1 Rachunek λ

Niech V będzie przeliczalnie nieskończonym zbiorem zmiennych przedmiotowych $x,\ y,\ \ldots$ (indeksowanych być może liczbami naturalnymi). Elementy takiego zbioru będziemy nazywali λ-*zmiennymi.* Ponieważ V jest potencjalnie nieskończony, zastrzegamy sobie możliwość wybierania w razie potrzeby wcześniej nie użytej zmiennej.

Definicja 1. (Zbiór $\tilde{\mathbf{\Lambda}}$ pretermów) Zbiorem pretermów będziemy nazywali najmniejszy (w sensie mnogościowym) zbiór wyrażeń $\tilde{\mathbf{\Lambda}}$ taki, że:

(P1) Jeśli $x \in V$, to $x \in \tilde{\mathbf{\Lambda}}$.

(P2) Jeśli $M,\ N \in \tilde{\mathbf{\Lambda}}$, to $(M\,N) \in \tilde{\mathbf{\Lambda}}$.

(P3) Jeśli $x \in V$ i $M \in \tilde{\mathbf{\Lambda}}$, to $(\lambda x.\, M) \in \tilde{\mathbf{\Lambda}}$.

Definicję 1 można równoznacznie wyrazić przy pomocy notacji Backusa-Naura. Wówczas ma ona następującą, zwięzłą postać:

$$\tilde{\mathbf{\Lambda}} \leftarrow V \mid (\tilde{\mathbf{\Lambda}}\,\tilde{\mathbf{\Lambda}}) \mid (\lambda V.\, \tilde{\mathbf{\Lambda}})$$

Powiemy, że dwa λ-termy są *syntaktycznie równe*, jeśli rozumiane jako ciągi znaków są identyczne. Równość syntaktyczną będziemy oznaczali znakiem $\equiv$.

Elementy $\tilde{\mathbf{\Lambda}}$ będziemy oznaczali literami $L,\ M,\ N,\ P,\ Q,\ R$ i ich wariantami z górnymi lub dolnymi indeksami. Wyrażenia postaci (P2) nazywamy *aplikacjami* M do N. Symbol λ występujący w (P3) nazywamy λ-*abstraktorem*, zaś wyrażenia powstałe przez zastosowanie tej reguły to λ-*abstrakcje.* W wyrażeniu postaci $(\lambda x.\, M)$ preterm M jest w *zasięgu* λ-abstraktora, a zmienna x jest przez niego *związana.* Ponadto, będziemy stosowali następujące konwencje notacyjne:

- *najbardziej zewnętrzne nawiasy bedą pomijane,*
- *aplikacja wiąże lewostronnie; wyrażenia postaci* $(PQ)R$ *będą zapisywane w postaci* PQR*,*
- λ-*abstrakcja wiaże prawostronnie:* $\lambda x_1.\,(\lambda x_2.\,P)$ *zapisujemy* $\lambda x_1.\,\lambda x_2.\,P$*,*
- *następujące po sobie* λ-*abstrakcje postaci* $\lambda x_1.\,\lambda x_2.\,\ldots\lambda x_n.\,P$ *zapisujemy pod wspólnym* λ-*abstraktorem:* $\lambda x_1\,x_2\,\ldots\,x_n.\,P$.
- n-*krotną aplikację* $P \in \tilde{\mathbf{\Lambda}}$ *do siebie zapisujemy skrótowo:* $P^n \equiv \underbrace{P\,P\ \ldots\ P}_{n\text{-razy}}$

Przykład 1. Podajmy kilka przykładów λ-pretermów pogrupowanych ze względu na ich konstrukcję.

(P1): $x,\ y,\ z.$

(P2): $x\,x,\ y\,x,\ x\,(x\,z),$
$(\lambda x.\,(xz))y,\ y\,(\lambda x.\,(x\,z)),\ (\lambda x.\,x)\,(\lambda x.\,x).$

(P3): $\lambda x.(x\,z),\ \lambda yz.\,x,\ \lambda x.(\lambda x.(x\,x))$.

Podwyrażenia λ-pretermu mogą być wzajemnie identyczne i występować wielokrotnie. Obserwację tę ujmuje następująca definicja.

Definicja 2. (Multizbiór Sub podtermów pretermu)

(1) $\mathrm{Sub}(x) = \{x\}$

(2) $\mathrm{Sub}(MN) = \mathrm{Sub}(M) \cup \mathrm{Sub}(N) \cup \{M\,N\}$

(3) $\mathrm{Sub}(\lambda x.\,M) = \mathrm{Sub}(M) \cup \{\lambda x.\,M\}$

Elementy multizbioru $\mathrm{Sub}(M)$ nazywamy *podtermami* M. Jeśli L jest podtermem M, ale $L \not\equiv M$, to L nazywamy podtermem *właściwym.*

Przykład 2. Podtermy wybranych λ-pretermów.

$$\begin{aligned}
&\text{(a)}\quad \mathrm{Sub}\,(\lambda x.\,x\,x) = \left\{(\lambda x.\,x\,x)^1,\ (x\,x)^1,\ x^2\right\}\\
&\text{(b)}\quad \mathrm{Sub}\,((\lambda x.\,x\,x)\ (\lambda x.\,x\,x)) =\\
&\qquad = \left\{((\lambda x.\,x\,x)\,(\lambda x.\,x\,x))^1,\ (\lambda x.\,x\,x)^2,\ (x\,x)^2,\ x^4\right\}
\end{aligned}$$

W powyższych przykładach użyliśmy standardowej notacji w górnym indeksie umieszczając krotność występowania elementu.

Definicja 3. (Zbiór FV zmiennych wolnych) Dla dowolnego pretermu M określamy zbiór $\mathrm{FV}(M)$ *zmiennych wolnych* w M w następujący sposób:

$$\begin{aligned}
\mathrm{FV}(x) &= \{x\}\\
\mathrm{FV}(\lambda x.\,P) &= \mathrm{FV}(P) \setminus \{x\}\\
\mathrm{FV}(PQ) &= \mathrm{FV}(P) \cup \mathrm{FV}(Q)
\end{aligned}$$

Jesli $\mathrm{FV}(M) = \varnothing$, to mówimy, że M jest *domknięty* lub nazywamy M *kombinatorem.*

Przykład 3. (a) $\mathrm{FV}(\lambda x.\,x\,y) = \{y\}$

(b) $\mathrm{FV}(x\,(\lambda x.\,x\,y)) = \{x,\ y\}$

(c) $\mathrm{FV}(\lambda xyz.\,xy) = \varnothing$

Definicja 4. (Podstawienie) Dla dowolnych M, N $\in \tilde{\boldsymbol{\Lambda}}$ i $x \in V$ przez $N[x/N]$ oznaczamy rezultat podstawienia termu N za wszystkie wolne wystąpienia zmiennej x w M, o ile w rezultacie podstawienia nie zostaną związane żadne zmienne wolne występujące w N. W takim wypadku:

(S1) $x[x/N] = N$

(S2) $y[x/N] = y$, o ile $x \not\equiv y$

(S3) $(PQ)[x/N] = P[x/N]\,Q[x/N]$

(S4) $(\lambda y.\,P)[x/N] = \lambda y.\,P[x/N]$, gdzie $x \not\equiv y$ i $y \notin \mathrm{FV}(N)$

(S5) $(\lambda x.\,P)[x/N] = \lambda x.\,P$

Lemat 1. *(O podstawieniu) Niech $M, N, L \in \tilde{\mathbf{\Lambda}}$ i niech ponadto $x \not\equiv y$ oraz $x \notin \mathrm{FV}(L)$. Wówczas*

$$M[x/N][y/L] \equiv M[y/L]\,[x/N\,[y/L]]\,. \tag{1}$$

Dowód. Dowód przebiega przez indukcję strukturalną względem M. Rozważmy następujące przypadki:

i) M jest zmienną. Wówczas:

  a. Jeśli $M \equiv x$, to obie strony (1) po podstawieniu są postaci $N[y/L]$.

  b. Jeśli $M \equiv y$, to ponieważ $x \not\equiv y$ i $x \notin \mathrm{FV}(M)$, po wykonaniu podstawienia po lewej stronie (1) otrzymujemy $M[x/N][y/L] \equiv L$. Ponieważ $x \notin \mathrm{FV}(L)$, to po wykonaniu podstawienia po prawej stronie widzimy, że obydwie strony są identyczne.

  c. Jeśli $M \equiv z$ i $z \not\equiv x$ oraz $z \not\equiv y$, to obydwie strony (1) sa identyczne.

ii) $M \equiv PQ$ dla pewnych $P, Q \in \tilde{\mathbf{\Lambda}}$. Wówczas korzystając z hipotezy indukcyjnej wnosimy, że

$$\begin{aligned} P[x/N][y/L] &\equiv P[y/L]\,[x/N\,[y/L]]\,, \\ Q[x/N][y/L] &\equiv Q[y/L]\,[x/N\,[y/L]]\,. \end{aligned}$$

Mając na względzie (S3) widzimy, że twierdzenie zachodzi i w tym przypadku.

iii) Jeśli $M \equiv \lambda z.\,P$ oraz $z \equiv x$ lub $z \equiv y$, to z (S'5) widzimy, że obydwie strony (1) sa identyczne. Przypuśćmy, że $z \not\equiv x$ i $z \not\equiv y$ i $z \notin \mathrm{FV}(L)$. Wówczas na podstawie hipotezy indukcyjnej mamy:

$$\begin{aligned} (\lambda z.\,P)[x/N][y/L] &= \lambda z.\,P[x/N][y/L] = \\ &= \lambda z.\,P[y/L][x/N[y/L]] = \\ &= (\lambda z.\,P)[y/L][x/N[y/L]]. \end{aligned}$$

□

Wniosek 1. *Jesli $M[x/y]$ jest określone i $y \notin \mathrm{FV}(M)$, to $M[x/y][y/x]$ jest określone oraz $M[x/y][y/x] = M$.*

Dowód. Mając na uwadze Lemat 4 dowód przebiega przez indukcję strukturalną względem M. □

1.1 Wyrażenia λ

Na ogół chcielibyśmy utożsamiać pretermy, które różnią się wyłącznie zmiennymi związanymi, tak jak w przypadku wyrażeń $\lambda x.\, zx$ i $\lambda y. zy$. W takim wypadku powiemy o nich, że są swoimi *α-wariantami* lub że są ze sobą w relacji *α-konwersji.*

Definicja 5. (Relacja α-konwersji) Relacją $=_\alpha$ (α-konwersji) nazywamy najmniejszy w sensie mnogościowym praporządek na $\tilde{\mathbf{\Lambda}}$ taki, że

(α1) Jeśli $y \notin \mathrm{FV}(M)$ oraz $M[x/y]$ jest określone, to $\lambda x.\, M \;=_\alpha\; \lambda y.\, M[x/y]$

(α2) Jeśli $M =_\alpha N$, to dla dowolnego $x \in V$ zachodzi $\lambda x.\, M =_\alpha \lambda x.\, N$

(α3) Jeśli $M =_\alpha N$, to dla dowolnego $Z \in \tilde{\mathbf{\Lambda}}$ zachodzi $MZ =_\alpha NZ$

(α4) Jeśli $M =_\alpha N$, to dla dowolnego $Z \in \tilde{\mathbf{\Lambda}}$ zachodzi $ZM =_\alpha ZN$

Przykład 4.

$$
\begin{aligned}
\lambda xy.\, x(xy) &\equiv \lambda x.\, (\lambda y.\, x(xy)) \\
&\equiv_\alpha \lambda x.\, (\lambda z.\, x(xz)) \\
&\equiv_\alpha \lambda v.\, (\lambda z.\, v(vz)) \\
&\equiv \lambda vz.\, v(vz).
\end{aligned}
$$

Wniosek 2. *Relacja $=_\alpha$ jest relacją równoważności.*

Dowód. Wystarczy, że pokażemy, że relacja $=_\alpha$ jest symetryczna. Dowód przebiega przez indukcję względem Definicji 5. Rozważmy następujące przypadki:

i) Jeśli $M =_\alpha N$ w konsekwencji zwrotności $=_\alpha$, to $M \equiv N$, a zatem również $N \equiv M$. Stąd $N =_\alpha M$.

ii) Jeśli $M =_\alpha N$ w konsekwencji przechodniości $=_\alpha$, to istnieje $L \in \tilde{\mathbf{\Lambda}}$ takie, że $M =_\alpha L$ i $L =_\alpha N$. Wówczas z hipotezy indukcyjnej $N =_\alpha L$ i $L =_\alpha M$. Z przechodniości relacji $=_\alpha$ otrzymujemy spodziewaną tezę.

iii) Przypuśćmy, że $M =_\alpha N$ w konsekwencji (α1) dla $M \equiv \lambda x.\, M'$ i $N \equiv \lambda y.\, M'[x/y]$. Ponieważ $x \notin \mathrm{FV}(M'[x/y])$, to ze względu na Wniosek 1 mamy, że $M'[x/y][y/x] = M'$. Zatem, na podstawie (α1):

$$\lambda y.\, M'[x/y] =_\alpha \lambda x.\, M'[x/y][y/x].$$

iv) Jeśli $M =_\alpha N$ w konsekwencji (α2), gdzie $M = \lambda x.\, M'$ i $N = \lambda x.\, N'$ dla $M' =_\alpha N'$, to z hipotezy indukcyjnej $N' =_\alpha M'$ i w konsekwencji (α2) mamy, że $N =_\alpha M$.

v) Jeśli $M =_\alpha N$ w konsekwencji ($\alpha 3$) dla $M \equiv M'Z$ i $N \equiv N'Z$ takich, że $M' =_\alpha N'$, to z hipotezy indukcyjnej oczywiście $N' =_\alpha M'$, a zatem z ($\alpha 3$) $N =_\alpha M$.

vi) Jeśli $M =_\alpha N$ w konsekwencji ($\alpha 3$), to postępujemy jak w przypadku (v). $\square$

Definicja 6. (Zbiór $\mathbf{\Lambda}$ λ-termów) Każdą klasę abstrakcji relacji $=_\alpha$ nazywamy λ-termem. Zbiór wszystkich λ-termów $\mathbf{\Lambda}$ to zbiór ilorazowy relacji α-konwersji:

$$\mathbf{\Lambda} = \left\{ [M]_{=_\alpha} \mid M \in \tilde{\mathbf{\Lambda}} \right\}$$

Konwencja. *Wprowadzamy następujące konwencje notacyjne:*

$$\begin{aligned} &x = [x]_{=_\alpha}, \\ &PQ = [M'N']_{=_\alpha}, \text{ gdzie } M = [M']_{=_\alpha} \text{ i } N = [N']_{=_\alpha}, \\ &\lambda x.\, M = [\lambda x.\, M']_{=_\alpha}, \text{ gdzie } N = [N']_{=_\alpha}. \end{aligned}$$

Twierdzenie 1. *Każdy $M \in \mathbf{\Lambda}$ ma jedną z poniższych postaci:*

(1) $M \equiv \lambda x_1 \dots x_n.\; y\, N_1 \dots N_m$, *gdzie* $n, m \geq 0$ *i* $y \in \mathrm{V}$

(2) $M \equiv \lambda x_1 \dots x_n.\; (\lambda y.\; N_0)\, N_1 \dots N_m$, *gdzie* $n \geq 0$ *i* $m \geq 1$

O λ-termach postaci (1) mówimy, że są w czołowej postaci normalnej *(HNF, ang.* head normal form*).*

Dowód. Z definicji λ-term M jest albo zmienną, albo aplikacją postaci PQ, albo abstrakcją postaci $(\lambda x.\, P)$. Wówczas mamy nastepujące przypadki:

i) Jeśli M jest zmienną, to wówczas M jest postaci (1).

ii) Jeśli M jest aplikacją, to wówczas $M \equiv P_0 P_1 \dots P_m$, gdzie P_0 nie jest aplikacją. Wówczas M jest postaci (1) albo postaci (2) dla $n = 0$, w zależności od tego czy P_0 jest zmienną (wówczas jest to przypadek (1)) czy abstrakcja (wówczas jest to przypadek (2)).

iii) Jeśli M jest abstrakcją, to wówczas $M \equiv \lambda x_1\, x_2 \dots x_m.\; P_0\, P_1 \dots P_n$, gdzie P_0 abstrakcją już nie jest. Wówczas P_0 jest ablo zmienną (przypadek (1)) albo aplikacją (przypadek (2)).

Na zbiór $\mathbf{\Lambda}$ przenoszą się pojęcia podtermu, zmiennych wolnych i operacji podstawienia definiowane uprzednio dla pretermów.

Definicja 7. (Multizbiór Sub podtermów λ-termu) Dla dowolnego λ-termu $M = [M']_{=_\alpha}$ okreslamy

$$\mathrm{Sub}(M) = \mathrm{Sub}(M'),$$

gdzie $\mathrm{Sub}(M')$ jest multizbiorem podwyrażeń pretermu M' zdefiniowanym w myśl Definicji 2.

Definicja 8. (Zbiór zmiennych wolnych FV) Dla dowolnego λ-termu $M = [M']_{=_\alpha}$ określamy zbiór $\mathrm{FV}(M)$ *zmiennych wolnych* w M

$$\mathrm{FV}(M) = \mathrm{FV}(M'),$$

gdzie $\mathrm{FV}(M')$ jest zbiorem zmiennych wolnych pretermu M' zdefiniowanym w myśl Definicji 3.

Definicja 9. (Podstawienie) Niech $M = [M']_{=_\alpha}$ i $N = [N']_{=_\alpha}$ i niech $M'[x/N']$ będzie określone w myśl Definicji 4. Wówczas

$$M[x/N] = [M'[x/N']]_{=_\alpha}.$$

Operacja podstawienia wymaga jednak pewnej delikatności. Rozważmy następującą relację:

$$\lambda x.\, zx =_\alpha \lambda y.\, zy$$

Zauważmy, że traktując podstawienie w sposób naiwny, mamy, że $(\lambda x.\, zx)[z/x] \neq_\alpha (\lambda y.\, zy)[z/x]$, a więc tracimy pożądaną własność niezmienniczości α-konwersji względem podstawienia. Stąd w Definicji 4 wymóg, aby podstawienie nie prowadziło do uszczuplenia zbioru zmiennych wolnych. Alternatywnym rozwiązaniem jest określenie podstawienia, które wprowadzałoby do wyrażenia nową zmienną i prowadziło w konsekwencji do abstrahowania po wcześniej nie występujacych zmiennych:

$$(\lambda x.\, M)[y/N] = \lambda x'.\, M[x/x'][y/N],$$

w przypadku, gdy $x \not\equiv y$, gdzie $x' \notin \mathrm{FV}(M)$ i $x' \notin \mathrm{FV}(N)$. Rozstrzygnięcie takie przytacza się w [HS08]. Po uwzględneniu odpowiednich modyfikacji, Definicja 4 przyjmuje następującą postać:

Definicja 4'. *(Podstawienie')*

(S'1) $x[x/N] = N$

(S'2) $y[x/N] = y$, *o ile* $x \not\equiv y$

(S'3) $(PQ)[x/N] = P[x/N]\,Q[x/N]$

(S'4) $(\lambda x.\, P)[x/N] = \lambda x.\, P$

(S'5) $(\lambda y.\, P)[x/N] = \lambda y.\, P$, *jeśli* $x \notin \mathrm{FV}(P)$

(S'6) $(\lambda y.\, P)[x/N] = \lambda y.\, P[x/N]$, *gdzie* $x \in \mathrm{FV}(P)$ *i* $y \notin \mathrm{FV}(N)$

(S'7) $(\lambda y.\, P)[x/N] = \lambda z.\, P[y/z][x/N]$, *gdzie* $x \in \mathrm{FV}(P)$ *i* $y \in \mathrm{FV}(N)$

przy czym w (S'7) wymagamy, aby zmienna z *nie występowała wcześniej w termach* N *i* P *jako zmienna wolna, zaś dla (S'5)-(S'7) dodatkowo* $y \not\equiv x$.

Uwaga 1. Każde podstawienie $[x/N]$ jest funkcją z $\mathbf{\Lambda} \to \mathbf{\Lambda}$, gdzie $x \in V$ i $N \in \mathbf{\Lambda}$ są dowolnymi parametrami. Zbiór S podstawień ma strukturę monoidu z działaniem składania

$$M([x_2/N_2] \circ [x_1/N_1]) = (M[x_1/N_1])[x_2/N_2] \equiv M[x_1/N_1][x_2/N_2]$$

dla dowolnych $[x_1/N_1]$, $[x_2/N_2] \in S$, o ile S posiada element neutralny ι taki, że

$$M\iota = M, \text{ gdzie } [x/x] = \iota \text{ dla dowolnego } x \in V.$$

W literaturze znajdujemy mnogość propozycji, które w ten czy inny sposób starają się ułatwić rzeczywistą implementację podstawienia. Na szczególną uwagę zasługują tutaj tak zwane *indeksy de Bruijna*. Zaproponowana przez N. G. de Brujina w [Bru72] notacja eliminuje bezpośrednie występowanie symboli zmiennych w λ-termach, zastępując je liczbą naturalną wyrażającą głębokość zagnieżdżenia odpowiedniej λ-abstrakcji przez którą jest związana, przykładowo:

$$\lambda f.\,(\lambda x.\,(f(xx))\lambda x.\,(f(xx))) \equiv_{deBrujin} \lambda(\lambda 2(11))\lambda 2(11)$$

Historycznie wiąże się ta notacja z jego pracami nad systemem komputerowo wspomaganego dowodzenia twierdzeń AUTOMATH. Rozwiązanie takie, podobnie jak w przypadku tzw. logik kombinatorów (np. rachunku SKI), eliminuje konieczność utożsamiania termów przez α-konwersję, ale istotnie zmniejsza ich czytelność.

Szerszy komentarz dotyczący dotychczasowych prób uchwycenia operacji podstawienia można prześledzić w [Alt02]. Nasze rozważania opierają się w tej materii przeważająco na [SU06]. Samo podejście do definiowania λ-termow przez operację α-konwersji nie jest powszechne w literaturze przedmiotu. Analogiczną konstrukcję należałoby powtarzać wprowadzając każdy kolejny system, dlatego w dalszej części tej pracy będziemy poprzestawali na nieformalnym traktowaniu wyrażeń danego systemu jako odpowiednich klas α-konwersji.

Definicja 10. (Podstawienie jednoczesne) Dla dowolnego $M \in \mathbf{\Lambda}$, ciągu λ-zmiennych $\vec{x}$ i ciągu λ-termów $\vec{N}$ określamy:

($\vec{s}1$) $x_i[\vec{x}/\vec{N}] = N_i$ dla $i \in \mathbb{N}$.

($\vec{s}2$) $y[\vec{x}/\vec{N}] = y$ o ile dla dowolnego $i \in \mathbb{N},\ y \not\equiv x_i$.

($\vec{s}3$) $(PQ)[\vec{x}/\vec{N}] = P[\vec{x}/\vec{N}]Q[\vec{x}/\vec{N}]$

($\vec{s}4$) $(\lambda y.\,P)[\vec{x}/\vec{N}] = \lambda y.\,P[\vec{x}/\vec{N}]$, jeśli $y \neq x_i$ dla wszystkich $i \in \mathbb{N}$ i $y \notin \bigcup_{i\in\mathbb{N}} FV(N_i)$

Konwencja. *Jeśli* $N_i \equiv x_i$ *dla wszystkich poza skończenie wieloma* $i_1, i_2, \ldots, i_n \in \mathbb{N}$, *to* $[x_{i_1}/N_{i_1}, x_{i_2}/N_{i_2}, \ldots, x_{i_n}/N_{i_n}] \equiv [\vec{x}/\vec{N}]$.

Przykład 5. Zauważmy, że podstawienia w myśl Definicji 4 i Definicji 10 mogą, ale nie muszą, prowadzić do różnych rezultatów.

a) $(xy)[y/x][x/u] = uu$,
$(xy)[y/x, x/u] = ux$.

b) $(\lambda x.\, yx)[x/y][y/z] = \lambda x.\, zx$,
$(\lambda x.\, yx)[x/y, y/z] = \lambda x.\, zx$.

1.2 Redukcja

Sens obliczeniowy λ-termom nadajemy przez określenie na $\mathbf{\Lambda}$ operacji β- i η-redukcji. Pożądane jest, żeby operacje te wykonywane na podtermach pozostowały w *zgodzie* ze strukturą całego λ-termu.

Definicja 11. (Relacja zgodna) Relację binarną $\mathcal{R}$ na zbiorze $\mathbf{\Lambda}$ nazywamy *zgodną*, jeśli dla dowolnych M, N, $P \in \mathbf{\Lambda}$ zachodzą następujące warunki:

(c1) Jeśli $M\mathcal{R}N$, to $(\lambda x.\, M)\,\mathcal{R}\,(\lambda x.\, N)$ dla dowolnej λ-zmiennej x.

(c2) Jeśli $M\mathcal{R}N$, to $(MP)\,\mathcal{R}\,(NP)$.

(c3) Jeśli $M\mathcal{R}N$, to $(PM)\,\mathcal{R}\,(PN)$.

Przez *domknięcie relacji* $\mathcal{R}_1$ będziemy rozumieli najmniejszą (w sensie mnogościowym) relację $\mathcal{R}_2$ taką, że $\mathcal{R}_1 \subset \mathcal{R}_2$. Z pewnego rodzaju domknięciami, ze względu na ich szczególną rolę, wiążemy następującą notację:

(a) Przez $\mathcal{R}^+$ oznaczamy przechodnie domknięcie relacji $\mathcal{R}$.

(b) Przez $\mathcal{R}^*$ oznaczamy zwrotnie domknięcie relacji $\mathcal{R}^+$.

(c) Przez $=_\mathcal{R}$ oznaczamy symetryczne domknięcie relacji $\mathcal{R}^*$.

Dla lepszego zrozumienia powyższych operacji warto zauważyć, że (b) wyznacza praporzadek, który w odniesieniu do redukcji określonych na $\mathbf{\Lambda}$ można rozumieć jako graf skierowany (w przypadku $\mathbf{\Lambda}$ być może nieskończony) w którym krawędzie odpowiadają możliwym krokom obliczenia, zaś (c) – kongruencję, która znów w szczególnym odniesieniu do λ-termów, będzie dokonywała podziału w $\mathbf{\Lambda}$ ze względu na rezultat obliczenia.

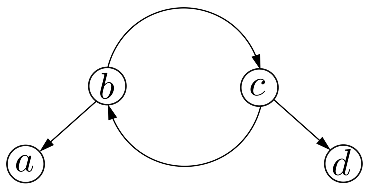
Definicja 12. Niech $\to$ będzie relacją binarną w zbiorze A.

(CR) Powiemy, że $\to$ ma *własność Churcha-Rossera*, jeśli dla dowolnych a, b, $c \in A$ takich, że $a \to^* b$ oraz $a \to^* c$ istnieje $d \in A$ takie, że $b \to^* d$ i $c \to^* d$. Innymi słowy, przemienny jest diagram:

$$\begin{array}{ccc} a & \xrightarrow{\ *\ } & b \\ {\scriptstyle *}\big\downarrow & & \big\downarrow{\scriptstyle *} \\ c & \xrightarrow{\ *\ } & d \end{array}$$

(WCR) Powiemy, że $\to$ ma *słabą własność Churcha-Rossera*, jesli dla dowolnych $a, b, c \in A$ takich, że $a \to b$ oraz $a \to c$ istnieje $d \in A$ takie, że $b \to^* d$ i $c \to^* d$. Innymi słowy, przemienny jest diagram:

$$\begin{tikzcd} a \arrow[r, "*"] \arrow[d, "*"'] & b \arrow[d] \\ c \arrow[r] & d \end{tikzcd}$$



Rysunek 1: Rozważmy graf skierowany, w którym krawędzie odpowiadają relacji $\to$ w zbiorze $\{a, b, c, d\}$. Widzimy, że relacja $\to$ ma własnosność WCR, ale nie ma własności CR.

Definicja 13. (Postać normalna) Powiemy, że $x \in A$ jest *redukowalny*, jeśli istnieje $y \in A$ takie, że $x \to y$. W przeciwnym wypadku powiemy, że x jest w *postaci normalnej* i będziemy pisali $x \in \mathrm{NF}$.

Element $y \in A$ nazywamy *postacią normalną* $x \in A$, jesli $x \to^* y$ i $y \in \mathrm{NF}$. Jeśli y jest postacią normalną x i y jest jedyną postacią normalną x, to piszemy $x \downarrow y$. W przeciwnym wypadku, czyli jeśli istnieją $y, z \in \mathrm{NF}, y \neq z$ takie, że $x \to^* y$ i $x \to^* z$, powiemy, że x jest *niejednoznaczny.*

Definicja 14. Niech $\to$ będzie relacją binarną na zbiorze A.

(WN) Powiemy, że relacja $\to$ jest *słabo normalizująca*, jeśli dla dowolnego $a \in A$ istnieje $a' \in \mathrm{NF}$ taki, że $a \to^* a'$. W takim wypadku o $a \in A$ będziemy mówili, że jest *słabo normalizowalny* i pisali $a \in \mathrm{WN}$.

(SN) Powiemy, że relacja $\to$ jest *silnie normalizująca*, jeśli nie istnieje nieskończony ciąg relacji $a_0 \to a_1 \to a_2 \to \ldots$. W takim wypadku o $a \in A$ będziemy mówili, że jest *silnie normalizowalny* i pisali $a \in \mathrm{SN}$.

Twierdzenie 2. *(Lemat Newmana) Niech $\to$ bedzie relacją binarną mającą własność SN. Jeśli $\to$ ma własność WCR, to $\to$ ma własność CR.*

Dowód. Niech $\to$ będzie relacją binarną na A o własności SN i WCR. Ponieważ $\to$ jest SN, to każdy a jest normalizowalny.

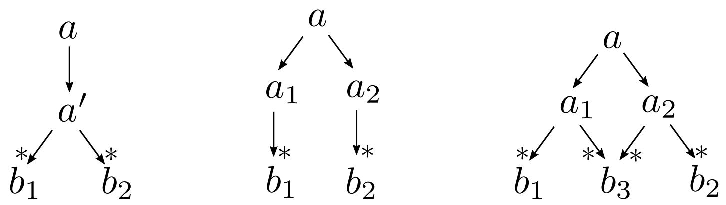
Jeśli A nie zawiera elementów niejednoznacznych, to twierdzenie zachodzi w sposób trywialny. Przypuśćmy, że $a \in A$ jest niejednoznaczny. Twierdzimy, że istnieje $a' \in A$ taki, że $a \to a'$ i a' jest niejednoznaczny. Niech $b_1, b_2 \in \mathrm{NF}$, $b_1 \neq b_2$ i $a \to^* b_1$ oraz $a \to^* b_2$. Ponieważ $b_1 \neq b_2$, to istnieją a_1, $a_2 \in A$ takie, że:

$$a \to a_1 \to^* b_1 \quad \text{oraz} \quad a \to a_2 \to^* b_2$$

Jeśli $a_1 = a_2$, to $a' = a_1 = a_2$ i wystarczy wybrać $a' = a_1$. Jeśli jednak $a_1 \neq a_2$, to z własności WCR istnieje $b_3 \in A$ taka, że $a_1 \to^* b_3$ oraz $a_2 \to^* b_3$. Z własności SN możemy przyjąć, że b_3 jest w postaci normalnej. Zachodzą więc dwa przypadki:

i) $a_1 = a_2$. Wówczas wystarczy ustalić $a' = a_1$ albo $a' = a_2$ (Rysunek 2a).

ii) $a_1 \neq a_2$ (Rysunek 2b). Wówczas z WCR istnieje $b_3 \in A$ takie, że $a_1 \to^* b_3$ oraz $a_2 \to^* b_3$ (Rysunek 2c). Przypuśćmy, że $b_3 \in \mathrm{NF}$. Ponieważ $b_1 \neq b_3$, to $b_3 \neq b_1$ lub $b_3 \neq b_2$, zatem możemy wybrać $a' = a_1$ albo $a' = a_2$.



(a) (b) (c)

Rysunek 2: Warianty konstruowania redukcji.

Stosując powyższe rozumowanie do a' otrzymujemy kolejny element niejednoznaczny. a zatem możemy skontruować nieskończony ciąg redukcji, wbrew założeniu, że relacja $\to$ jest SN. Zatem A nie zawiera elementów niejednoznacznych. □

Definicja 15. (β-redukcja) β-redukcją nazywamy najmniejszą (w sensie mnogościowym) zgodną na $\mathbf{\Lambda}$ relację binarną $\to_\beta$ taką, że

$$(\lambda x.\, M)N \to_\beta M[x/N].$$

β-redeksami bedziemy nazywali wyrażenia postaci $(\lambda x.\, M)N$, zaś rezultat ich β-redukcji w postaci termu $M[x/N]$ – *β-reduktem.* Przez $\to_\beta^+$, $\to_\beta^*$, $=_\beta$ oznaczamy odpowiednie domknięcia relacji β-redukcji. Symbolem $\leftarrow_\beta$ oznaczać będziemy relację odwrotną do β-redukcji, zaś przez $\underset{\beta}{\leftrightarrow}$ jej symetryczne domknięcie.

Ciągiem β-redukcji nazywamy kazdy skończony lub nieskończony ciąg λ-termów M_0, M_1, … taki, że $M_0 \to_\beta M_1 \to_\beta \ldots$

Relację $=_\beta$ nazywamy *β-konwersją.* Zauważmy, że $M =_\beta N$ wtedy i tylko wtedy, gdy istnieje skończony ciąg λ-termów $M \equiv M_0$, M_1, …, $M_n \equiv N$ taki, że $M_i \to_\beta M_{i+1}$ lub $M_{i+1} \to_\beta M_i$ dla $0 \leq i \leq n$.

Przykład 6. Wszystkie pary λ-termów ze zbioru

$$\{(\lambda x.\,(\lambda y.\,yx)\,z)\,v,\ (\lambda y.\,yv)\,z,\ (\lambda x.\,zx)\,v,\ zv\}$$

są swoimi β-konwersami. Mamy:

$$\begin{aligned}(\lambda y.\,yv)z \to_\beta zv \ \leftarrow_\beta (\lambda x.\,zx)v,\\ (\lambda y.\,yv)z \leftarrow_\beta (\lambda x.\,(\lambda y.\,yx)z)v \to_\beta (\lambda x.\,zx)v.\end{aligned}$$

Lemat 2. *Dla dowolnych $N, Q \in \mathbf{\Lambda}$, jeśli $N[y/Q] \in \mathrm{SN}_\beta$, to $N \in \mathrm{SN}_\beta$. Jeśli dodatkowo $y \in \mathrm{FV}(N)$, to także $Q \in \mathrm{SN}_\beta$.*

Dowód. Dowód przeprowadzamy przez indukcję względem definicji 4'. □

Definicja 16. (Strategia redukcji) Strategią redukcji nazywamy każde odwzorowanie $S : \mathbf{\Lambda} \to \mathbf{\Lambda}$ postaci

$$S(M) = \begin{cases} M, & \text{jeśli } M \in \mathrm{NF}_\beta,\\ M', & \text{jeśli } M \to_\beta M'. \end{cases}$$

Strategię S nazywamy *normalizującą*, jeśli dla każdego $M \in \mathrm{WN}_\beta$ istnieje $i \in \mathbb{N}$ takie, że $F^i(M) \equiv \underbrace{F(F(\ldots(F}_{i\text{-razy}}(M))\ldots)) \in \mathrm{NF}_\beta$.

Przykład 7. (a) Oznaczmy $\mathrm{Y} \equiv \lambda f.\,(\lambda x.\,(f(xx))\lambda x.\,(f(xx)))$ i niech F będzie dowolnym λ-termem. Wówczas otrzymujemy nieskończony ciąg redukcji postaci

$$\begin{aligned}\mathrm{Y}F &\equiv (\lambda f.\,(\lambda x.\,(f(xx))\lambda x.\,(f(xx))))F\\ &\to_\beta (\lambda x.\,F(xx))\lambda x.\,F(xx)\\ &\to_\beta F((\lambda x.\,F(xx))\lambda x.\,F(xx))\\ &\to_\beta F(\underbrace{F((\lambda x.\,F(xx))\lambda x.\,F(xx))}_{=_\beta \mathrm{Y}F})\\ &\to_\beta \ldots\end{aligned}$$

Y nazywamy *kombinatorem punktu stałego*. Widzimy, że relacja β-redukcji w rachunku λ nie jest ani słabo, ani silnie normalizująca.

(b) Niech $\Omega \equiv (\lambda x.\,xx)(\lambda x.\,xx)$. Ω jest β-redeksem, którego redukcja prowadzi do ponownego otrzymania termu Ω i w konsekwencji do stałego ciągu redukcji postaci:

$$\Omega \to_\beta \Omega \to_\beta \Omega \to_\beta \ldots$$

(c) Niech $\Delta \equiv \lambda x.\ xxx$. Wówczas:

$$\Delta\Delta \to_\beta \Delta\Delta\Delta \to_\beta \Delta\Delta\Delta\Delta \to_\beta \ldots$$

Ponownie, ponieważ każda redukcja powoduje wydłużenie termu, $\Delta\Delta$ nie ma postaci normalnej i w konsekwencji każdy powstały ciąg redukcji termu $\Delta\Delta$ jest nieskończony.

(d) Redukcja λ-termu posiadającego więcej niż jeden redeks może prowadzić do różnych (choć β-równowaznych) reduktów. Zależy to od wyboru strategii redukcji. Rozważmy następujący term: $(\lambda u.\ v)\,\Omega$. Konsekwentne redukowanie podtermu Ω prowadzić musi do niekończącego się stałego ciągu redukcji

$$(\lambda u.\ v)\,\Omega \to_\beta (\lambda u.\ v)\,\Omega \to_\beta \ldots$$

Wybierając strategię polegającą na aplikacji Ω do $(\lambda u.\ v)$ otrzymujemy natychmiastowo redeks w postaci normalnej.

Definicja 17. (η-redukcja) η-redukcją nazywamy najmniejszą (w sensie mnogościowym) zgodną na $\mathbf{\Lambda}$ relację binarną $\to_\eta$ taką, że

$$\lambda x.\,Mx \to_\eta M, \text{ o ile } x \notin \mathrm{FV}(M).$$

η-redukcja pozwala na pominięcie niczego nie wnoszącej λ-abstrakcji. Operację odwrotną nazywamy *η-abstrakcją*, zaś λ-termy będące w którejkolwiek z tych relacji nazywamy *η-konwersami*. Operacja ta nie ma wpływu na rezultat obliczenia, jedynie optymializuje zapis λ-termów i stąd ma duże znaczenie stylistyczne w programowaniu funkcyjnym.

Przykład 8. Przypuśćmy, że $(+1) \in \mathbf{\Lambda}$. Wówczas $\lambda x.((+1)x) =_\eta (+1)$.

Widzieliśmy, że β-redukcja może prowadzić do uzyskania rezultatu lub nie. Fakt 1 i następujące po nim Wniosek 3 i Wniosek 4 stwierdzają, że jeśli tylko mamy pewność, że λ-term ma postać normalną, to jest ona wyznaczona jednoznacznie i doprowadzi nas do niej każda strategia normalizująca. Fakt 1 to klasyczne twierdzenie, którego dowód można znaleźć w [Bar92] i ze względu na jego obszerność pozwalamy sobie go pominąć.

Fakt 1. *(Twierdzenie Churcha-Rossera). β-redukcja ma własność CR.*

Wniosek 3. *Jeśli $M =_\beta N$, to istnieje $L \in \mathbf{\Lambda}$ takie, że $M \to_\beta^* L$ i $N \to_\beta^* L$.*

Dowód. Niech $M, N \in \mathbf{\Lambda}$ będą takie, że $M =_\beta N$. Wówczas istnieje ciąg λ-termów $M_0, M_1, \ldots, M_{n-1}, M_n$ taki, że

$$M_0 \underset{\beta}{\leftrightarrow} M_1 \underset{\beta}{\leftrightarrow} \ \ldots\ \underset{\beta}{\leftrightarrow} M_{n-1} \underset{\beta}{\leftrightarrow} M_n,$$

gdzie $M_0 \equiv M$ i $M_n \equiv N$. Dowód przeprowadzimy przez indukcję względem n. Rozważmy następujące przypadki:

(1) Jeśli $n = 0$, to $M \equiv N$. Ustalając $L \equiv M (\equiv N)$ w oczywisty sposób $M \to_\beta^* L$ i $N \to_\beta^* L$.

(2) Jeśli $n = k > 0$, to istnieje $M_{k-1} \in \mathbf{\Lambda}$ takie, że

$$M \equiv M_0 \underset{\beta}{\leftrightarrow} M_1 \underset{\beta}{\leftrightarrow} \dots \underset{\beta}{\leftrightarrow} M_{k-1} \underset{\beta}{\leftrightarrow} M_k \equiv N$$

Z założenia indukcyjnego wiemy, że istnieje $L' \in \mathbf{\Lambda}$ takie, że $M_0 \to_\beta^* L'$ i $M_{k-1} \to_\beta^* L'$. Ponieważ $\underset{\beta}{\leftrightarrow}$ jest symetryczna, rozważmy osobno przypadki $M_{k-1} \to_\beta M_k$ i $M_k \to_\beta M_{k-1}$.

(a) Jeśli $M_{k-1} \to_\beta M_k$, to tym bardziej $M_{k-1} \to_\beta^* M_k$. Ponieważ $M_{k-1} \to_\beta^* L'$, to korzystając Faktu 1 wnosimy, że istnieje $L \in \mathbf{\Lambda}$ taki, że $L' \to_\beta^* L$ i $M_k \to_\beta^* L$, czyli

$$\begin{tikzcd} M_0 \arrow[r, leftrightarrow] \arrow[dr, "*"] & \dots \arrow[r, leftrightarrow] & M_{k-1} \arrow[r] \arrow[dl, "*"'] & M_k \arrow[d, "*"] \\ & L' \arrow[rr, "*"] & & L \end{tikzcd}$$

(b) Jeśli $M_k \to_\beta M_{k-1}$, to ponieważ $M_{k-1} \to_\beta^* L'$, natychmiast otrzymujemy, że $M_k \to_\beta^* L'$. Ustalając $L \equiv L'$ otrzymujemy tezę.

□

Wniosek 4. *(1) Jeśli N to postać normalna M, to $M \to_\beta^* N$.*

(2) Każdy λ-term ma co najwyżej jedną postać normalną.

Dowód. (1) Przypuśćmy, że $N \in \mathrm{NF}_\beta$ i $M =_\beta N$. Wówczas z Wniosku 3 istnieje L takie, że $M \to_\beta^* L$ i $N \to_\beta^* L$. Ponieważ $N \in \mathrm{NF}_\beta$ i $N \to_\beta^* L$, to $N \equiv L$. Ponieważ $M \to_\beta^* L$, to $M \to_\beta^* N$.

(2) Przypuśćmy, że M ma dwie różne postacie normalne, N_1, N_2. Wówczas z cześci (1) tego twierdzenia, $M \to_\beta^* N_1$ i $M \to_\beta^* N_2$. Z Faktu 1 istnieje $L \in \mathbf{\Lambda}$ taki, że $N_1 \to_\beta^* L$ i $N_2 \to_\beta^* L$. Ponieważ N_1, $N_2 \in \mathrm{NF}_\beta$, to $N_1 \equiv L \equiv N_2$.

□

1.3 Kodowanie typów danych

Prosta składnia języka rachunku λ pozwala wyrazić zaskakująco wiele struktur danych reprezentując je i operacje na nich jako funkcje. Z tego powodu, stanowiąc inspirację dla wielu projektantów języków programowania, uchodzi za protoplastę rodziny języków funkcyjnych, chociaż bezpośrednio nie ma on praktycznego zastosowania w praktyce programistycznej. Rozwój tej legendy dobrze oddaje cykl klasycznych artykułów (tzw. *Lambda Papers*) zapoczątkowany przez dokumentację języka Scheme [SS75].

Najpopularniejszym sposobem reprezentacji danych przez funkcje w rachunku λ oparty jest na kodowaniu liczb Peano za pomocą tzw. liczebników Churcha. Metoda ta, ze względu na wynikające zeń problemy natury złożonościowej [KPJ14], ma obecnie wyłącznie walory edukacyjne, dlatego w dalszej cześci pracy pokażemy tzw. kodowanie Scotta. Jest ona interesująca ze względu na praktyczną możliwość reprezentacji algebraicznych typów danych (ADT[1]) znanych ze współczesnych języków funkcyjnych [Jan13], pozwalając tym samym zaimplementować te konstrukcje na przykład w paradygmacie imperatywnym. Fakt, że każdy typ danych można zastąpić tym sposobem odpowiadającą mu funkcją, wskazuje na metodę konstruowania prostych języków funkcyjnych [JKP06] oraz na uniwersalność rachunku λ jako języka przejściowego dla kompilatorów języków funkcyjnych [PL92, Rozdział 3].

1.3.1 Algebraiczne typy danych

Algebraiczne typy danych są podstawowym środkiem współczesnych języków funkcyjnych do wyrażania struktur danych. Powstają one przy użyciu tzw. typów sumacyjnych i typów produktowych, jednak pojęcia te na gruncie formalnym będą szczegółowo omówione w późniejszej części pracy. Na potrzeby prezentacji poszczególnych kodowań wystarczą nam w tym rozdziale intuicje o ADT zbudowane na gruncie następujących definicji w języku Haskell:

```haskell
data Boolean     = True
                 | False
data Tuple a b   = Tuple a b
data Temperature = Fahrenheit Int
                 | Celsius Int
data Maybe a     = Nothing
                 | Just a
data Nat         = Zero
                 | Succ Nat
data List t      = Nil
                 | Cons t (List t)
```

Definicja typu rozpoczynają się od słowa kluczowego `data`[2] po którym występuje *konstruktor typu*. Na wzór notacji BNF, typy przyjmują jedną z *wartości* odzie-

[1]Skrót od angielskojęzycznego *Algebraic Data Types*; nie należy mylić z *Abstract Data Types*.

[2]Dyskusja ta ma na celu wyłącznie ustalenie uwagi; świadomi jesteśmy niuansów związanych z określaniem synonimów typów lub definiowaniem typów przy pomocy słowa kluczowego `newtype`.

lonych znakiem "|". Każda z wartości składa się z *konstruktora wartości* i ewentualnie występujących po nim *parametrów typowych*. Zauważmy, że umożliwia to rekurencyjnie konstruowanie typów, tak jak w wypadku `Nat` i `List`.

Pokażemy, że algebraiczne typy danych możemy reprezentować w zwięzły sposób w rachunku λ bez typów. Przedstawione tutaj koncepcje w zaskakujący sposób przenoszą się do bardziej złożonych typowanych systemów rachunku λ.

1.3.2 Proste typy wyliczeniowe

Typy wyliczeniowe to typy, które reprezentują możliwe warianty przyjmowanej wartości. Najprostrzym nietrywialnym przykładem takiego typu jest `Boolean`. Ma on dwa konstruktory wartości: `True`, `False`. Praca z tego rodzaju typami wymaga mechanizmu dopasowywania wzorców (ang. *pattern-matching*) [PL92, Rozdział IV], który pozwala na wybór częściowej definicji funkcji w zależności od zadanego konstruktora wartości. Ponieważ w rachunku λ wyrażenia nie mają typów (lub, przyjmując perspektywę systemów z typami: wszystkie wyrażenia mają jeden, ten sam typ), interesowało nas będzie nie bezpośrednie kodowanie typu, ale kodowanie mechanizmu, który odpowiada za dopasowywanie wzorców. Posłużmy się znowu przykładem z języka Haskell i określmy funkcję odpowiadającą wykonaniu instrukcji warunkowej:

```
if True  a b = a
if False a b = b
```

gdzie `True` i `False` są wartościami typu `Boolean`. Właśnie ze względu na nie, mechanizm dopasowywania wzorca wybiera odpowiednią implementację instrukcji warunkowej. Ten sam efekt osiągnęlibyśmy kodując `True` i `False` w rachunku λ w następujący sposób:

$$\begin{aligned} \text{True} &\equiv \lambda ab.\, a \\ \text{False} &\equiv \lambda ab.\, b \end{aligned}$$

Wówczas funkcję `if` możemy reprezentować wyrażeniem $\text{if} \equiv \lambda cte.\, cte$ lub jego η-reduktem: $\lambda c.\, c$.

1.3.3 Pary w rachunku λ

Parą nazywamy każdy nierekurencyjny typ, który posiada jeden konstruktor wartości parametryzowany przez dwa typy. W takim wypadku potrzebujemy dwóch projekcji zwracających odpowiednio pierwszy i drugi element pary. Przykładem takiego typu jest `Tuple`. Mamy wówczas:

```
fst (Tuple a b) = a
snd (Tuple a b) = b
```

Tego rodzaju typy możemy reprezentować przez tak zwane *domknięcie* (ang. *closure*), czyli cześciową aplikację termu. Standardowym sposobem reprezentacji pary w rachunku λ jest:

$$\text{Tuple} \equiv \lambda abf.\, fab$$

Aplikując Tuple tylko do dwóch termów (*domykając* term Tuple) otrzymujemy reprezentację pary. Pozostały, trzeci argument f nazywamy *kontynuacją*, gdyż aplikując (Tuple x y) dla dowolnych $x, y \in \mathbf{\Lambda}$ do pewnego $f \in \mathbf{\Lambda}$, w konsekwencji x i y zostają zaaplikowane do f. Zauważmy, że wówczas reprezentacja `fst` i `snd` ma postać:

$$\text{fst} \equiv \lambda t.\, t(\lambda ab.\, a)$$
$$\text{snd} \equiv \lambda t.\, t(\lambda ab.\, b)$$

Przykład 9. Wprowadzone konstrukcje pozwalają nam na definicję skończonych (w sensie liczby konstruktorów) typów. Rozważmy następujące przykłady:

a) Konstruktory wartości typu `Maybe` możemy reprezentować przez

$$\text{Nothing} \equiv \lambda nj.\, n$$
$$\text{Just} \equiv \lambda anj.\, ja$$

Rozważmy następującą funkcję:

```
maybe :: b -> (a -> b) -> Maybe a -> b
maybe n _ Nothing  = n
maybe _ f (Just x) = f x
```

Odpowiadająca jej reprezentacja to

$$\text{maybe} \equiv \lambda bft.\, tb(\lambda a.\, fa)$$

b) Rozważmy następującą funkcję

```
fromTemperature :: Temperature -> Int
fromTemperature (Fahrenheit a) = a
fromTemperature (Celsius a) = a
```

Ustalając reprezentację konstruktorów `Fahrenheit` i `Celsius`:

$$\text{Fahrenheit} \equiv \lambda tfc.\, ft$$
$$\text{Celsius} \equiv \lambda tfc.\, ct$$

otrzymujemy reprezentację funkcji `formTemperature` postaci:

$$\text{fromTemperature} \equiv \lambda t.\, t(\lambda f.\, f)(\lambda c.\, c)$$

1.3.4 Kodowanie rekurencji

Rozważmy następującą funkcję dodawania liczb Peano w języku Haskell:

```
add Zero     m = m
add (Succ n) m = Succ (add n m)
```

Funkcję tę możemy wyrazić w rachunku λ przy pomocy kodowania Scotta w następujący sposób:

$$\mathrm{add}_0 \equiv \lambda nm.\, n\, m\, (\lambda n.\, \mathrm{Succ}(\mathrm{add}_0\, n\, m))$$

Formalizm rachunku λ nie pozwala na okreslanie nowych nazw i rekurencyjne odnoszenie się przez nie do nich samych. Standardową techniką w rachunku λ do określania funkcji w ten sposób jest użycie operatora punktu stałego Y. Przypomnijmy:

$$\mathrm{Y} \equiv \lambda f.\, (\lambda x.\, (f(xx))\lambda x.\, (f(xx)))$$

Wówczas określamy

$$\mathrm{add}_\mathrm{Y} \equiv \mathrm{Y}\, (\lambda a\, n\, m.\ nm\, (\lambda n.\, \mathrm{Succ}(a\, n\, m)))$$

Mając na uwadze możliwość przeprowadzenia powyższej konstrukcji przy użyciu rekurencji, będziemy dopuszczali w notacji odnoszenie się wprowadzanych λ-termów do nich samych.

1.3.5 Kodowanie Scotta typów rekursywnych

Stosując metody kodowania prostych typów wyliczeniowych i par, łatwo odnajdujemy reprezentację konstruktorów wartości dla typów `Nat` i `List`:

$$\mathrm{Zero} \equiv \lambda zs.\, z \qquad \mathrm{Nil} \equiv \lambda nc.\, n$$
$$\mathrm{Succ} \equiv \lambda nzs.\, sn \qquad \mathrm{Cons} \equiv \lambda x\, x_s\, n\, c.\ c\, x\, x_s$$

Zwróćmy uwagę, że konstruktory Nat i Maybe są swoimi α-konwersami. Podobieństwo nie jest przypadkowe: na poziomie typów konstrukcja Maybe jest odpowiednikiem brania następnika. Określając dodatkowo Void $\equiv \lambda x.x$ jako element neutralny działania łącznego, otrzymujemy na poziomie typów strukturę półpierścienia z działaniem mnożenia okresloną przez konstrukcję par i dzałaniem dodawania określonego przez konstrukcję typów wyliczeniowych. Stąd algebraicze typy danych biorą swoją nazwę.

Z łatwością możemy określić teraz operacje brania poprzednika, głowy i ogona listy, odpowiednio:

$$\begin{aligned} \text{pred} &\equiv \lambda n.\, n\, \text{undef}\, (\lambda m.\, m) \\ \text{head} &\equiv \lambda x_s.\, x_s\, \text{undef}\, (\lambda x_s. x) \\ \text{tail} &\equiv \lambda x_s.\, \text{undef}\, (\lambda x_s.\, x_s) \end{aligned}$$

gdzie undef jest stałą o którą rozszerzamy rachunek λ celem sygnalizowania błędnej aplikacji.

Celem lepszego porównania kodowania Churcha i Scotta podamy reprezentacje funkcji `foldl` dla typu `Nat`. Określmy:

```
foldl f x Zero     = x
foldl f x (Succ n) = f (foldl f x n)
```

`foldl` może być przy pomocy kodowania Scotta zapisane jako

$$\text{foldl} \equiv \lambda f\, x\, n.\ n\, x\, (\lambda n.\ (\text{foldl}\ f\ x\ n))$$

Ogólnie, przy pomocy `foldl` wyabstrahowujemy pojęcie tzw. rekursji od strony ogona (ang. *tail recrusion*), w teorii obliczalności nazywane rekursją prostą lub, popularnie, zwijaniem od lewej. Operator foldl spełnia następującą własność [Hut99]

$$f = \text{foldl}\ \varphi\ a \iff \begin{cases} f\ \text{Zero} = a \\ f\ (\text{Succ}\ n) = \varphi\ (f\ n) \end{cases} \tag{2}$$

1.3.6 Kodowanie Churcha typów rekursywnych

Przedstawimy teraz klasyczny sposób kodowania typów po raz pierwszy zaprezentowany dla liczb naturalnych przez A. Churcha w [Chu41]. Różni się on od kodowania Scotta tylko w przypadku typów rekursywnych, w pozostałych przypadkach obydwa kodowania dają te same rezultaty. Typ `Nat` ma dwa konstruktory: `Zero` i `Succ`. W kodowaniu Churcha reprezentujemy je w następujący sposób:

$$\begin{aligned} \text{Zero}_{Ch} &\equiv \lambda f x.\, x \\ \text{Succ}_{Ch} &\equiv \lambda n f x.\, f\, (n\, f\, x) \end{aligned}$$

Wyrażenia będące skutkiem konsekwentnej aplikacji Succ do Zero w literaturze popularnie nazywa się *liczebnikami Churcha* i oznacza następująco:

$$\begin{aligned} \bar{1} &\equiv \text{Succ}_{Ch}\ \text{Zero}_{Ch} \ =_\beta\ \lambda f x.\, f x \\ \bar{2} &\equiv \text{Succ}_{Ch}\ \text{Succ}_{Ch}\ \text{Zero}_{Ch} \ =_\beta\ \lambda f x.\, f\, f\, x \\ &\vdots \\ \bar{n} &\equiv \text{Succ}^n_{Ch}\ \text{Zero}_{Ch} \ =_\beta\ \lambda f x.\, f^n\, x \end{aligned}$$

Liczba naturalna n jest kodowana przez funkcję w której jej pierwszy argument jest aplikowany n razy do drugiego argumentu. Porównując je do kodowania Scotta widzimy, że różnica polega na aplikowaniu do kontynuacji termu $(n\, f\, x)$ w przypadku brania następnika. Da się pokazać [HIN05], że liczebniki Churcha są w istocie operacją `foldl` na argumentach `Succ` i Zero. Istotnie, niech nat $\equiv \lambda c.\ c$ Succ Zero. Wówczas nat $\bar{n} =_\beta \bar{n}$. Z tego powodu kodowanie operacji na liczebnikach Churcha, lub ogólnie – funkcji opartych na rekursji prostej po zbiorze liczb naturalnych – jest wyjątkowo proste przy użyciu tej metody. Przykładowo, używając metody Churcha, operację dodawania kodujemy w następujący sposób:

$$\mathrm{add}_{Ch} \equiv \lambda n\, m.\ n\ \mathrm{Succ}_{Ch}\ m$$

Dla porównania, używając kodowania Scotta:

$$\mathrm{add}_{S} \equiv \lambda n\, m.\ \mathrm{foldl}\ \mathrm{Succ}\ n\ m$$

1.3.7 Ogólny schemat kodowania Scotta typów ADT

W ogólnym przypadku, mając nastepującą definicję ADT:

```
data type_constructor t1 t2 ... tk = C1 t11 ... t1n1
                                   | C2 t21 ... t2n2
                                   ...
                                   | Cm tm1 ... tmnm
```

dla $m, n \in \mathbb{N}$, wiążemy z nią reprezentację każdego z konstruktorów:

$$\begin{aligned}
\mathrm{C}_1 &\equiv \lambda t_{11}\, t_{12} \dots t_{1n_1}\, f_1\, f_2 \dots f_m.\ f_1 t_{11} t_{12} \dots t_{1n_1} \\
\mathrm{C}_2 &\equiv \lambda t_{21}\, t_{22} \dots t_{2n_2}\, f_1\, f_2 \dots f_m.\ f_2 t_{21} t_{22} \dots t_{2n_2} \\
&\vdots \\
\mathrm{C}_\mathrm{m} &\equiv \lambda t_{m1}\, t_{m2} \dots t_{mn_m}\, f_1\, f_m \dots f_m.\ f_1 t_{m1} t_{m2} \dots t_{mn_m}
\end{aligned}$$

Wówczas następującą definicję cześciową funkcji `f`:

```
f (C1 v11 ... v1n1) = y1
...
f (Cm vm1 ... vmnm) = ym
```

kodujemy przy za pomocą następujego λ-termu:

$$\begin{aligned}
\lambda x.\ x\, &(\lambda v_{11} \dots v_{1n_1}.\ y_1) \\
&\vdots \\
&(\lambda v_{m1} \dots v_{mn_m}.\ y_m)
\end{aligned}$$

gdzie y_1 są kodowaniami Scotta `yi` dla $i \in \mathbb{N}$.

2 Rachunek λ z typami prostymi

2.1 Typy proste

Niech U będzie przeliczalnie nieskończonym zbiorem zmiennych przedmiotowych p, q, $\ldots$ (być może indeksowanych liczbami naturalnymi), które będziemy nazywali *zmiennymi typowymi.*

Definicja 18. (Typy proste) *Typami prostymi* będziemy określali najmniejszy w sensie mnogościowym zbiór wyrażeń taki, że:

(S1) Jeśli p jest zmienną typową, to p jest typem prostym.

(S2) Jeśli σ i τ są typami prostymi, to $(\sigma \to \tau)$ jest typem prostym.

Typy proste zbudowane tylko wedle reguły (S1) nazywamy typami *atomowymi*, zaś wyrażenia zbudowe wedle reguły (S2) – typami *funkcyjnymi.* Zbiór typów prostych określony w myśl powyższej definicji będziemy oznaczali przez $\mathbb{T}$. Definicję 18 można równoznacznie wyrazić przy pomocy notacji Backusa-Naura. Wówczas ma ona następującą, zwięzłą postać:

$$\mathbb{T} \leftarrow U \mid (\mathbb{T} \to \mathbb{T})$$

Późniejsze litery alfabetu greckiego (σ, τ, ρ, $\ldots$), być może z indeksami, będą służyły nam za zmienne metasyntaktyczne do oznaczania typów prostych. Dla lepszej czytelności będziemy pomijali najbardziej zewnętrzne nawiasy. Konstruktor typu $\to$ wiąże prawostronnie; oznacza to, że typy $\sigma \to \tau \to \rho$ oraz $\sigma \to (\tau \to \rho)$ będziemy uznawali za tożsame.

Zauważmy, że obiekty skonstruowane w myśl Definicji 18 mają strukturę drzewa binarnego. Wysokość takiego drzewa będziemy nazywali *stopniem* typu.

Definicja 19. (Stopień typu) Stopniem typu nazywamy następująco określoną funkcję $\delta : \mathbb{T} \to \mathbb{N}$

$$\begin{aligned} \delta(p) &= 0, \text{ gdzie } p \text{ jest typem atomowym,} \\ \delta(\sigma \to \sigma) &= 1 + \max\left(\delta(\sigma),\ \delta(\sigma)\right). \end{aligned}$$

Definicja 20. (Stwierdzenie, deklaracja, kontekst, sąd)

(1) *Stwierdzeniem* (ang. *statement*) nazywamy każdy napis postaci $M : \sigma$, gdzie $M \in \mathbf{\Lambda}$ i $\sigma \in \mathbb{T}$. W stwierdzeniu $M : \sigma$ λ-term M nazwamy *podmiotem* (ang. *subject*), zaś σ – *predykatem.*

(2) *Deklaracją* (ang. *declaration*) nazywamy każde stwierdzenie w którym podmot jest zmienną termową.

(3) *Kontekstem* (ang. *context*) nazywamy skończony liniowo uporządkowany zbiór (*listę*) deklaracji, w którym wszystkie podmioty są wzajemnie różne.

(4) *Sądem* (ang. *judgement*) nazywamy kazdy napis postaci $\Gamma \vdash M : \sigma$, gdzie Γ jest kontekstem, zaś $M : \sigma$ – stwierdzeniem.

Definicja 21. (1) Jeśli $\Gamma = (x_1 : \sigma_1, \ldots, x_n : \sigma_n)$, to liniowo uporządkowany zbiór $\mathrm{dom}\Gamma = (x_1, \ldots, x_n)$ nazywamy *dziedziną* kontekstu Γ.

(2) Kontekst Γ' nazywamy *podkontekstem* Γ i piszemy $\Gamma' \subseteq \Gamma$, jeśli wszystkie deklaracje występujące w Γ występują również w Γ z zachowaniem tego samego porządku.

(3) Kontekst Γ' nazywamy *permutacją* kontekstu Γ, jeśli wszystkie deklaracje w Γ' występują w Γ i odwrotnie.

(4) Jeśli Γ jest kontekstem i Φ jest zbiorem λ-zmiennych, wówczas *projekcją* Γ na Φ (symbolicznie $\Gamma \upharpoonright \Phi$) nazywamy podkontekst Γ' kontekstu Γ taki, że $\mathrm{dom}\Gamma' = (\mathrm{dom}\Gamma) \cap \Phi$

(5) Dla kontekstów $\Gamma = (x_1 : \sigma_1, \ldots, x_k : \sigma_m)$ i $\Gamma' = (y_1 : \rho_1, \ldots y_n : \rho_n)$ takich, że $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma') = \varnothing$ *konkatenacją* Γ i Γ' nazywamy kontekst

$$\Gamma + \Gamma' = (x_1 : \sigma_1, \ldots, x_k : \sigma_m, y_1 : \rho_1, \ldots y_n : \rho_n).$$

Przykład 10. Niech $\Gamma \equiv (y : \sigma, x_1 : \rho_1, x_2 : \rho_2, z : \tau, x_3 : \rho_3)$. Wówczas:

(1) $\mathrm{dom}\,\Gamma = (y, x_1, x_2, z, x_3)$.

(2) $\varnothing \subseteq (x_1 : \rho_1, z : \tau) \subseteq \Gamma$

(3) $(x_1 : \rho_1, x_2 : \rho_2, x_3 : \rho_3, y : \sigma, z : \tau)$ jest permutacją Γ.

(4) $\Gamma \upharpoonright \{z, u, x_1\} = (x_1 : \rho_1, z : \tau)$.

Wprowadzamy następujące reguły wyprowadzania typu:

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\ (\mathrm{var}), \qquad \frac{\Gamma \vdash M : \varphi \to \psi \qquad \Gamma \vdash N : \varphi}{\Gamma \vdash (MN) : \psi}\ (\mathrm{app}),$$

$$\frac{\Gamma, x : \varphi \vdash M : \psi}{\Gamma \vdash (\lambda x. M) : \varphi \to \psi}\ (\mathrm{abs}).$$

Definicja 22. (Typowalność)
Mówimy, że λ-term M jest typu σ w kontekście Γ (jest *typowalny*), jeśli istnieje skończone drzewo sądów spełniające poniższe warunki:

(D1) W korzeniu drzewa znajduje się sąd $\Gamma \vdash M : \sigma$.

(D2) Liście są *aksjomatami*, czyli sądami postaci $\Gamma, x : \sigma \vdash x : \sigma$.

(D3) Każdego rodzica można otrzymać z jego dzieci przez zastosowanie którejś z reguł wyprowadzania typu.

Tak określony obiekt będziemy nazywali *wyprowadzeniem* typu dla M (w kontekście Γ) i pisali $\Gamma \vdash_{\mathbb{T}} M : \sigma$. O sądzie $\Gamma \vdash M : \sigma$ będziemy wówczas mówili, że jest *wyprowadzalny.*

Definicja 23. (Poprawność) λ-term $M \in \mathbf{\Lambda}$ nazywamy *poprawnym* (ang. *legal*), jeśli istnieje wyprowadzenie $\Gamma \vdash M : \rho$ dla pewnego kontekstu Γ i typu $\rho \in \mathbb{T}$.

Lemat 3. *(O podtermie) Podterm poprawnego λ-termu jest poprawny.*

Dowód. Załóżmy, że sąd $J : \Gamma \vdash M : \sigma$ jest wyprowadzalny. Dowód przebiega przez indukcję wględem długosci wyprowadzenia J. Rozważmy następujące przypadki:

(a) Jeśli J jest konsekwencją reguły *var*, to $\mathrm{Sub}(M) = \{M\}$ (Definicja 2.1), a zatem teza jest trywialnie spełniona.

(b) Jesli J jest konsekwencją reguły *app*, to $M \equiv PQ$ dla P, Q dla których twierdzenie zachodzi. Ponieważ $\mathrm{Sub}(M) = \mathrm{Sub}(P) \cup \mathrm{Sub}(Q) \cup \{PQ\}$ (Definicja 2.2), to teza również zachodzi.

(c) Jeśli J jest konsekwencją reguły *abs*, to $M \equiv \lambda x.\, P$ dla pewnego P dla którego twierdzenie zachodzi. Ponieważ $\mathrm{Sub}(\lambda x.\, M) = \mathrm{Sub}(M) \cup \{\lambda x.\, M\}$, (Definicja 2.3) to teza zachodzi również w tym przypadku.

$\square$

Lemat 4. *(O zmiennych wolnych) Jeśli sąd $J : \Gamma \vdash L : \sigma$ jest wyprowadzalny, to* $\mathrm{FV}(L) \subseteq \mathrm{dom}\,\Gamma$.

Dowód. Prosty dowód przeprowadzamy przez indukcję względem długości wyprowadzenia sądu J. Rozważmy następujące przypadki:

(a) Jeśli J jest konsekwencą reguły *var*, to $L \equiv x$ dla pewnej λ-zmiennej x. Wobec tego $x : \sigma \in \Gamma$, a zatem $\mathrm{FV}(x) \subseteq \mathrm{dom}\,\Gamma$.

(b) Jesli J jest konsekwencją reguły *app*, to J musi mieć postać $\Gamma \vdash MN : \sigma$. Z założenia indukcyjnego: $\mathrm{FV}(M) \subseteq \mathrm{dom}\,\Gamma$ i $\mathrm{FV}(N) \subseteq \mathrm{dom}\,\Gamma$. Z Definicji 3: $\mathrm{FV}(MN) = \mathrm{FV}(M) \cup \mathrm{FV}(N)$. Stąd $\mathrm{FV}(MN) \subseteq \mathrm{dom}\,\Gamma$.

(c) Jesli J jest konsekwencją reguły *abs*, to J musi mieć postać $\Gamma \vdash \lambda x.\, M : \sigma$. Z założenia indukcyjnego $FV(M) \subseteq \mathrm{dom}\,\Gamma$. Ponieważ $\mathrm{FV}(\lambda x.\, M) = \mathrm{FV}(M) \setminus \{x\} \subseteq \mathrm{FV}(M)$ (z Definicji 3), to $\mathrm{FV}(M) \subseteq \mathrm{dom}\,\Gamma$.

$\square$

Lemat 5. *(1) Niech Γ' i Γ'' bedą kontekstami takimi, że $\Gamma' \subseteq \Gamma''$. Jeśli $\Gamma' \vdash M : \sigma$, to $\Gamma'' \vdash M : \sigma$.*

(2) Jeśli $\Gamma \vdash M : \sigma$, to $\Gamma \restriction \mathrm{FV}(M) \vdash M : \sigma$.

(3) Jeśli $\Gamma \vdash M : \sigma$ i Γ' jest permutacją Γ, to $\Gamma' \vdash M : \sigma$.

Dowód. Dowody przebiegają przez indukcję względem długości wyprowadzenia. Czytelnika zainteresowanego szczegółami odsyłamy do [Bar92, Tw. 3.1.7]. □

Lemat 6. *(O generowaniu)*

(1) Jeśli $\Gamma \vdash_{\mathbb{T}} x : \sigma$, to $x : \sigma \in \Gamma$.

(2) Jeśli $\Gamma \vdash_{\mathbb{T}} MN : \tau$, to $\Gamma \vdash_{\mathbb{T}} M : \sigma \to \tau$ i $\Gamma \vdash_{\mathbb{T}} N : \sigma$ dla pewnego $\sigma \in \mathbb{T}$.

(3) Jeśli $\Gamma \vdash_{\mathbb{T}} \lambda x. M : \tau$ i $x \notin \mathrm{dom}\,\Gamma$, to $\tau \equiv \tau_1 \to \tau_2$ oraz $\Gamma, x : \tau_1 \vdash_{\mathbb{T}} N : \tau_2$.

Dowód. (1)

(2,3) Analogicznie do (1).

□

Lemat 7. *(O podstawieniu) Załóżmy, że*

(a) $\Gamma, x : \sigma \vdash M : \rho$

(b) $\Gamma \vdash N : \sigma$

Wówczas $\Gamma \vdash M[x/N] : \rho$.

Dowód. Dowód przeprowadzimy przez indukcję względem długości wyprowadzenia $\Gamma, x : \sigma \vdash M : \rho$. Rozważmy następujące przypadki:

(i) Jeśli $\Gamma, x : \sigma \vdash M : \rho$ jest konsekwencją reguły *var*, to $M \equiv x$. Wówczas $M[x/N] \equiv N$ i $\rho \equiv \sigma$. Teza zachodzi w oczywisty sposób.

(ii) $\Gamma, x : \sigma \vdash M : \rho$ jest konsekwencją reguły *app*. Wówczas $M \equiv PQ$ i istnieją wyprowadzenia $\Gamma, x : \sigma \vdash P : \tau \to \rho$ oraz $\Gamma, x : \sigma \vdash Q : \tau$. Z założenia indykcyjnego mamy, że $\Gamma \vdash P[x/N] : \tau \to \rho$ oraz $\Gamma \vdash Q[x/N] : \tau$. Wówczas stosując regułę *app* mamy:

$$\frac{\Gamma \vdash P[x/N] : \tau \to \rho \qquad \Gamma \vdash Q[x/N] : \tau}{\Gamma \vdash (P[x/N]Q[x/N]) : \rho}\ \text{(app)},$$

Tezę otrzymujemy z faktu, że $(PQ)[x/N] = P[x/N]Q[x/N]$.

(iii) Jeśli $\Gamma, x : \sigma \vdash M : \rho$ jest konsekwencją reguły *abs*, to $M \equiv \lambda y. P : \rho$ dla $\rho \equiv \sigma \to \tau$, $y \not\equiv x$. Z założenia indukcyjnego istnieje wyprowadzenie $\Gamma', y : \tau \vdash P[x/N] : \rho$, gdzie $\Gamma' = \Gamma + (x : \sigma)$. Wówczas, stosując regułę *abs* mamy:

$$\frac{\Gamma', y : \tau \vdash P[x/N] : \rho}{\Gamma' \vdash (\lambda y.\, P[x/N]) : \tau \to \rho} \text{ (abs)}$$

Ponieważ $(\lambda y.\, P)[x/N] = \lambda y.\, P[x/N]$ oraz $M \equiv \lambda y.\, P : \tau \to \rho$, otrzymujemy tezę.

□

Lemat 8. *(Redukcja podmiotu) Załóżmy, że*

(i) $\Gamma \vdash_{\mathbb{T}} M : \sigma$

(ii) $M \to_\beta^* N$

Wówczas $\Gamma \vdash_{\mathbb{T}} N : \sigma$*.*

Dowód. Pokażemy, że twierdzenie zachodzi dla jednego kroku redukcji $\to_\beta$. Dowód zwrotności jest trywialny, zaś aby pokazać przechodniość wystarczy skorzystać z indukcji wzgledem długości ciągu redukcji.

Niech $M \to_\beta N$. Dowód przeprowadzimy przez indukcję względem długości wyprowadzenia $\Gamma \vdash M : \sigma$. Rozważmy następujące przypadki:

(a) $\Gamma \vdash M : \sigma$ jest konsekwencją reguły *var*. Wówczas $M \equiv x$ dla pewnej λ-zmiennej $x \in V$. Wówczas poprzednik nie jest spełniony, bowiem M nie da się zredukować. Zatem twierdzenie trywialnie zachodzi.

(b) $\Gamma \vdash M : \sigma$ jest konsekwencją reguły *app*. Wówczas $M \equiv PQ$ oraz istnieją wyprowadzenia $\Gamma \vdash P : \tau \to \sigma$ oraz $\Gamma \vdash Q : \sigma$. Ponadto zakładamy, że dla pewnych P', $Q' \in \mathbf{\Lambda}$ mamy $P \to_\beta P'$ i $Q \to_\beta Q'$. Istnieją dwie możliwości redukcji $M \to_\beta N$:

  (1) $N \equiv PQ'$. Poniważ $\Gamma \vdash Q' : \sigma$ (założenie indukcyjne), to możemy zastosować regułę *app*:

$$\frac{\Gamma \vdash P : \tau \to \sigma \qquad \Gamma \vdash Q' : \tau}{\Gamma \vdash PQ' : \sigma} \text{ (app)},$$

  Ponieważ $N \equiv PQ'$, to otrzymujemy tezę.

  (2) $N \equiv P'Q$. Postępujemy analogicznie do przypadku (1)

(c) $\Gamma \vdash M : \sigma$ jest konsekwencją reguły *abs*. Wówczas $M \equiv \lambda x.\, P$, dla pewnych $\rho, \tau \in \mathbb{T}$ mamy $\sigma \equiv \rho \to \tau$ oraz istnieje wyprowadzenie sądu Γ, $x : \rho \vdash P : \tau$. Ponadto zakładamy, że dla pewnego $P' \in \mathbf{\Lambda}$ mamy $P \to_\beta P'$. β-redukcja $M \to_\beta N$ musi prowadzić w tym wypadku do $N \equiv \lambda x.\, P'$. Ponieważ $\Gamma, x : \rho \vdash P' : \tau$ (założenie indukcyjne), to możemy zastosowac regułę *abs*:

$$\frac{\Gamma', x : \rho \vdash P' : \tau}{\Gamma \vdash \lambda x.\, P' : \rho \to \tau} \text{ (abs)}$$

□

2.2 Silna normalizacja

Lemat 9. *Niech $\tau \in \mathbb{T}$ bedzie dowolnym typem prostym. Wówczas:*

(1) $[\![\tau]\!] \subseteq \mathrm{SN}$.

(2) Jeśli $N_1, N_2, \ldots, N_k \in \mathrm{SN}$, *to* $xN_1N_2\ldots N_k \in [\![\tau]\!]$.

Dowód. Dowód przeprowadzimy przez indukcję strukturalną względem τ. Mamy do rozważenia następujące dwa przypadki:

(a) τ jest zmienną typową.

(1) Wynika bezpośrednio z definicji $[\![\tau]\!] \in \mathrm{SN}$.

(2) Niech $N_1, N_2, \ldots, N_k \in \mathrm{SN}$. Wówczas $N_1, N_2, \ldots, N_k \in \mathrm{SN}$. Z definicji $[\![\tau]\!]$ mamy, że $xN_1N_2\ldots N_k \in [\![\tau]\!]$.

(b) Przypuśćmy, że $\tau = \sigma \to \rho$ oraz twierdzenie zachodzi dla σ i ρ.

(1) Niech $M \in [\![\sigma \to \rho]\!]$ i niech x bedzie dowolną λ-zmienną. Z części (2) założenia indukcyjnego mamy $x \in [\![\sigma]\!]$, zatem z definicji $[\![\sigma \to \rho]\!]$ mamy $Mx \in [\![p]\!]$. Ponieważ z części (1) założenia indukcyjnego $[\![\rho]\!] \in \mathrm{SN}$, to $Mx \in \mathrm{SN}$ i w konsekwencji $[\![\sigma \to \rho]\!] \subseteq \mathrm{SN}$.

(2) Niech $P \in [\![\sigma]\!]$. Wówczas z części (1) założenia indukcyjnego $P \in \mathrm{SN}$. Chcemy pokazać, że $xN_1N_2\ldots N_k \in [\![\rho]\!]$. Z części (2) założenia indukcyjnego

$$xN_1N_2\ldots N_kN_{k+1} \in [\![\rho]\!].$$

Ustalając $N_{k+1} \equiv P$ otrzymujemy tezę.

□

Lemat 10. *Załóżmy, że:*

(a) $M[x/N_0]N_1\ldots N_k \in \mathrm{SN}$,

(b) $N_0 \in \mathrm{SN}$.

Wówczas $(\lambda x.\, M)N_0N_1\ldots N_k \in \mathrm{SN}$.

Dowód. (Ad absurdum) Przypuśćmy, że $P_0 \equiv (\lambda x.\, M)N_0N_1\ldots N_k \notin \mathrm{SN}$. Wówczas istnieje nieskończony ciąg redukcji

$$P_0 \to P_1 \to \ldots$$

Każdy podterm λ-termu silnie normalizowalnego jest silnie normalizowalny. Ponieważ $P_0 \equiv M[x/N_0]N_0N_1\ldots N_k \in \mathrm{SN}$, to $M[x/N_0]$, $N_0, N_1, \ldots, N_k \in \mathrm{SN}$. Na podstawie Lematu 2 mamy ponadto, że $M \in \mathrm{SN}$. Wobec tego dla pewnego $n \in \mathbb{N}$ redukcji ulega redeks czołowy:

$$P_n \equiv (\lambda x.\, M')N_0'N_1' \dots N_k' \to_\beta M'[x/N_0']N_0'N_1' \dots N_k' \equiv P_{n+1},$$

gdzie $M \to_\beta^* M'$ oraz $N_i \to_\beta^* N_i'$ dla $i \leq k$. Ale skoro tak, to prawdą jest również, że $M[x/N_0]N_1 \dots N_k \to_\beta^* P_{n+1}$, zaś $M[x/N_0]N_1 \dots N_k \in \mathrm{SN}$. Zatem $P_{n+1} \in \mathrm{SN}$, co prowadzi do sprzeczonści. □

Lemat 11. *Załóżmy, że:*

(a) $M[x/N_0]N_1 \dots N_k \in [\![\tau]\!]$,

(b) $N_0 \in \mathrm{SN}$.

Wówczas $(\lambda x.\, M)N_0N_1 \dots N_k \in [\![\tau]\!]$.

Dowód. Dowód przebiega przez indukcję strukturalną względem τ. Rozważmy następujące przypadki:

(a) Jeśli τ jest zmienną typową, to $[\![\tau]\!] = \mathrm{SN}$. Wobec tego problem sprowadza się do Lematu 10.

(b) Przypuśćmy, że $\tau \equiv \sigma \to \rho$ i niech $M[x/N_0]N_1 \dots N_k \in [\![\sigma \to \rho]\!]$. Wybierzmy dowolny $P \in [\![\sigma]\!]$. Wówczas $M[x/N_0]N_1 \dots N_kN_{k+1} \in [\![\rho]\!]$. Z założenia indukcyjnego mamy jednak, że $(\lambda x.\, M)N_0N_1 \dots N_kN_{k+1} \in [\![\rho]\!]$. Wystarczy więc przyjąć $N_{k+1} \equiv P$ i z definicji $[\![\sigma \to \rho]\!]$ mamy, że $(\lambda x.\, M)N_0N_1 \dots N_k \in [\![\sigma \to \rho]\!]$.

□

Definicja 24. Powiemy, że kontekst $\Gamma = \{x_1 : \sigma_1, x_2 : \sigma_2, \dots, x_n : \sigma_n\}$ *spełnia* stwierdzenie $M : \sigma$ i będziemy pisali $\Gamma \vDash M : \sigma$, jeśli dla dowolnych $N_1 \in [\![\sigma_1]\!]$, $N_2 \in [\![\sigma_2]\!]$, …, $N_n \in [\![\sigma_n]\!]$ mamy, że:

$$M[x_1/N_1,\, x_2/N_2,\, \dots,\, x_n/N_n] \in [\![\tau]\!].$$

Lemat 12. *Jeśli* $\Gamma \vdash_{\mathbb{T}} M : \tau$, *to* $\Gamma \vDash M : \tau$.

Dowód. Dowód będzie przebiegał przez indukcję względem wyprowadzenia $\Gamma \vdash M : \tau$. Niech $\Gamma = (x_1 : \tau_1, x_2 : \tau_2, \dots, x_n : \tau_n)$ będzie kontekstem dla którego istnieje wyprowadzenie $J : \Gamma \vdash M : \tau$. Wybierzmy $N_1 \in [\![\tau_1]\!]$, $N_2 \in [\![\tau_2]\!]$, …, $N_n \in [\![\tau_n]\!]$. Rozważmy następujące przypadki:

(a) J jest konsekwencją reguły *var*. Wówczas J jest postaci $\Gamma \vdash x_i : \tau$ dla pewnego $i \in \mathbb{N}$, $1 \leq i \leq n$, gdzie $x_i : \tau \in \Gamma$. Stąd $M[\vec{x}/\vec{N}] = x_i[x_i/N_i] = N_i \in [\![\tau]\!]$. Z dowolności N_i, $\Gamma \vDash M : \tau$.

(b) J jest konsekwencją reguły *app*. Wówczas J jest postaci $\Gamma \vdash PQ : \tau$. Z założenia indukcyjnego istnieje $\sigma \in \mathbb{T}$ takie, że $\Gamma \vDash P : \sigma \to \tau$ i $\Gamma \vDash Q : \sigma$. Wobec tego $P[\vec{x}/\vec{N}] \in [\![\sigma \to \tau]\!]$ i $Q[\![\vec{x}/\vec{N}]\!] \in [\![\sigma]\!]$. Z definicji jednoczesnego podstawienia (Definicja 10) mamy:

$$PQ[\vec{x}/\vec{N}] = P[\vec{x}/\vec{N}]Q[\vec{x}/\vec{N}]$$

Z definicji $[\![\sigma \to \tau]\!]$ wówczas $M \in [\![\tau]\!]$.

(c) J jest konsekwencją reguły *abs*. Wówczas J jest postaci $\Gamma \vdash \lambda y.\, P : \sigma \to \rho$, gdzie $y \notin \mathrm{dom}\Gamma$. Z założenia indukcyjnego mamy, że $\Gamma, y : \sigma \vDash P : \rho$. Oznacza to, że dla dowolnych $N_1 \in [\![\tau_1]\!]$, $N_2 \in [\![\tau_2]\!]$, …, $N_n \in [\![\tau_n]\!]$ mamy

$$\forall N \in [\![\sigma]\!] \ \big(P[\vec{x}, y/\vec{N}, N] \in [\![\rho]\!]\big) \qquad (*)$$

Ustalmy $P' \equiv P[y/y'][\vec{x}/\vec{N}]$, gdzie $y' \notin \mathrm{dom}\Gamma$ i $y' \notin \mathrm{FV}(N_i)$ dla $i \in \mathbb{N}$, $1 \leq i \leq n$. Wówczas z $(*)$:

$$\forall N \in [\![\sigma]\!] \ (P'[y'/N] \in [\![\rho]\!])$$

Ustalmy $N_0 \in [\![\sigma]\!]$. Wówczas z cześci (1) Lematu 9 $N_0 \in \mathrm{SN}$. Wobec tego z Lematu 11 wnioskujemy, że:

$$(\lambda y'.\, P')N_0 \in [\![\rho]\!] \qquad (**)$$

Zauważmy teraz, że ponieważ $\forall i \ y_i \notin \mathrm{FV}(N_i)$

$$\begin{aligned}(\lambda y'.\, P') &= (\lambda y'.\, P[y/y'][\vec{x}/\vec{N}]) \\ &= (\lambda y'.\, P[y/y'])[\vec{x}/\vec{N}] = (\lambda y.\, P)[\vec{x}/\vec{N}]\end{aligned} \qquad (***)$$

Z $(**)$ i $(***)$ otrzymujemy

$$\big((\lambda y.\, P)[\vec{x}/\vec{N}]\big)\, N_0 \in [\![\rho]\!].$$

Ponieważ $N_0 \in [\![\sigma]\!]$, to z definicji $[\![\sigma \to \rho]\!]$ mamy, że

$$(\lambda y.\, P)[\vec{x}/\vec{N}] \in [\![\sigma \to \rho]\!].$$

Z dowolności $\vec{N}$ otrzymujemy ostatecznie, że $\Gamma \vDash \lambda y.\, P$.

□

Twierdzenie 3. *(O silnej normalizacji) Jeżeli* $\Gamma \vdash_{\mathbb{T}} M : \tau$, *to* $M \in \mathrm{SN}_\beta$.

Dowód. Na podstawie Lematu 12, jeśli $\Gamma \vdash_{\mathbb{T}} M : \tau$, to $M \in [\![\tau[\![$. Stosując Lemat 9 otrzymujemy tezę. □

Literatura

[Alt02] Thorsten Altenkirch. "α-conversion is easy". Under Revision. 2002. URL: https://www.cs.nott.ac.uk/~psztxa/publ/alpha-draft.pdf.

[Bar92] Henk (Hendrik Barendregt. "Lambda Calculi with Types". In: vol. 2. Jan. 1992, pp. 117–309. ISBN: 0198537611.

[Bru72] N.G. de Bruijn. "Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem". In: *Indagationes Mathematicae (Proceedings)* 75 (Dec. 1972), pp. 381–392. DOI: 10.1016/1385-7258(72)90034-0.

[Chu41] Alonzo Church. *The Calculi of Lambda-Conversion*. Princeton University Press, 1941.

[HIN05] RALF HINZE. "THEORETICAL PEARL Church numerals, twice!" In: *Journal of Functional Programming* 15.1 (2005), pp. 1–13. DOI: 10.1017/S0956796804005313.

[HS08] J. Roger Hindley and Jonathan P. Seldin. *Lambda-Calculus and Combinators: An Introduction*. 2nd ed. New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521898854, 9780521898850.

[Hut99] Graham Hutton. "A Tutorial on the Universality and Expressiveness of Fold". In: *J. Funct. Program.* 9.4 (July 1999), pp. 355–372. ISSN: 0956-7968. DOI: 10.1017/S0956796899003500. URL: http://dx.doi.org/10.1017/S0956796899003500.

[Jan13] Jan Martin Jansen. "Programming in the λ-Calculus: From Church to Scott and Back". In: *Essays Dedicated to Rinus Plasmeijer on the Occasion of His 61st Birthday on The Beauty of Functional Code - Volume 8106*. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 168–180. ISBN: 978-3-642-40354-5. DOI: 10.1007/978-3-642-40355-2_12. URL: https://doi.org/10.1007/978-3-642-40355-2_12.

[JKP06] Jan Martin Jansen, Pieter Koopman, and Rinus Plasmeijer. "Efficient Interpretation by Transforming Data Types and Patterns to Functions". In: Jan. 2006, pp. 73–90.

[KPJ14] Pieter Koopman, Rinus Plasmeijer, and Jan Martin Jansen. "Church Encoding of Data Types Considered Harmful for Implementations: Functional Pearl". In: *Proceedings of the 26Nd 2014 International Symposium on Implementation and Application of Functional Languages*. IFL '14. Boston, MA, USA: ACM, 2014, 4:1–4:12. ISBN: 978-1-4503-3284-2. DOI: 10.1145/2746325.2746330. URL: http://doi.acm.org/10.1145/2746325.2746330.

[PL92] Simon L. Peyton Jones and David R. Lester. *Implementing Functional Languages*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992. ISBN: 0-13-721952-0.

[SS75] Gerald J. Sussman and Guy L. Steele Jr. *An Interpreter for Extended Lambda Calculus*. Tech. rep. Cambridge, MA, USA, 1975.

[SU06] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism, Volume 149 (Studies in Logic and the Foundations of Mathematics)*. New York, NY, USA: Elsevier Science Inc., 2006. ISBN: 0444520775.