurückführen lassen.

Als Beispiel betrachten wir die rekursive Deklaration

```
fun fac n = if n<2 then 1 else n * fac (n-1)
fac : int -> int
```

einer Prozedur, die die Fakultätsfunktion berechnet. Diese Deklaration bringen wir zunächst in die Form

```
val rec fac = fn n => if n<2 then 1 else n * fac (n-1)
```

Jetzt ist es nicht schwer, das Funktional für die FP-Form dieser Deklaration zu deklarieren:

```
val facFun = fn fac => fn n => if n<2 then 1 else n * fac (n-1)
facFun : (int -> int) -> (int -> int)
```

Damit können wir eine zu `fac` äquivalente Prozedur wie folgt deklarieren:

```
val rec fac' = facFun fac'
```

Leider ist diese Deklaration in Standard ML unzulässig, da die rechte Seite einer rekursiven Wertdeklaration eine Abstraktion sein muss.[1] Dieses Problem können wir durch das Einfügen einer eigentlich unnötigen Abstraktion beheben:

```
val rec fac' = fn x => facFun fac' x
fac' : int -> int
```

Überzeugen Sie sich durch Experimente, dass `fac'` in der Tat dieselbe Funktion berechnet wie `fac`.

Wir werden gleich zeigen, wie man einen Fixpunktoperator `fix` deklariert, der zu den Funktionalen rekursiver Prozeduren entsprechende Prozeduren liefert. Damit

[1] Diese Einschränkung vereinfacht die Implementierung von Standard ML.

werden wir in der Lage sein, eine Fakultätsprozedur ohne Rekursion zu deklarieren:

```
val fac'' = fix facFun
fac'' : int -> int
```

Den Fixpunktoperator deklarieren wir wie folgt:

```
fun fix f x = f (fix f) x
val fix : (('a -> 'b) -> ('a -> 'b)) -> ('a -> 'b)
```

Vorerst sollten Sie noch nicht versuchen, die Deklaration des Fixpunktoperators zu verstehen. Stattdessen sollten Sie sich mit dem Interpreter davon überzeugen, dass die obige Deklaration von `fac"` tatsächlich eine Fakultätsprozedur liefert.

Wir illustrieren die gerade gezeigte Transformation von rekursiven Prozedurdeklarationen an einem zweiten Beispiel:

```
fun potenz(x,n) = if n<1 then 1 else x * potenz(x,n-1)
val potenz : int * int -> int
```

Zuerst deklarieren wir das zu dieser Deklaration gehörige Funktional:

```
val potenzFun = fn potenz => fn (x,n) =>
                   if n<1 then 1 else x * potenz(x,n-1)
val potenzFun : (int * int -> int) -> (int * int -> int)
```

Durch Anwendung des Fixpunktoperator auf das Funktional

```
val potenz' = fix potenzFun
val potenz' : int * int -> int
```

bekommen wir eine zu `potenz` äquivalente Prozedur `potenz'`.

Wahrscheinlich wird es Ihnen vorerst nicht gelingen, vollständig zu verstehen, warum die Prozedur `fix` das richtige Ergebnis liefert. Das ist auch nicht notwendig. Es genügt, wenn Sie verstehen, wie man in Standard ML rekursive Prozedurdeklarationen mithilfe eines gegebenen Fixpunktoperators `fix` auf nichtrekursive Deklarationen zurückführen kann.

Gegeben zwei Typen t_1 und t_2, verhält sich das Triple

$$\langle t_1 \to t_2,\ (t_1 \to t_2) \to (t_1 \to t_2),\ \texttt{fix} \rangle$$

so wie eine R-Struktur. Durch Experimente können Sie sich davon überzeigen, dass für jede Prozedur

$$f:\ (t_1 \to t_2) \to (t_1 \to t_2)$$

die zwei Prozeduren `fix` f und f (`fix` f) dieselbe Funktion berechnen. Die Prozedur `fix` verhält sich also tatsächlich wie ein Fixpunktoperator.

Man kann auch polymorphe Prozeduren mit dem Fixpunktoperator `fix` deklarieren. Allerdings muss man dabei eine redundante Abstraktion einfügen, um ambige Deklarationen[2] zu vermeiden:

```
val foo' = fn x => (fix foofun) x
```

5.3 Vollständige partielle Ordnungen

Wir zeigen jetzt, wie man mit mathematischen Mitteln R-Strukturen konstruiert. Dazu benötigen wir das Konzept einer vollständigen partiellen Ordnung, kurz VPO genannt.

Sei $\langle X, \leq \rangle$ eine partiell geordnete Menge. Eine *Kette* ist eine unendliche Folge $x_0, x_1, x_2, \ldots$ von Elementen von X für die gilt: $x_0 \leq x_1 \leq x_2 \leq \cdots$. Wenn das Supremum (kleinste obere Schranke) einer Kette $x_0 \leq x_1 \leq x_2 \leq \cdots$ existiert, bezeichnen wir es mit

$$\bigsqcup_{i \in \mathbb{N}} x_i$$

Eine Funktion $f \in X \to X$ heißt *monoton* genau dann, wenn für alle $x, y \in X$: $x \leq y \Rightarrow f(x) \leq f(y)$.

Eine *VPO* (*vollständige partielle Ordnung*) ist eine partiell geordnete Menge $\langle X, \leq \rangle$, sodass jede aufsteigende Kette $x_0 \leq x_1 \leq x_2 \leq \cdots$ von Elementen von X ein Supremum hat.

Das Standardbeispiel für VPOs sind Potenzmengen mit der Inklusionsordnung. Eine Kette $X_0 \subseteq X_1 \subseteq \ldots$ von Teilmengen einer Menge X hat dabei das folgende Supremum:

$$\bigsqcup_{i \in \mathbb{N}} X_i \;=\; \bigcup_{i \in \mathbb{N}} X_i \;=\; \{\, x \in X \mid \exists i \in \mathbb{N}\colon\; x \in X_i \,\}$$

Proposition 5.3.1 *Sei $\langle X, \leq \rangle$ eine VPO und $f \in X \to X$ monoton. Dann gilt für jede Kette $x_0 \leq x_1 \leq \ldots$ in X:*

(1) $f(x_0) \leq f(x_1) \leq \ldots$ *ist eine Kette in* X.

(2) $f(\bigsqcup_{i \in \mathbb{N}} x_i) \geq \bigsqcup_{i \in \mathbb{N}} f(x_i)$

[2] Ambige Deklarationen führen zur Fehlermeldung „value polymorphism“.

Beweis Sei $x_0 \leq x_1 \leq \ldots$ eine Kette in X. Da f monoton ist, gilt (1). Aus der Monotonie von f folgt auch, dass $f(x_n) \leq f(\bigsqcup_{i\in\mathbb{N}} x_i)$ für alle $n \in \mathbb{N}$ gilt. Also ist $f(\bigsqcup_{i\in\mathbb{N}} x_i)$ eine obere Schranke für die Kette $f(x_0) \leq f(x_1) \leq \ldots$. Da $\bigsqcup_{i\in\mathbb{N}} f(x_i)$ das Supremum dieser Kette ist, folgt (2). □

Sei $\langle X, \leq\rangle$ eine VPO. Eine monotone Funktion $f \in X \to X$ heißt *stetig* genau dann, wenn für jede Kette $x_0 \leq x_1 \leq \ldots$ in X die Ungleichung

$$f(\bigsqcup_{i\in\mathbb{N}} x_i) \leq \bigsqcup_{i\in\mathbb{N}} f(x_i)$$

gilt. Die Menge aller stetigen Funktionen in $X \to X$ bezeichnen wir mit $[X \to X]$.

Proposition 5.3.2 *Sei $\langle X, \leq\rangle$ eine VPO und $f \in X \to X$ stetig. Dann gilt für jede Kette $x_0 \leq x_1 \leq \ldots$ in X:*

$$f(\bigsqcup_{i\in\mathbb{N}} x_i) = \bigsqcup_{i\in\mathbb{N}} f(x_i)$$

Diese Eigenschaft bezeichnen wir als Distributivität.

Proposition 5.3.3 *Sei $\langle X, \leq\rangle$ eine partiell geordnete Menge. Wenn X endlich ist, gilt:*

1. *$\langle X, \leq\rangle$ eine VPO.*
2. *Jede monotone Funktion $X \to X$ ist stetig.*

Für jede Funktion $f \in X \to X$ und jedes $n \in \mathbb{N}$ definieren wir:

$$f^n \in X \to X$$
$$f^n(x) = \text{if } n = 0 \text{ then } x \text{ else } f(f^{n-1}(x))$$

Der Fixpunktsatz formuliert die wichtigste Eigenschaft von VPOs.

Satz 5.3.4 (Fixpunktsatz) *Sei $\langle X, \leq\rangle$ eine VPO mit einem kleinsten Element $\bot \in X$ (sprich bottom). Weiter sei $f \in X \to X$ eine stetige Funktion. Dann hat f einen eindeutig bestimmten kleinsten Fixpunkt x_0. Zusätzlich gilt:*

1. *$f^0(\bot) \leq f^1(\bot) \leq f^2(\bot) \leq \cdots$ ist eine Kette.*
2. *$x_0 = \bigsqcup_{n\in\mathbb{N}} f^n(\bot)$*
3. *x_0 ist das kleinste x mit $f(x) \leq x$.*

Beweis Da f monoton ist und $\bot = f^0(\bot) \leq f^1(\bot)$, ist

$$f^0(\bot) \leq f^1(\bot) \leq f^2(\bot) \leq \cdots$$

eine Kette. Die Stetigkeit von f liefert

$$f(x_0) = f(\bigsqcup_{n\in\mathbb{N}} f^n(\bot)) = \bigsqcup_{n\in\mathbb{N}} f(f^n(\bot)) = \bigsqcup_{n\in\mathbb{N}^+} f^n(\bot) = \bigsqcup_{n\in\mathbb{N}} f^n(\bot) = x_0$$

Damit ist x_0 ein Fixpunkt von f. Die noch zu beweisende, zweite Behauptung des Satzes impliziert, dass x_0 der kleinste Fixpunkt von f ist.

Um die zweite Behauptung zu zeigen, nehmen wir an, dass $f(x) \leq x$ gilt. Wir müssen zeigen, dass $x_0 \leq x$ gilt. Da x_0 das Supremum der Kette $f^0(\bot) \leq f^1(\bot) \leq f^2(\bot) \leq \cdots$ ist, genügt es zu zeigen, dass

$$\forall n \in \mathbb{N}\colon\ f^n(\bot) \leq x$$

gilt. Diese Behauptung zeigen wir durch vollständige Induktion.

Sei $n = 0$. Dann $f^0(\bot) = \bot \leq x$.

Sei $n > 0$. Nach Induktionsannahme gilt $f^{n-1}(\bot) \leq x$. Also folgt mit der Monotonie von f: $f^n(\bot) = f(f^{n-1}(\bot)) \leq f(x) \leq x$. □

Sei $\langle X, \leq\rangle$ eine VPO mit einem kleinsten Element $\bot$. Dann heißt die Funktion

$$\mathit{fix} \in [X \to X] \to X$$

$$\mathit{fix}\ f\ = \bigsqcup_{n\in\mathbb{N}} f^n(\bot)$$

der *Fixpunktoperator* der VPO.

Korollar 5.3.5 *Sei $\langle X, \leq\rangle$ eine VPO mit einem kleinsten Element und sei fix der Fixpunktoperator der VPO. Dann ist $\langle X,\ [X \to X],\ \mathit{fix}\rangle$ eine R-Struktur.*

Die mit VPOs konstruierbaren R-Strukturen eignen sich für die Fixpunkt-Behandlung fast aller in der Praxis vorkommenden rekursiven Definitionen. Wir stützen diese Behauptung durch zwei Fallstudien.

5.4 Fallstudie: Rekursive Definition von Mengen

Wir zeigen jetzt, wie man die rekursive Definition einer Menge so umformulieren kann, dass die Menge mithilfe des Fixpunktoperators einer VPO definiert wird.

Wir betrachten Mengen, die mithilfe von Inferenzregeln definiert werden. Bisher haben wir nicht genau erklärt, was wir unter einer Inferenzregel verstehen wollen. Um dies zu tun, müssten wir eine logische Sprache einführen, mit der die Prämissen und Konklusionen von Inferenzregeln formuliert werden können. Das ist nicht so einfach. Wir sind aber auch mit einfacheren Mittel in der Lage, mehr über die rekursive Definition von Mengen mit Inferenzregeln zu sagen.

Sei X eine Menge. Eine *Grundregel für* X ist ein Paar $\langle A, x\rangle$, dass aus einer endlichen Menge $A \subseteq X$ und einem $x \in X$ besteht. Beachten Sie, dass die Menge A endlich sein muss.

Stellen Sie sich nun eine Menge von Inferenzregeln vor, mit denen eine Teilmenge $I \subseteq X$ definiert werden soll. Dann können wir uns jede Inferenzregel als ein Schema vorstellen, dass eine Menge von Grundregeln definiert. Als Beispiel wählen wir

$$\frac{}{3 \in P} \qquad \frac{}{5 \in P} \qquad \frac{x \in P \quad y \in P \quad z = x \cdot y}{z \in P}$$

Hier ist $X = \mathbb{Z}$. Die erste und zweite Inferenzregel definieren je genau eine Grundregel für $\mathbb{Z}$, nämlich

$$\langle \emptyset, 3\rangle \quad \text{und} \quad \langle \emptyset, 5\rangle$$

Die dritte Inferenzregel definiert dagegen eine unendliche Menge von Grundregeln:

$$\{\, \langle \{x, y\}, z\rangle \mid x, y \in \mathbb{Z} \wedge z = x \cdot y \,\}$$

Beachten Sie, dass einige dieser Grundregeln im Gegensatz zur Inferenzregel nur eine Prämisse haben. Die Prämissen einer Inferenzregel teilen wir in *echte Prämissen* (zum Beispiel $x \in P$) und *Seitenbedingungen* (zum Beispiel $z = x \cdot y$) ein. Nur die echten Prämissen einer Inferenzregel können als Prämissen der Grundregeln auftauchen.

Wir können jetzt die rekursive Definition einer Menge mit Inferenzregeln in zwei Schritten beschreiben. Im ersten Schritt leiten wir aus dem Kontext und den Inferenzregeln eine *Grundmenge* X und eine Menge R von Grundregeln für X ab. Dazu braucht man mathematische Erfahrung, um die Inferenzregel richtig zu verstehen. Der zweite Schritt legt zu R eine Menge $I_R \subseteq X$ fest, die als die durch die Inferenzregel definierte Menge zu verstehen ist. Den zweiten Schritt können wir vollständig formal beschreiben. Dabei werden wir die Menge I_R ohne Rekursion mithilfe eines Fixpunktoperators definieren.

Sei R eine Menge von Grundregeln für X. Dann definieren wir eine Funktion

$$\hat{R} \in \mathcal{P}(X) \to \mathcal{P}(X)$$
$$\hat{R}(A) = \{\, x \mid \exists B \subseteq A\colon\ \langle B, x\rangle \in R \,\}$$

Die Menge $\hat{R}(A)$ enthält offensichtlich genau die Objekte, die man durch einmaliges Anwenden einer Regel in R aus den Elementen von A konstruieren kann.

Offensichtlich ist $\hat{R}$ bezüglich der Inklusionsordnung auf $\mathcal{P}(X)$ monoton. Also haben wir

$$\emptyset \subseteq \hat{R}(\emptyset) \subseteq \hat{R}^2(\emptyset) \subseteq \ldots$$

Machen Sie sich jetzt klar, dass $\hat{R}^n(\emptyset)$ genau die Objekte enthält, die man durch höchstens n-mal iterierte Regelanwendung konstruieren kann.

Die Menge I_R können wir jetzt wie folgt definieren:

$$I_R \stackrel{\text{def}}{=} \bigcup_{i\in\mathbb{N}} \hat{R}^i(\emptyset)$$

Offensichtlich enthält I_R genau die Objekte, die man mit den Regeln aus R konstruieren kann.

Proposition 5.4.1 *Sei R eine Menge von Grundregeln für X. Dann ist $\hat{R}$ eine stetige Funktion $\mathcal{P}(X) \to \mathcal{P}(X)$ (bezüglich der VPO $\langle \mathcal{P}(X), \subseteq\rangle$).*

Beweis Wir wissen bereits, dass $\hat{R}$ monoton ist. Sei $A_0 \subseteq A_1 \subseteq \ldots$ eine Kette in $\mathcal{P}(X)$. Wir müssen die folgende Ungleichung zeigen:

$$\hat{R}(\bigcup_{i\in\mathbb{N}} A_i) \subseteq \bigcup_{i\in\mathbb{N}} \hat{R}(A_i)$$

Sei $x \in \hat{R}(\bigcup_{i\in\mathbb{N}} A_i)$. Dann existiert eine Regel $(B, x) \in R$ mit $B \subseteq \bigcup_{i\in\mathbb{N}} A_i$. Da B endlich ist, gibt es ein n, so dass $B \subseteq A_n$. Also $x \in \hat{R}(A_n) \subseteq \bigcup_{i\in\mathbb{N}} \hat{R}(A_i)$.

Beachten Sie, dass der Beweis ausnützt, dass jede Regel nur endlich viele Prämissen hat. □

Sei R eine Menge von Grundregeln für X. Wir sagen, dass eine Menge $A \subseteq X$ *unter R abgeschlossen* ist genau dann, wenn $\hat{R}(A) \subseteq A$. Abgeschlossenheit einer Menge X unter einer Regelmenge R bedeutet, dass man mit den Regeln in R aus den Objekten in X nur Objekte ableiten kann, die bereits in X sind.

Der Fixpunktsatz liefert uns jetzt die folgende Proposition.

Proposition 5.4.2 *Sei R eine Menge von Grundregeln für X. Sei fix der Fixpunktoperator der VPO $\langle \mathcal{P}(X), \subseteq \rangle$. Dann gilt:*

1. $I_R = fix(\hat{R})$.
2. $I_R = \hat{R}(I_R)$.
3. *I_R ist die kleinste Teilmenge von X, die unter R abgeschlossen ist.*

Die nächste Proposition formuliert ein wichtige Beweistechnik für rekursiv definierte Mengen, die als *Regelinduktion* bezeichnet wird. Unsere erste Anwendung dieser Technik wird der Beweis von Proposition 6.3.1 sein.

Proposition 5.4.3 (Regelinduktion) *Sei R eine Menge von Grundregeln für X und $P \subseteq X$. Dann $I_R \subseteq P$, wenn gilt: $\forall (A, x) \in R\colon\ A \subseteq P \Rightarrow x \in P$.*

Beweis Gelte $\forall (A, x) \in R\colon\ A \subseteq P \Rightarrow x \in P$. Dann ist P unter R abgeschlossen. Mit Proposition 5.4.2 (3) folgt, dass $I_R \subseteq P$. □

5.5 Fallstudie: Rekursive Definition von Funktionen

Wir zeigen jetzt, wie man die rekursive Definition einer Funktion so umformulieren kann, dass die Funktion mithilfe des Fixpunktoperators einer VPO definiert wird.

Approximationen für rekursive Prozeduren

Zuerst zeigen wir, dass wir rekursive Prozeduren mit nicht-rekursiven Prozeduren beliebig gut approximieren können.

Als Beispiel betrachten wir wieder die Fakultätsprozedur:

```
fun fac x = if x<2 then 1 else x * fac (x-1)
fac : int -> int
```

Das `fac` entsprechende Funktional deklarieren wir wie folgt:

```
val facFun = fn fac => fn x => if x<2 then 1 else x * fac (x-1)
facFun : (int -> int) -> (int -> int)
```

Wir konstruieren jetzt eine Folge von nicht-rekursiven Prozeduren, die die rekursive Prozedur `fac` zunehmend besser approximieren. Jede Approximation ist eine Prozedur `int -> int`, die entweder dasselbe Ergebnis wie `fac` liefert oder die Ausnahme

```
exception Approx
```

wirft. Die nullte Approximation

```
fun fac0 (x:int):int = raise Approx
```

wirft für alle Argumente die Ausnahme `Approx`. Die erste, zweite und dritte Approximation deklarieren wir wie folgt:

```
val fac1 = facFun fac0
val fac2 = facFun fac1
val fac3 = facFun fac2
```

Die erste Approximation `fac1` stimmt genau für die Argumente mit `fac` überein, für die `fac` mit der Rekursionstiefe null rechnet. Die zweite Approximation `fac2` stimmt genau für die Argumente mit `fac` überein, für die `fac` mit der Rekursionstiefe eins rechnet. Allgemein gilt, dass die n-te Approximation genau für die Argumente mit `fac` übereinstimmt, für die `fac` mit der Rekursionstiefe $n-1$ rechnet. Die rekursive Prozedur `fac` können wir also als den „Grenzwert“ der Approximationen auffassen.

Die VPO $X \to Y_\perp$

Wir übertragen jetzt die gerade vorgeführte Approximationsidee für Prozeduren auf Funktionen.

Seien X und Y Mengen und sei $\perp \notin Y$. Wir definieren:

$$Y_\perp \stackrel{\text{def}}{=} Y \cup \{\perp\}$$

Wir betrachten die Funktionen $f \in X \to Y_\perp$. Dabei soll das spezielle Ergebnis $\perp$ dem Werfen der Ausnahme `Approx` entsprechen. Für die Funktionen in $X \to Y_\perp$ definieren wir die folgende partielle Ordnung:

$$f \leq g \stackrel{\text{def}}{\iff} \forall x \in X\colon\ f(x) = \perp \vee f(x) = g(x).$$

Intuitiv bedeutet $f \leq g$, dass die Funktion f die Funktion g approximiert. Die Funktion $\lambda\, x \in X\,.\,\perp$ approximiert jede Funktion $X \to Y_\perp$. Die Funktionen $X \to Y$ sind die maximalen Elemente der Approximationsordnung.

Proposition 5.5.1 *$\langle X \to Y_\perp,\ \leq\rangle$ ist eine VPO mit dem kleinsten Element $\lambda\, x \in X\,.\,\perp$. Wenn $f_0 \leq f_1 \leq \ldots$ eine Kette in $X \to Y_\perp$ ist, dann ist die Funktion $f \in X \to Y_\perp$ mit*

$$f(x) \stackrel{\text{def}}{=} \begin{cases} y & \text{falls } \exists i \in \mathbb{N}\colon\ f_i(x) = y \in Y \\ \perp & \text{falls } \forall i \in \mathbb{N}\colon\ f_i(x) = \perp \end{cases}$$

das Supremum für die Kette.

Beweis Offensichtlich ist $\leq$ eine partielle Ordnung auf $X \to Y_\perp$. Es ist auch klar, dass $\lambda\, x \in X \,.\, \perp$ das kleinste Element ist. Sei $f_0 \leq f_1 \leq \ldots$ eine Kette in $X \to Y_\perp$. Die Definition der Ordnung erzwingt, dass sich die Glieder der Kette nur dadurch unterscheiden, dass spätere Glieder für ein $x \in X$ statt dem bisherigen $\perp$ ein $y \in Y$ liefern dürfen. Wenn ein Glied für ein $x \in X$ bereits ein $y \in Y$ liefert, müssen alle nachfolgenden Glieder für x genau dieses y liefern. Also ist die Funktion f das Supremum der Kette. □

Proposition 5.5.2 *Für jede Kette $f_0 \leq f_1 \leq \cdots$ in $X \to Y_\perp$ gilt:*

1. $\forall x \in X\ \exists n \in \mathbb{N}\colon\ (\bigsqcup_{i\in\mathbb{N}} f_i)(x) = f_n(x)$
2. $\forall x \in X\ \forall n \in \mathbb{N}\colon\ f_n(x) \neq \perp\ \Rightarrow\ (\bigsqcup_{i\in\mathbb{N}} f_i)(x) = f_n(x)$

Wir geben jetzt eine hinreichende Bedingung für die Stetigkeit von monotonen Funktionalen an, die in der Praxis leicht zu verifizieren ist. Salopp gesprochen sagt diese Bedingung, dass ein monotones Funktional F stetig ist, wenn für die Bestimmung von $F\,f\,x$ immer nur endlich viele Werte $f(y_1), \ldots, f(y_n)$ erforderlich sind.

Eine Funktion $f \in X \to Y_\perp$ heißt *finitär*, wenn es nur endlich viele $x \in X$ mit $f(x) \neq \perp$ gibt.

Proposition 5.5.3 *Sei $g \in X \to Y_\perp$ finitär und $f_0 \leq f_1 \leq \cdots$ eine Kette in $X \to Y_\perp$ mit $g \leq \bigsqcup_{i\in\mathbb{N}} f_i$. Dann existiert ein $n \in \mathbb{N}$ mit $g \leq f_n$.*

Beweis Durch Induktion über die Anzahl k der $x \in X$ mit $gx \neq \perp$. Wenn $k = 0$, dann $g = \lambda\, x \in X \,.\, \perp$. Also $g \leq f_n$ für alle $n \in \mathbb{N}$. Sei $k > 0$. Wir wählen ein $x \in X$ mit $gx \neq \perp$. Da $\perp \neq g(x) = (\bigsqcup_{i\in\mathbb{N}} f_i)(x)$, existiert ein $m \in \mathbb{N}$ mit $g(x) = f_m(x)$. Da $g[\perp/x] \leq \bigsqcup_{i\in\mathbb{N}} f_i$, existiert nach Induktionsannahme ein $l \in \mathbb{N}$ mit $g[\perp/x] \leq f_l$. Also $g \leq f_n$ für $n = \max\{m, l\}$. □

Proposition 5.5.4 *Sei $F \in (X \to Y_\perp) \to (X \to Y_\perp)$ monoton. Dann ist F stetig, wenn die folgende Bedingung gilt:*

$$\forall f \in X \to Y_\perp\ \forall x \in X\ \exists g \in X \to Y_\perp\colon\ g \text{ finitär} \wedge g \leq f \wedge F\,f\,x = F\,g\,x$$

Beweis Sei $f_0 \leq f_1 \leq \cdots$ eine Kette in $X \to Y_\perp$ und $x \in X$. Sei $F(\bigsqcup_{i\in\mathbb{N}} f_i)(x) \neq \perp$. Es genügt zu zeigen, dass ein $n \in \mathbb{N}$ existiert mit $F(\bigsqcup_{i\in\mathbb{N}} f_i)(x) = F(f_n)(x)$. Nach Voraussetzung gibt es ein finäters $g \leq \bigsqcup_{i\in\mathbb{N}} f_i$

mit $F(\bigsqcup_{i\in\mathbb{N}} f_i)(x) = F(g)(x)$. Da g finitär ist und $g \leq \bigsqcup_{i\in\mathbb{N}} f_i$, existiert ein $n \in \mathbb{N}$ mit $g \leq f_n$ (Proposition 5.5.3). Mit der Monotonie von F folgt $F(g) \leq F(f_n)$. Da $F(g)(x) \neq \bot$, folgt $F(g)(x) = F(f_n)(x)$. Also $F(\bigsqcup_{i\in\mathbb{N}} f_i)(x) = F(f_n)(x)$. □

Rückführung von rekursiven Funktionsdefinitionen

Eine rekursive Definition

$$f \in X \to Y_\bot$$
$$f(x) = a$$

einer Funktion f können wir mathematisch einfacher als die nicht-rekursive Definition

$$F \in (X \to Y_\bot) \to (X \to Y_\bot)$$
$$F(f) = \lambda x \in X \,.\, a$$

des entsprechenden Funktionals auffassen. Die Definition von f betrachten wir nur dann als zulässig, wenn das Funktional F stetig ist (bezüglich der VPO $\langle X \to Y_\bot, \leq\rangle$). Die Funktion f definieren wir dann

$$f \stackrel{\text{def}}{=} \mathit{fix}\, F = \bigsqcup_{n\in\mathbb{N}} F^n(\lambda x \in X \,.\, \bot)$$

als das Supremum der approximierenden Kette

$$F^0(\lambda x \in X \,.\, \bot) \leq F^1(\lambda x \in X \,.\, \bot) \leq F^2(\lambda x \in X \,.\, \bot) \leq \ldots$$

Aus unseren Vorüberlegungen folgt, dass das Supremum dieser Kette in der Tat die Funktion ist, die wir mit der rekursiven Ausgangsdefinition definieren wollen.

Der Fixpunktsatz liefert uns zwei nützliche Eigenschaften von f:

1. $f = F(f)$.
2. $\forall g \in X \to Y_\bot\colon\ F(g) \leq g \Rightarrow f \leq g$.

Beispiel Wir betrachten die folgende rekursive Definition:

$$\mathit{fac} \in \mathbb{Z} \to \mathbb{Z}_\bot$$
$$\mathit{fac}(x) = \text{if } x = 0 \text{ then } 1 \text{ else if } \mathit{fac}(x-1) = \bot \text{ then } \bot \text{ else } y \cdot \mathit{fac}(x-1)$$

Bisher hätten wir diese Definition nicht zugelassen, da die Rekursion für negative Argumente divergiert. Mithilfe der gerade entwickelten Fixpunkttechnik können

wir die Definition aber so interpretieren, dass sie tatsächlich genau eine totale Funktion $\mathbb{Z} \to \mathbb{Z}_\perp$ liefert. Dazu betrachten wir das Funktional der Definition:

$$\begin{aligned} &Fac \in (\mathbb{Z} \to \mathbb{Z}_\perp) \to (\mathbb{Z} \to \mathbb{Z}_\perp) \\ &Fac(fac) \;=\; \lambda\, x \in \mathbb{Z}.\;\; \text{if } x = 0 \text{ then } 1 \\ &\qquad\qquad\qquad\qquad \text{else if } fac(x-1) = \perp \text{ then } \perp \text{ else } x \cdot fac(x-1) \end{aligned}$$

Die Monotonie von *Fac* ist offensichtlich. Die Stetigkeit folgt mit Proposition 5.5.4, da $Fac(fac)x$ die Funktion *fac* nur für das Argument $x - 1$ benötigt. Damit ist die Definition von *fac* zulässig und kann wie folgt formuliert werden:

$$fac \stackrel{\text{def}}{=} fix\, Fac \;=\; \bigsqcup_{n \in \mathbb{N}} Fac^n(\lambda\, x \in \mathbb{Z}.\, \perp)$$

Man kann leicht zeigen, dass $fac(x) \in \mathbb{N}$ für alle $x \in \mathbb{N}$ gilt. □

Divergenz und $\perp$

Betrachten Sie die Funktionsdefinition

$$\begin{aligned} &f \in \mathbb{Z} \to \mathbb{Z}_\perp \\ &f(x) = f(x) \end{aligned}$$

Das entsprechende Funktional

$$\begin{aligned} &F \in (\mathbb{Z} \to \mathbb{Z}_\perp) \to (\mathbb{Z} \to \mathbb{Z}_\perp) \\ &F(f) \;=\; \lambda\, x \in \mathbb{Z}.\, f(x) \end{aligned}$$

ist stetig und liefert die Funktion

$$f \;=\; fix\, F \;=\; \lambda\, x \in \mathbb{Z}.\, \perp$$

die für alle Argumente $\perp$ liefert. Als Prozedurdefinition betrachtet, definiert die obige Definition eine Prozedur, die für alle Argumente divergiert. Das Objekt $\perp$ entspricht also der Divergenz von Prozeduren.

Im einleitenden Beispiel dieses Abschnitts haben wir die Approximationen der Fakultätsprozedur mithilfe der Ausnahme `Approx` realisiert. Alternativ haben wir die Möglichkeit, das Werfen der Ausnahme `Approx` durch eine divergierende Berechnung zu ersetzen. Diese Alternative passt besser zu unserem mathematischen Modell, da sie das Objekt $\perp$ in jedem Fall als Divergenz interpretiert. Wir können diese Alternative realisieren, indem wir die nullte Approximation wie folgt deklarieren:

```
fun fac0 (x:int):int = fac0 x
```

Partialität und $\perp$

Wir können die partiellen Funktionen in

$$X \rightharpoonup Y$$

durch die totalen Funktionen in

$$X \to Y_\perp$$

darstellen. Eine partielle Funktion $f \in X \rightharpoonup Y$ stellen wir dabei durch die totale Funktion

$$f' \in X \to Y_\perp$$
$$f'(x) = \text{if } x \in \mathit{Dom}\ f \text{ then } f(x) \text{ else } \perp$$

dar. Das bedeutet, dass

$$f'(x) = \perp$$

genau dann gilt, wenn f auf x undefiniert ist. Diese Zuordnung definiert eine Bijektion von $X \rightharpoonup Y$ nach $X \to Y_\perp$, für die gilt:

$$\forall f, g \in X \rightharpoonup Y: \quad f \subseteq g \iff f' \leq g'$$