1 Primitv rekursive Funktionen

Die primitiv rekursiven Funktionen sind definiert als die kleinste Menge von Funktionen von $\mathbb{N}^m$ nach $\mathbb{N}$, für ein beliebiges $m \in \mathbb{N}$, das im Folgenden Stelligkeit genannt wird, die für alle $i, n \in \mathbb{N}$, die folgenden Grundfunktionen enthalten:

$$\begin{aligned} \texttt{Z n} &\colon \mathbb{N}^n \to \mathbb{N} \\ \texttt{Z n}(x_1, \ldots, x_n) &= 0 \end{aligned}$$

$$\begin{aligned} \texttt{S} &\colon \mathbb{N} \to \mathbb{N} \\ \texttt{S}(x) &= x + 1 \end{aligned}$$

$$\begin{aligned} \texttt{Pi n i} &\colon \mathbb{N}^n \to \mathbb{N} \\ \texttt{Pi n i}(x_1, \ldots, x_i, \ldots, x_n) &= x_i \end{aligned}$$

und unter den folgenden Operationen abgeschlossen sind:
Sei h eine primitiv rekursive Funktion mit Stelligkeit m und seien $g_1 \ldots g_m$ primitiv rekursive Funktionen mit Stelligkeit n. Dann ist $f := \texttt{C}(\texttt{h}, (\texttt{g}_1, \texttt{g}_2, \ldots, \texttt{g}_\texttt{m}))$

$$\begin{aligned} f &\colon \mathbb{N}^n \to \mathbb{N} \\ f(x_1, \ldots, x_n) &= h(g_1(x_1, \ldots, x_n), g_2(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)) \end{aligned}$$

eine primitiv-rekursive Funktion.

Sei g ein n-stellige und h eine $n+2$-stellige primitiv rekursive Funktion. Dann ist $f := P(g, h)$

$$\begin{aligned} f &\colon \mathbb{N}^{n+1} \to \mathbb{N} \\ f(0, x_1, \ldots, x_n) &= g(x_1, \ldots, x_n) \\ f(y+1, x_1, \ldots, x_n) &= h(f(y, x_1, \ldots, x_n), y, x_1, \ldots, x_n) \end{aligned}$$

eine primtiv rekursive Funktion.

1.1 Beispiel

$$\begin{aligned} plus &: \mathbb{N}^2 \to \mathbb{N} \\ plus(0, x_1) &= x_1 = \texttt{Pi 1 1}\ (x_1) \\ plus(y+1, x_1) &= plus(y, x_1) + 1 = \texttt{C}(\texttt{S}, (\texttt{Pi 3 1}))\ (plus(y, x_1), y, x_1) \end{aligned}$$

Es gilt also: $plus = \texttt{P}\ (\texttt{Pi 1 1}, \texttt{C}\ (\texttt{S}, (\texttt{Pi 3 1})))$

Ein entsprechender Funktionsaufruf dieser Funktion sieht im Interprer folgendermaßen aus:

```
P (Pi 1 1, C (S, (Pi 3 1))) (3,5)
```

Da zusätzlich let-bindings erlaubt sind, lässt sich der Code auch übersichtlicher schreiben:

```
let idOnN = Pi 1 1
let sucOfFirst = C (S, (Pi 3 1))
let plus = P (idOnN, sucOfFirst)
plus (3,5)
```

2 Definition der Sprache

`nat`	::=	`[0-9]+`	*Natural Numbers*
`id`	::=	`[a-zA-Z][a-zA-Z0-9]*`	*Identifier*
`prek`	::=	`'Z' num`	*Arity*
	|	`'Pi' num num`	*Arity and Index*
	|	`'Suc'`	*Suc x = x + 1*
	|	`'P' '(' prek' ',' prek' ')'`	*Base Case and Inductive Case*
	|	`'C' prek' prektuple`	
`prek'`	::=	`prek`	
	|	`id`	
`prekTuple`	::=	`()`	
	|	`'(' prek' (',' prek')* ')'`	
`natTuple`	::=	`()`	
	|	`'(' nat (','nat)* ')'`	
`Assignment`	::=	`'let' id '=' prek`	
`FunCall`	::=	`id natTuple`	
	|	`prek natTuple`	
`Programm`	::=	`Assignment* FunCall*`	

Literatur

[1] Benjamin Hodgson. *Is it possible to ”bake dimension into a typeïn haskell?* `https://softwareengineering.stackexchange.com/questions/276867/is-it-possible-to-bake-dimension-into-a-type-in-haskell`. 2015.

[2] A Martin-Pizarro. *Logik für Studierende der Informatik Kurzskript*. 2019.

[3] tommyengstrom. *Codemirror-Elm*. `https://github.com/tommyengstrom/codemirror-elm`. 2013.

[4] Wikipedia. *Primitive recursive function — Wikipedia, The Free Encyclopedia*. `http://en.wikipedia.org/w/index.php?title=Primitive\%20recursive\%20function&oldid=1042441754`. [Online; accessed 27-October-2021]. 2021.