1 Rekursion

1.1 Binomialkoeffizient

Zuerst definieren wir eine μ-rekursive Fakultätsfunktion $fak : \mathbb{N} \to \mathbb{N}$ mit:

$$\begin{aligned} fak(0) =& c_1^0 \\ fak(n+1) =& mult \circ (succ(\Pi_1^2), \Pi_2^2)(n, fak(n)) \end{aligned}$$

Da die zur Definition verwendete $mult$-Funktion als μ-rekursiv angenommen werden kann, und alle weiteren Bestandteile Inhalt der Klasse der primitiv rekursiven Funktionen sind, ist die Funktion fak somit auch μ-rekursiv.

Mithilfe dieser Fakultätsfunktion können wir nun die μ-rekursive Funktion $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ mit:

$$g(n,k) = div \circ (fak \circ \Pi_1^2, mult \circ (fak \circ \Pi_2^2, fak \circ sub(\Pi_1^2, \Pi_2^2)))(n,k)$$

definieren.

Da wir vorher gezeigt haben, dass die Funktion fak μ-rekursiv ist, und jede Funktion, die durch Einsetzung/Komposition von μ-rekursiven Funktionen entsteht, auch μ-rekursiv ist, ist die Funktion g selbst μ-rekursiv.

1.2

Die geschlossene Formel für $f(1,x,y)$ für $y \leq x$ lautet:

$$2^x * \frac{x!}{y! * (x-y)!}$$

Beweis:

Wir führen Induktion über x:

Induktionsanfang: Für $x = 0$ gilt $y = 0$, da $y \leq x$ und

$$f(1,0,0) = 1 = 2^0 * \frac{0!}{0! * (0-0)!}$$

Induktionsvoraussetzung:

$$f(1,x,y) = 2^x * \frac{x!}{y! * (x-y)!}$$

Induktionsbehauptung:

$$f(1,x+1,y) = 2^{x+1} * \frac{(x+1)!}{y! * ((x+1)-y)!}$$

Induktionsschluss:

$$\begin{aligned}
f(1,x+1,y) &= f(0,f(1,x,y-1),f(1,x,y)) \\
&= 2*f(1,x,y-1)+2*f(1,x,y) \\
&= 2*2^x*\frac{x!}{(y-1)!*(x-(y-1))!}+2*2^x*\frac{x!}{y!*(x-y)!} \\
&= 2^{x+1}*(\frac{x!}{(y-1)!*(x-(y-1))!}+\frac{x!}{y!*(x-y)!}) \\
&= 2^{x+1}*(\binom{x}{y-1}+\binom{x}{y}) \\
&= 2^{x+1}*(\binom{x+1}{y}) \\
&= 2^{x+1}*\frac{(x+1)!}{y!*((x+1)-y)!}
\end{aligned}$$

Wir führen Induktion über y:

Induktionsanfang: Für y = 0 gilt:

$$\begin{aligned}
f(1,x,0) &= f(0,0,f(1,x-1,0)) \\
&= 2*f(1,x-1,0) \\
&= ... \\
&= 2^x*f(1,0,0) \\
&= 2^x*1 \\
&= 2^x*\frac{x!}{x!} \\
&= 2^x*\frac{x!}{0!*(x-0)!}
\end{aligned}$$

Induktionsvoraussetzung:

$$f(1,x,y)=2^x*\frac{x!}{y!*(x-y)!}$$

Induktionsbehauptung:

$$f(1,x,y+1)=2^x*\frac{(x)!}{(y+1)!*((x)-(y+1)!}$$

Induktionsschluss:

$$\begin{aligned}
f(1,x,y+1) &= f(0,f(1,x-1,y),f(1,x-1,y+1)) \\
&= 2*f(1,x-1,y)+2*f(1,x-1,y+1) \\
&= 2*2^{x-1}*\frac{(x-1)!}{y!*((x-1)-y)!}+2*2^{x-1}*\frac{(x-1)!}{(y+1)!*((x-1)-(y+1))!} \\
&= 2^x*(\frac{(x-1)!}{y!*((x-1)-y)!}+\frac{(x-1)!}{(y+1)!*((x-1)-(y+1))!}) \\
&= 2^x*(\binom{x-1}{y}+\binom{x-1}{y+1}) \\
&= 2^x*(\binom{x}{y+1}) \\
&= 2^x*\frac{(x)!}{(y+1)!*((x)-(y+1)!}
\end{aligned}$$

Aus den beiden durchgeführten Induktionsbeweisen folgt, dass die Formel

$$2^x*\frac{x!}{y!*(x-y)!}$$

für $f(1,x,y)$ für $y \leq x$ gilt.

2 Simulation

Simulation von WHILE durch LOOP*

Hinweis Zeigen Sie zunächst, dass LOOP-Programme mit dem zusätzlichen Konstrukt Turing-mächtig sind.

LOOP-Programme sind mithilfe des zusätzlichen Konstruktes *DO P WHILE* $x_i \neq 0$ *END* Turing-mächtig, da jedes WHILE-Programm *WHILE* $x_i \neq 0$ *DO P END* durch das folgende LOOP-Programm (mit Konstrukt) simuliert werden kann:

IF $x_i \neq 0$ THEN
  DO P WHILE $x_i \neq 0$ END;
END

Im Folgenden wird gezeigt, dass LOOP*-Programme Turing-mächtig sind, indem bewiesen wird, wie jedes WHILE-Programm durch ein LOOP*-Programm simuliert werden kann.

Definition Hilfskonstrukt Zunächst soll ein Hilfskonstrukt eingeführt werden, welches durch ein LOOP-Programm simuliert werden kann.

IF $x_i \neq 0$ THEN P END $\triangleq x_i := 0$; LOOP x_i DO $x_j := 1$ END; LOOP x_j DO P END;

Beweis

Ein jedes WHILE-Programm *WHILE* $x_i \neq 0$ *DO P END* kann durch das folgende LOOP*-Programm simuliert werden:

IF $x_i \neq 0$ THEN
  $x_{N_P} := x_0$;
  MAKE P BREAK
    IF $x_i \neq 0$ THEN
      $x_0 := 0$;
    END;
  END;
  $x_0 := x_{N_P}$
END;

Durch die IF-Anweisung in der ersten Zeile wird sichergestellt, dass der *MAKE*-Block nur dann ausgeführt wird, wenn $x_i \neq 0$ ist. Das ist wichtig, da bei WHILE-Programmen die Bedingung auch zuerst überprüft wird, bevor der Block das erste Mal ausgeführt wird.

Die Zuweisung des Wertes von x_0 an x_{N_P} vor dem *MAKE*-Block und wieder zurück nach Ausführung des *MAKE*-Blocks stellt sicher, dass der zuvor in x_0 gespeicherte Wert nicht verloren geht. Da zum Speichern des Wertes von x_0 die Variable x_{N_P} verwendet wird, ist sichergestellt, dass keine in P benutzte Variable überschrieben wird.

3 Unentscheidbarkeit

Sei

$$P := \{m_1\#m_2\#w \mid \exists M_1 : m_1 = \langle M_1\rangle, \exists M_2 : m_2 = \langle M_2\rangle \text{ und}$$
$$M_1 \text{ und } M_2 \text{ verhalten sich unterschiedlich auf Eingabewort w}\}$$

die Menge aller Wörter, welche die Frage „Verhalten sich M und M' bei Eingabewort w unterschiedlich?“ mit „Ja“ beantworten.

Wir zeigen, dass P nicht entscheidbar ist, in dem wir zeigen, dass

$$K = \{w \in \{0,1\}^* \mid M_w \text{ hält auf Eingabe } w\}$$

auf P reduzierbar ist.

Zu zeigen ist $K \leq P$.

Sei $f : \{0,1\}^* \to \{0,1,\#\}^*$ mit: $f(w) = w\#w\#w$

f ist total, da nach Definition M_w für jedes w eine Turing-Maschine M_w existiert. f ist berechenbar, da nach der Vorlesung M_w berechenbar ist, eine universelle Turing-Maschine und eine Turing-Maschine die das Band löscht existieren und die Kodierung einer Turing-Maschine berechenbar ist. Wenn des weiteren

$$... \Leftrightarrow ...$$

gilt, ist die Reduktion gezeigt.

Wenn also K entscheidbar wäre, dann musste auch P entscheidbar sein. Da dies nicht der Fall ist, ist auch P nicht entscheidbar.

4 Ackermann-Funktion