Dennoch ist die Klasse der linksnormalen TES sehr eingeschränkt. Daher betrachten wir eine weitere wichtige Klasse von Termersetzungssystemen, für die auch gute Auswertungsstrategien existieren:

Definition 3.12 *Sei R ein TES bzgl. einer Signatur $\Sigma = (S, F)$.*

- *$f : s_1, \ldots, s_n \to s \in F$ heißt **definierte Funktion**, falls eine Regel*

  $$f(t_1, \ldots, t_n) \to r$$

  existiert. Die Menge aller definierten Funktionen ist dann wie folgt definiert:

  $$D = \{f \mid f \text{ definierte Funktion}\} \subseteq F$$

- *$C = F \backslash D$ heißt Menge der **Konstruktoren**.*
- *$f(t_1, \ldots, t_n)$ $(n \geq 0)$ heißt **Muster** (**pattern**), falls $f \in D$ und $t_1, \ldots, t_n$ sind **Konstruktorterme**, d.h. sie enthalten keine definierte Funktionen.*
- *R heißt **konstruktorbasiert**, falls für alle Regeln $l \to r \in R$ die linke Seite l ein Muster ist.*

Beispiel:

$$\begin{array}{lrcl} R: & 0 + n & \to & n \\ & s(m) + n & \to & s(m + n) \end{array}$$

Hier ist $D = \{+\}$, $C = \{0, s\}$ und R ist konstruktorbasiert.

Die Gruppenaxiome aus Beispiel 3.1 sind dagegen nicht konstruktorbasiert.

Intuition dieser Aufteilung in Konstruktoren und definierte Funktionen:

- Konstruktoren bauen Datenstrukturen auf.
- Definierte Funktionen rechnen auf Datenstrukturen.
- Konstrukturbasierte TES entsprechen funktionalen Programmen, jedoch ist hier keine Reihenfolge der Regeln festgelegt (dies entspricht also eher einer gleichungsorientierten Interpretation der Regeln).

Induktiv-sequenzielle Termersetzungssysteme [Antoy 92] haben die Eigenschaft, dass Funktionen induktiv über den Datenstrukturen definiert sind. Dies ist zum Beispiel bei der obigen Additionsoperation "+" der Fall, die durch eine Fallunterscheidung (Induktion) über das erste Argument definiert ist.

Die genaue formale Definition ist wie folgt.

Definition 3.13 (Definierender Baum) *Ein **definierender Baum (definitional tree)** ist ein Baum, bei dem jeder Knoten mit einem Muster markiert ist. Dabei gibt es zwei Arten von definierenden Bäumen mit einem Muster π:*

- Regelknoten *der Form* $l \to r$ *mit* $\pi = l$
- Verzweigungsknoten *der Form* $branch(\pi, p, \mathcal{T}_1, \ldots, \mathcal{T}_k)$*, wobei gilt:*
  - $p \in Pos(\pi)$ *mit* $\pi|_p \in V$
  - $\mathcal{T}_i$ $(i = 1, \ldots, k)$ *ist ein definierender Baum mit dem Muster* $\pi[C_i(x_1, \ldots, x_{m_i})]_p$*, wobei* $x_1, \ldots, x_{m_i}$ *neue Variablen und* $C_1, \ldots, C_k$ *sind verschiedene Konstruktoren sind.*

Wir bezeichnen mit $pattern(\mathcal{T})$ *das Muster des definierenden Baumes* $\mathcal{T}$*.*

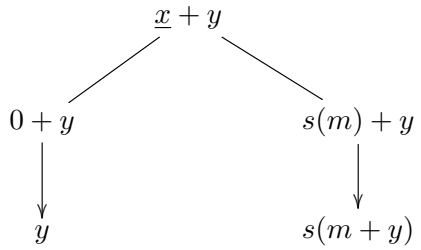
Definition 3.14 (Induktiv-sequenziell) *Sei* R *ein Termersetzungssystem.*

- $\mathcal{T}$ *heißt definierender Baum für die Funktion* f*, falls* $\mathcal{T}$ *endlich ist und das Muster* $f(x_1, \ldots, x_n)$ *hat* ($x_1, \ldots, x_n$ *sind verschiedene Variablen), jeder Regelknoten ist eine Variante einer Regel aus* R*, und jede Regel* $f(t_1, \ldots, t_n) \to r \in R$ *kommt in* $\mathcal{T}$ *genau einmal vor. In diesem Fall heißt* f ***induktiv-sequenziell****.*
- R *heißt* ***induktiv-sequenziell****, falls alle* $f \in D$ *induktiv-sequenziell sind.*

Beispiel: Betrachten wir noch einmal die Additionsfunktion:

$$\begin{array}{lrcl} R: & 0 + n & \to & n \\ & s(m) + n & \to & s(m + n) \end{array}$$

Der definierende Baum sieht wie folgt aus (in graphischer Darstellung):



Dagegen ist das parallele Oder

```
true   ∨  x      →  true
x      ∨  true   →  true
false  ∨  false  →  false
```

nicht induktiv-sequenziell, da hier kein eindeutiges Argument für eine Fallunterscheidung existiert.

Satz 3.8 *Jedes induktiv-sequenzielle TES ist orthogonal und konstruktorbasiert (aber nicht umgekehrt!)*

Für induktiv-sequenzielle TES existiert eine einfache sequenzielle Reduktionsstrategie, die Konstruktornormalformen berechnet:

Definition 3.15 *Die Reduktionsstrategie φ sei wie folgt definiert:*
Sei t ein Term, o die Position des linkesten äußersten definierten Funktionssymbols in t (d.h. $t|_o = f(t_1, \ldots, t_n)$ mit $f \in D$) und $\mathcal{T}_f$ ein definierender Baum für f. Dann ist:

$$\varphi(t) = \{o \cdot \varphi(t|_o, \mathcal{T}_f)\}$$

Hier ist zu beachten, dass $\varphi(t|_o, \mathcal{T}_f)$ eventuell undefiniert sein kann; in diesem Fall ist auch $\varphi(t)$ undefiniert. Weiterhin ist

$$\varphi(t, l \to r) = \epsilon \quad \textit{(d.h. wende Regel an)}$$

$$\varphi(t, branch(\pi, p, \mathcal{T}_1, \ldots, \mathcal{T}_k)) =$$

$$\begin{cases} \varphi(t, \mathcal{T}_j) & \textit{falls } t|_p = C_j(s_1, \ldots, s_{m_j}) \textit{ und } pattern(\mathcal{T}_j)|_p = C_j(x_1, \ldots, x_{m_j}) \\ p \cdot \varphi(t|_p, \mathcal{T}_g) & \textit{falls } t|_p = g(\ldots) \textit{ mit } g \in D \textit{ und } \mathcal{T}_g \textit{ definierender Baum für } g \end{cases}$$

Intuitiv: Die Auswertung einer Funktion erfolgt durch Analyse des zugehörigen definierenden Baumes:

- Bei Regelknoten: Wende die Regel an
- Bei Verzweigungsknoten: Betrachte den aktuellen Wert an der Verzweigungsposition:
  - Konstruktor $\rightsquigarrow$ betrachte entsprechenden Teilbaum
  - Funktion $\rightsquigarrow$ Werte Funktion aus

Beispiel:
Wir betrachten wieder die Regeln für "+" (s.o.), den definierenden Baum $\mathcal{T}_+$ für + (wie oben dargestellt und den Term

$$t = (s(0) + 0) + 0$$

Dann wird der erste Auswertungsschritt für t wie folgt berechnet:

$$\begin{aligned} \varphi(t) &= \epsilon \cdot \varphi(t, \mathcal{T}_+) \\ &= 1 \cdot \varphi(s(0) + 0, \mathcal{T}_+) \\ &= 1 \cdot \varphi(s(0) + 0, s(m) + y \to s(m + y)) \\ &= 1 \cdot \epsilon = 1 \end{aligned}$$

Insgesamt ergibt sich folgende Auswertung bzgl. der Strategie φ:

$$\begin{array}{lll} t & \to_R^\varphi & s(0+0)+0 \\ & \to_R^\varphi & s((0+0)+0) \quad \text{da } \varphi(s(0+0)+0) = \epsilon \\ & \to_R^\varphi & s(0+0) \quad\quad\quad\quad \text{da } \varphi(s((0+0)+0) = 1 \cdot 1 \\ & \to_R^\varphi & s(0) \quad\quad\quad\quad\quad \text{da } \varphi(s(0+0)) = 1 \end{array}$$

Die Strategie φ berechnet evtl. nicht die Normalform eines Terms. Dies kann zwei Ursachen haben:

1. Der Term enthält Variablen:

   $(x + 0) + (0 + 0)$

   φ ist hierfür undefiniert, allerdings ist die Normalform $(x + 0) + 0$.

2. Funktionen sind partiell, d.h. unvollständig, definiert:
   Betrachten wir hierzu die zusätzliche Regel:

   $f(s(m), n) \to 0$

   Dann ist φ undefiniert auf $f(0, 0 + 0)$, allerdings ist die Normalform $f(0, 0)$.

Definition 3.16 *Eine Funktion f heißt **vollständig definiert**, falls ein definierender Baum für f existiert, wobei in jedem Verzweigungsknoten jeder Konstruktor (der entsprechenden Sorte) in einem Teilbaum vorkommt.*

Beispiel: Die Konstruktoren von Nat sind $0, s$. Daher ist + vollständig definiert, die obige Funktion f jedoch nicht.

Für vollständig definierte Funktionen gilt:

Satz 3.9 *Ist R ein induktiv-sequenzielles Termersetzungssystem und sind alle Funktionen vollständig definiert, dann ist φ normalisierend auf Grundtermen.*

Diese Eigenschaft ist ausreichend für das Rechnen in funktionalen Sprachen:

- Wir wollen nur Grundterme ausrechnen.
- Falls Normalformen noch definierte Funktionen enthalten, wird dies nicht als Wert angesehen, sondern üblicherweise eine Fehlermeldung ausgegeben.

Vergleichen wir einmal das Pattern matching (Kapitel 2.3) mit der Strategie φ:

Pattern matching:

- abhängig von Reihenfolge der Regeln
- im Allg. nicht normalisierend
- normalisierend auf uniformen Programmen
- erlaubt überlappende linke Regelseiten (dann ist das Ergebnis allerdings von der Regelreihenfolge abhängig!)

Strategie φ:

- unabhängig von der Regelreihenfolge
- normalisierend

- erlaubt keine Überlappungen

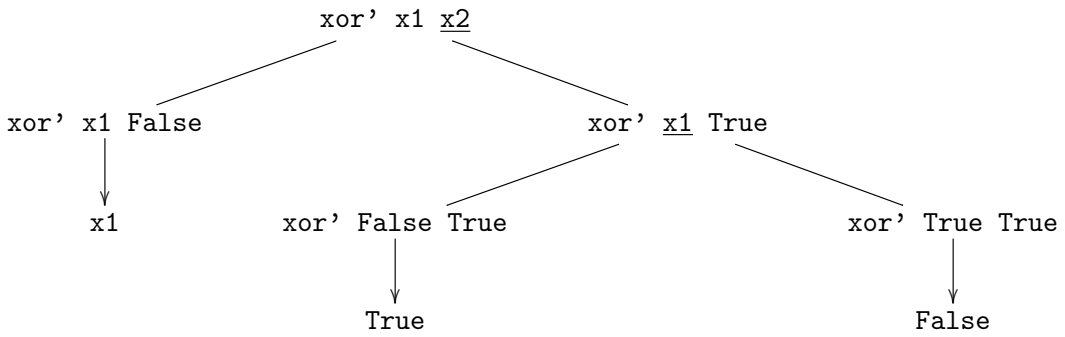
Satz 3.10 *Jede uniforme Funktionsdefinition ist induktiv-sequenziell.*

Die Umkehrung gilt allerdings nicht:

Beispiel: (vgl. Kap. 2.3)

```
xor'  x      False = x
xor'  False  True  = True
xor'  True   True  = False
```

Diese Funktion ist nicht uniform, aber induktiv-sequenziell (durch Verzweigung über das zweite Argument):



Hieraus ist ersichtlich, dass φ berechnungsstärker ist als einfaches Links-rechts Pattern Matching.

Weiterer Aspekt der Strategie φ: Durch eine einfache Erweiterung von definierenden Bäumen auf überlappende Regeln kann man φ zu einer Strategie mit paralleler Auswertung erweitern ([Antoy 92]).