3 Grundlagen der funktionalen Programmierung

In diesem Kapitel wollen wir wichtige Begriffe der deklarativen Programmierung formalisieren. Hierbei sind u.a. die folgenden Bereiche relevant:

Reduktionssysteme: Hiermit werden ganz allgemein Begriffe wie Reduktionsfolgen, Normalformen, Terminierung u.ä. definiert, die man dann in verschiedenen Arten von Reduktionssystemen (Termersetzung, λ-Kalkül) anwenden kann.

Termersetzung: Hiermit werden das Rechnen mit Pattern matching und passende Auswertungsstrategien definiert.

3.1 Reduktionssysteme

Rechnen in funktionalen Sprachen kann aufgefasst werden als das fortlaufende Anwenden von Reduktionsschritten. Um hierfür einige allgemeine Begriffe festzulegen, betrachten wir hier *allgemeine (abstrakte) Reduktionssysteme*. Die damit einhergehenden Begriffe werden z.B. beim λ-Kalkül, Termersetzung und der Logikprogrammierung wiederverwendet.

Definition 3.1 (Reduktionssystem) *Sei M eine Menge und $\to$ eine binäre (zweistellige) Relation auf M, d.h. $\to \subseteq M \times M$ und wir schreiben $e_1 \to e_2$ falls $(e_1, e_2) \in \to$. In diesem Fall heißt $(M, \to)$* ***Reduktionssystem****.*

Wir definieren verschiedene Erweiterungen der Relation $\to$:

- $x \to^0 y \;:\Leftrightarrow\; x = y$
- $x \to^i y \;:\Leftrightarrow\; \exists z :\; x \to^{i-1} z$ *und* $z \to y$ *(i-faches Produkt von $\to$)*
- $x \to^+ y \;:\Leftrightarrow\; \exists i > 0 :\; x \to^i y$ *(transitiver Abschluss)*
- $x \to^* y \;:\Leftrightarrow\; \exists i \geq 0 :\; x \to^i y$ *(reflexiv-transitiver Abschluss)*

Außerdem definieren wir von $\to$ abgeleitete Relationen:

- $x \leftarrow y \;:\Leftrightarrow\; y \to x$
- $x \leftrightarrow y \;:\Leftrightarrow\; x \to y$ *oder* $x \leftarrow y$

Insbesondere ist $x \leftrightarrow^* y$ der symmetrische, reflexiv-transitive Abschluss bzw. die kleinste Äquivalenzrelation, die $\rightarrow$ enthält.

Definition 3.2 *Sei $(M, \rightarrow)$ ein festes Reduktionssystem.*

- *$x \in M$ heißt **reduzierbar** $:\Leftrightarrow$ $\exists y$ mit $x \rightarrow y$*
- *$x \in M$ heißt **irreduzibel** oder in **Normalform**, falls x nicht reduzierbar ist.*
- *y ist eine **Normalform von** x $:\Leftrightarrow$ $x \rightarrow^* y$ und y Normalform (irreduzibel).*
  Falls x nur eine Normalform y hat, dann schreiben wir auch: $x\!\downarrow := y$
- *$x \downarrow y$ (x und y sind **zusammenführbar**) $:\Leftrightarrow$ $\exists z$ mit $x \rightarrow^* z$ und $y \rightarrow^* z$*

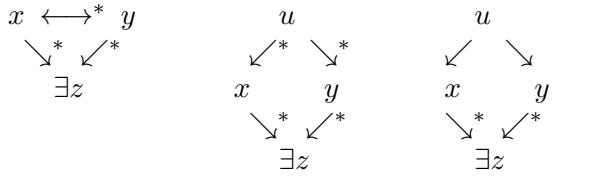
In der funktionalen Programmierung sollten Funktionen „sinnvoll“ definiert sein, d.h. die zu berechnende Ausdrücke sollten höchstens eine Normalform haben, damit das Ergebnis eindeutig bestimmt ist.

Um dies zu garantieren, muss das Reduktionssystem weitere Forderungen erfüllen. Zu diesem Zweck definieren wir einige weitere Begriffe:

Definition 3.3 *Sei $(M, \rightarrow)$ ein Reduktionssystem.*

1. *$\rightarrow$ heißt **Church-Rosser** (CR), falls $\leftrightarrow^* \subseteq \downarrow$ (d.h. $x \leftrightarrow^* y$ impliziert $x \downarrow y$).*
2. *$\rightarrow$ heißt **konfluent**, falls $\forall u, x, y$ mit $u \rightarrow^* x$ und $u \rightarrow^* y$ ein z existiert mit $x \rightarrow^* z$ und $y \rightarrow^* z$.*
3. *$\rightarrow$ heißt **lokal konfluent**, falls $\forall u, x, y$ mit $u \rightarrow x$ und $u \rightarrow y$ ein z existiert mit $x \rightarrow^* z$ und $y \rightarrow^* z$*

Graphisch:



CR konfluent lokal konfluent

Diese Begriffe haben die folgende Intuition:

Church-Rosser: Die Äquivalenz von Ausdrücken kann auch durch gerichtete Reduktionen entschieden werden.

Konfluenz: Unterschiedliche Berechnungswege spielen keine Rolle.

Lokale Konfluenz: Unterschiedliche lokale Berechnungswege spielen keine Rolle.

Aus der Definition der Konfluenz ist unmittelbar klar, dass eine konfluente Reduktionsrelation garantiert, dass jedes Element höchstens eine Normalform hat. Aus diesem Grund ist die *Konfluenz wünschenswert* für die effiziente Implementierung funktionaler Sprachen (da man dann nicht in beliebiger Richtung nach Normalformen suchen muss). Allerdings ist „lokal konfluent“ i.allg. leichter prüfbar als „konfluent“. Daher stellt sich die Frage, welche Zusammenhänge es zwischen diesen Begriffen gibt. Der folgende Satz gibt darüber Auskunft:

Satz 3.1

1. *„$\rightarrow$ ist CR“ genau dann wenn „$\rightarrow$ ist konfluent“*
2. *Aus „$\rightarrow$ ist konfluent“ folgt „$\rightarrow$ ist lokal konfluent“*

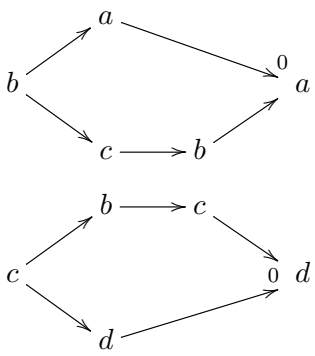
Die Umkehrung von 2 gilt leider nicht, wie das folgende Beispiel zeigt:

Sei die Relation $\rightarrow$ definiert durch:

$b \rightarrow a$
$b \rightarrow c$
$c \rightarrow b$
$c \rightarrow d$

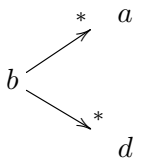
Graphisch: $a \leftarrow b \leftrightarrows c \rightarrow d$

Bei diesem Reduktionssystem gibt es zwei lokale Divergenzen:



Da alle lokalen Divergenzen zusammenführbar sind, ist das Reduktionssystem lokal konfluent.

Das Reduktionssystem ist aber nicht konfluent:



aber es gilt nicht: $a \downarrow d$.

Das Problem in diesem Beispiel ist die „Schleife“ zwischen b und c. Daher definieren wir zunächst Reduktionssysteme ohne solche Schleifen:

Definition 3.4 *Sei $(M, \to)$ ein Reduktionssystem.*

1. *$\to$ heißt **terminierend** oder **Noethersch**, falls keine unendlichen Ketten*

   $$x_1 \to x_2 \to x_3 \to \dots$$

   existieren.

2. *$\to$ heißt **konvergent** (eindeutig terminierend), falls $\to$ konfluent und terminierend ist.*

Für terminierende Reduktionssysteme gilt der folgende wichtige Satz:

Satz 3.2 (Newman-Lemma) *Sei $(M, \to)$ terminierend. Dann gilt:*

$$\to \text{ ist konfluent} \iff \to \text{ ist lokal konfluent}$$

Somit gilt: Für terminierende Reduktionssysteme ist Konfluenz „einfach“ prüfbar:

> Betrachte alle 1-Schritt-Divergenzen (evtl. sind dies nur endlich viele) und berechne die Normalformen der Elemente (diese existieren wegen Terminierung). Falls diese jeweils identisch sind, ist das Reduktionssystem (lokal) konfluent.

Leider sind i.allg. Reduktionen in funktionalen Sprachen nicht immer terminierend (z.B. bei Verwendung unendlicher Datenstrukturen). Aus diesem Grund werden wir später noch andere Kriterien zur Konfluenz kennenlernen.