

1 Rachunek λ

Niech V będzie przeliczalnie nieskończonym zbiorem zmiennych przedmiotowych x, y, \dots (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 $\tilde{\Lambda}$ pretermów) Zbiorem pretermów będziemy nazywali najmniejszy (w sensie mnogościowym) zbiór wyrażeń $\tilde{\Lambda}$ taki, że:

- (P1) Jeśli $x \in V$, to $x \in \tilde{\Lambda}$.
- (P2) Jeśli $M, N \in \tilde{\Lambda}$, to $(M N) \in \tilde{\Lambda}$.
- (P3) Jeśli $x \in V$ i $M \in \tilde{\Lambda}$, to $(\lambda x. M) \in \tilde{\Lambda}$.

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

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

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

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

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

- (P1): x, y, z .
- (P2): $xx, yx, x(xz),$
 $(\lambda x. (xz))y, y(\lambda x. (xz)), (\lambda x. x)(\lambda x. x)$.

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

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

Definicja 2. (Multizbiór Sub podtermów pretermu)

- (1) $\text{Sub}(x) = \{x\}$
- (2) $\text{Sub}(MN) = \text{Sub}(M) \cup \text{Sub}(N) \cup \{M N\}$
- (3) $\text{Sub}(\lambda x. M) = \text{Sub}(M) \cup \{\lambda x. M\}$

Elementy multizbioru $\text{Sub}(M)$ nazywamy *podtermami* M . Jeśli L jest podtermem M , ale $L \neq M$, to L nazywamy podtermem *właściwym*.

Przykład 2. Podtermy wybranych λ -pretermów.

- (a) $\text{Sub}(\lambda x. x x) = \{(\lambda x. x x)^1, (x x)^1, x^2\}$
- (b) $\text{Sub}((\lambda x. x x) (\lambda x. x x)) =$
 $= \{((\lambda x. x x) (\lambda x. x x))^1, (\lambda x. x x)^2, (x x)^2, x^4\}$

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 $\text{FV}(M)$ *zmiennych wolnych* w M w następujący sposób:

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

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

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

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

(c) $\text{FV}(\lambda x y z. x y) = \emptyset$

Definicja 4. (Podstawienie) Dla dowolnych $M, N \in \tilde{\Lambda}$ 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 $x \neq y$
(S3) $(PQ)[x/N] = P[x/N]Q[x/N]$
(S4) $(\lambda y. P)[x/N] = \lambda y. P[x/N]$, gdzie $x \neq y$ i $y \notin \text{FV}(N)$
(S5) $(\lambda x. P)[x/N] = \lambda x. P$

Lemat 1. (O podstawieniu) Niech $M, N, L \in \tilde{\Lambda}$ i niech ponadto $x \neq y$ oraz $x \notin \text{FV}(L)$. Wówczas

$$M[x/N][y/L] \equiv M[y/L][x/N[y/L]]. \quad (1)$$

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 eostaci $N[y/L]$.
 - b. Jeśli $M \equiv y$, to ponieważ $x \neq y$ i $x \notin \text{FV}(M)$, po wykonaniu podstawienia po lewej stronie (1) otrzymujemy $M[x/N][y/L] \equiv L$. Ponieważ $x \notin \text{FV}(L)$, to po wykonaniu podstawienia po prawej stronie widzimy, że obydwie strony są identyczne.
 - c. Jeśli $M \equiv z$ i $z \neq x$ oraz $z \neq y$, to obydwie strony (1) są identyczne.
- ii) $M \equiv PQ$ dla pewnych $P, Q \in \tilde{\Lambda}$. 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) są identyczne. Przypuśćmy, że $z \neq x$ i $z \neq y$ i $z \notin \text{FV}(L)$. 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. Jeśli $M[x/y]$ jest określone i $y \notin \text{FV}(M)$, 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 $\tilde{\Lambda}$ taki, że

- ($\alpha 1$) Jeśli $y \notin \text{FV}(M)$ oraz $M[x/y]$ jest określone,
to $\lambda x. M =_\alpha \lambda y. M[x/y]$
- ($\alpha 2$) Jeśli $M =_\alpha N$, to dla dowolnego $x \in V$ zachodzi $\lambda x. M =_\alpha \lambda x. N$
- ($\alpha 3$) Jeśli $M =_\alpha N$, to dla dowolnego $Z \in \tilde{\Lambda}$ zachodzi $MZ =_\alpha NZ$
- ($\alpha 4$) Jeśli $M =_\alpha N$, to dla dowolnego $Z \in \tilde{\Lambda}$ 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 $L \in \tilde{\Lambda}$ 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 ($\alpha 1$) dla $M \equiv \lambda x. M'$ i $N \equiv \lambda y. M'[x/y]$. Ponieważ $x \notin \text{FV}(M'[x/y])$, to ze względu na Wniosek 1 mamy, że $M'[x/y][y/x] = M'$. Zatem, na podstawie ($\alpha 1$):

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

- iv) Jeśli $M =_\alpha N$ w konsekwencji ($\alpha 2$), gdzie $M = \lambda x. M'$ i $N = \lambda x. N'$ dla $M' =_\alpha N'$, to z hipotezy indukcyjnej $N' =_\alpha M'$ i w konsekwencji ($\alpha 2$) 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 Λ λ -termów) Każdą klasę abstrakcji relacji $=_{\alpha}$ nazywamy λ -termem. Zbiór wszystkich λ -termów Λ to zbiór ilorazowy relacji α -konwersji:

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

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 $M \in \Lambda$ 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 $y \in V$
- (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 następujące 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 albo zmienną (przypadek (1)) albo aplikacją (przypadek (2)).

Na zbiór Λ przenoszą się pojęcia podtermu, zmiennych wolnych i operacji podstawienia definiowane uprzednio dla pretermów.

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

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

gdzie $\text{Sub}(M')$ 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 $\text{FV}(M)$ *zmiennych wolnych* w M

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

gdzie $\text{FV}(M')$ 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 występujących zmiennych:

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

w przypadku, gdy $x \neq y$, gdzie $x' \notin \text{FV}(M)$ i $x' \notin \text{FV}(N)$. Rozstrzygnięcie takie przytacza się w [HS08]. Po uwzględnieniu odpowiednich modyfikacji, Definicja 4 przyjmuje następującą postać:

Definicja 4'. (*Podstawienie'*)

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

$$(S'2) \quad y[x/N] = y, \text{ o ile } x \neq y$$

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

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

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

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

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

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 $y \neq x$.

Uwaga 1. Każde podstawienie $[x/N]$ jest funkcją z $\mathbf{\Lambda} \rightarrow \mathbf{\Lambda}$, 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 Bruijna 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:

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

Historycznie wiąże się ta notacja z jego pracami nad systemem komputerowo wspomagane 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 -termów 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 $i \in \mathbb{N}$, $y \neq x_i$.
- ($\vec{s}3$) $(PQ)[\vec{x}/\vec{N}] = P[\vec{x}/\vec{N}]Q[\vec{x}/\vec{N}]$
- ($\vec{s}4$) $(\lambda y. P)[\vec{x}/\vec{N}] = \lambda y. P[\vec{x}/\vec{N}]$, 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 $i_1, i_2, \dots, i_n \in \mathbb{N}$, to $[x_{i_1}/N_{i_1}, x_{i_2}/N_{i_2}, \dots, x_{i_n}/N_{i_n}] \equiv [\vec{x}/\vec{N}]$.

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

$$\begin{array}{ll} \text{a)} & (xy)[y/x][x/u] = uu, \\ & (xy)[y/x, x/u] = ux. \\ \text{b)} & (\lambda x. yx)[x/y][y/z] = \lambda x. zx, \\ & (\lambda x. yx)[x/y, y/z] = \lambda x. zx. \end{array}$$

1.2 Redukcja

Sens obliczeniowy λ -termom nadajemy przez określenie na Λ operacji β - i η -redukcji. Pożądane jest, żeby operacje te wykonywane na podtermach pozostawały w zgodzie ze strukturą całego λ -termu.

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

- (c1) Jeśli $M\mathcal{R}N$, to $(\lambda x. M)\mathcal{R}(\lambda x. N)$ dla dowolnej λ -zmiennnej 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 Λ można rozumieć jako graf skierowany (w przypadku Λ 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 Λ ze względu na rezultat obliczenia.

Definicja 12. Niech \rightarrow będzie relacją binarną w zbiorze A .

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

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

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

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



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

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

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

Definicja 14. Niech \rightarrow będzie relacją binarną na zbiorze A .

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

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

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

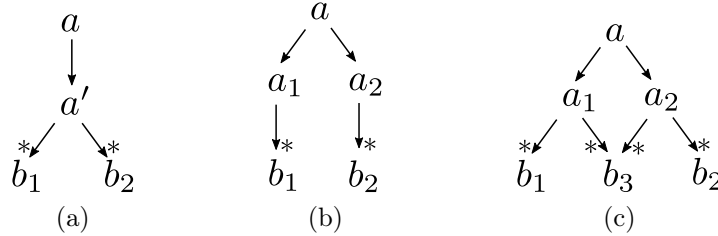
Dowód. Niech \rightarrow będzie relacją binarną na A o własności SN i WCR. Ponieważ \rightarrow jest SN, to każdy a jest normalizowalny.

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 $a \rightarrow a'$ i a' jest niejednoznaczny. Niech $b_1, b_2 \in \text{NF}$, $b_1 \neq b_2$ i $a \rightarrow^* b_1$ oraz $a \rightarrow^* b_2$. Ponieważ $b_1 \neq b_2$, to istnieją $a_1, a_2 \in A$ takie, że:

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

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 $a_1 \rightarrow^* b_3$ oraz $a_2 \rightarrow^* b_3$. 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 $a_1 \rightarrow^* b_3$ oraz $a_2 \rightarrow^* b_3$ (Rysunek 2c). Przypuśćmy, że $b_3 \in \text{NF}$. 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 $a' = a_2$.



Rysunek 2: Warianty konstruowania redukcji.

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

Definicja 15. (β -redukcja) β -redukcją nazywamy najmniejszą (w sensie mnogościowym) zgodną na Λ relację binarną \rightarrow_β taką, że

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

β -redexami będziemy nazywali wyrażenia postaci $(\lambda x. M)N$, zaś rezultat ich β -redukcji w postaci termu $M[x/N]$ – β -reduktem. Przez \rightarrow_β^+ , \rightarrow_β^* , $=_\beta$ oznaczamy odpowiednie domknięcia relacji β -redukcji. Symbolem \leftarrow_β oznaczać będziemy relację odwrotną do β -redukcji, zaś przez \leftrightarrow_β jej symetryczne domknięcie.

Ciągiem β -redukcji nazywamy każdy skończony lub nieskończony ciąg λ -termów M_0, M_1, \dots taki, że $M_0 \rightarrow_\beta M_1 \rightarrow_\beta \dots$

Relację $=_\beta$ nazywamy β -konwersją. Zauważmy, że $M =_\beta N$ wtedy i tylko wtedy, gdy istnieje skończony ciąg λ -termów $M \equiv M_0, M_1, \dots, M_n \equiv N$ taki, że $M_i \rightarrow_\beta M_{i+1}$ lub $M_{i+1} \rightarrow_\beta M_i$ 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:

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

Definicja 16. (Strategia redukcji) Strategią redukcji nazywamy każde odwzorowanie $S: \Lambda \rightarrow \Lambda$ postaci

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

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

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

$$\begin{aligned} YF &\equiv (\lambda f. (\lambda x. (f(xx)) \lambda x. (f(xx)))) F \\ &\rightarrow_{\beta} (\lambda x. F(xx)) \lambda x. F(xx) \\ &\rightarrow_{\beta} F((\lambda x. F(xx)) \lambda x. F(xx)) \\ &\rightarrow_{\beta} F(\underbrace{F((\lambda x. F(xx)) \lambda x. F(xx))}_{=_{\beta} YF}) \\ &\rightarrow_{\beta} \dots \end{aligned}$$

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 -redexsem, którego redukcja prowadzi do ponownego otrzymania termu Ω i w konsekwencji do stałego ciągu redukcji postaci:

$$\Omega \rightarrow_{\beta} \Omega \rightarrow_{\beta} \Omega \rightarrow_{\beta} \dots$$

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

$$\Delta\Delta \rightarrow_{\beta} \Delta\Delta\Delta \rightarrow_{\beta} \Delta\Delta\Delta\Delta \rightarrow_{\beta} \dots$$

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 -równoważnych) 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

$$(\lambda u. v) \Omega \rightarrow_{\beta} (\lambda u. v) \Omega \rightarrow_{\beta} \dots$$

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 Λ relację binarną \rightarrow_{η} taką, że

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

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 optymalizuje zapis λ -termów i stąd ma duże znaczenie stylistyczne w programowaniu funkcyjnym.

Przykład 8. Przypuśćmy, że $(+1) \in \Lambda$. 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 $L \in \Lambda$ takie, że $M \rightarrow_{\beta}^* L$ i $N \rightarrow_{\beta}^* L$.

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

$$M_0 \leftrightarrow_{\beta} M_1 \leftrightarrow_{\beta} \dots \leftrightarrow_{\beta} M_{n-1} \leftrightarrow_{\beta} M_n,$$

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 $M \rightarrow_{\beta}^* L$ i $N \rightarrow_{\beta}^* L$.

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

$$M \equiv M_0 \xleftrightarrow[\beta]{} M_1 \xleftrightarrow[\beta]{} \dots \xleftrightarrow[\beta]{} M_{k-1} \xleftrightarrow[\beta]{} M_k \equiv N$$

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

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

$$\begin{array}{ccccc} M_0 & \longleftrightarrow & \dots & \longleftrightarrow & M_{k-1} & \longrightarrow & M_k \\ & \searrow^* & & \searrow^* & & & \downarrow^* \\ & & L' & \xrightarrow{*} & & & L \end{array}$$

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

□

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

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

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

(2) Przypuśćmy, że M ma dwie różne postaci normalne, N_1, N_2 . Wówczas z części (1) tego twierdzenia, $M \rightarrow_{\beta}^* N_1$ i $M \rightarrow_{\beta}^* N_2$. Z Faktu 1 istnieje $L \in \mathbf{\Lambda}$ taki, że $N_1 \rightarrow_{\beta}^* L$ i $N_2 \rightarrow_{\beta}^* L$. Ponieważ $N_1, N_2 \in \text{NF}_{\beta}$, 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 z niej problemy natury złożonościowej [KPJ14], ma obecnie wyłącznie walory edukacyjne, dlatego w dalszej części pracy pokażemy tzw. kodowanie Scotta. Jest ona interesująca ze względu na praktyczną możliwość reprezentacji algebraicznych typów danych (ADT¹) 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:

```
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 rozpoczyna się od słowa kluczowego `data`² po którym występuje *konstruktor typu*. Na wzór notacji BNF, typy przyjmują jedną z *wartości* oddzielonych 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`.

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

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

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 bezpośrednio 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 częściową 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 $x, y \in \Lambda$ do pewnego $f \in \Lambda$, 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 **fromTemperature** 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:

$$\text{add}_0 \equiv \lambda n m. n m (\lambda n. \text{Succ}(\text{add}_0 n m))$$

Formalizm rachunku λ nie pozwala na określanie 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:

$$Y \equiv \lambda f. (\lambda x. (f(xx)) \lambda x. (f(xx)))$$

Wówczas określamy

$$\text{add}_Y \equiv Y (\lambda a n m. n m (\lambda n. \text{Succ}(a n m)))$$

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**:

$$\begin{aligned} \text{Zero} &\equiv \lambda z s. z & \text{Nil} &\equiv \lambda n c. n \\ \text{Succ} &\equiv \lambda n z s. s n & \text{Cons} &\equiv \lambda x x_s n c. c x x_s \end{aligned}$$

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 $\text{Void} \equiv \lambda x. x$ jako element neutralny działania łącznego, otrzymujemy na poziomie typów strukturę półpierścienia z działaniem mnożenia określoną przez konstrukcję par i działaniem dodawania określonego przez konstrukcję typów wyliczeniowych. Stąd algebraiczne 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:

$$\begin{aligned}\text{foldl } f \ x \ \text{Zero} &= x \\ \text{foldl } f \ x \ (\text{Succ } n) &= f \ (\text{foldl } f \ x \ n)\end{aligned}$$

`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 recursion*), w teorii obliczalności nazywane rekursją prostą lub, popularnie, zwijaniem od lewej. Operator `foldl` spełnia następującą własność [Hut99]

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

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:

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

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 $\text{nat} \equiv \lambda c. c \text{ Succ Zero}$. Wówczas $\text{nat } \bar{n} =_{\beta} \bar{n}$. 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:

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

Dla porównania, używając kodowania Scotta:

$$\text{add}_S \equiv \lambda n m. \text{foldl Succ } n m$$

1.3.7 Ogólny schemat kodowania Scotta typów ADT

W ogólnym przypadku, mając następującą definicję ADT:

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

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

$$\begin{aligned} C_1 &\equiv \lambda t_{11} t_{12} \dots t_{1n_1} f_1 f_2 \dots f_m. f_1 t_{11} t_{12} \dots t_{1n_1} \\ C_2 &\equiv \lambda t_{21} t_{22} \dots t_{2n_2} f_1 f_2 \dots f_m. f_2 t_{21} t_{22} \dots t_{2n_2} \\ &\vdots \\ C_m &\equiv \lambda t_{m1} t_{m2} \dots t_{mn_m} f_1 f_m \dots f_m. f_1 t_{m1} t_{m2} \dots t_{mn_m} \end{aligned}$$

Wówczas następującą definicję częściową funkcji f :

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

kodujemy przy za pomocą następującego λ -termu:

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

gdzie y_i są kodowaniami Scotta y_i 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, \dots (być może indeksowanych liczbami naturalnymi), które będziemy nazywali *zmiennymi typowymi*.

Definicja 18. (Typy proste)

Typami prostymi będziemy określali najmniej w sensie mnogościowym zbiór wyrażeń taki, że:

(ST1) Jeśli p jest zmienną typową, to p jest typem prostym.

(ST2) Jeśli τ i σ są typami prostymi, to $(\tau \rightarrow \sigma)$ jest typem prostym.

Typy proste zbudowane tylko wedle reguły (ST1) nazywamy typami *atomowymi*, zaś wyrażenia zbudowe wedle reguły (ST2) – typami *funkcyjnymi*. Zbiór typów prostych określony w myśl powyższej definicji będziemy oznaczali przez \mathbf{T}_{\rightarrow} .

Późniejsze litery alfabetu greckiego, tj. $\sigma, \tau, \rho, \dots$ będą służyły nam za zmienne metasyntaktyczne do oznaczania typów prostych. Dla lepszej czytelności będziemy pomijali najbardziej zewnętrzne nawiasy. Konstruktor typu \rightarrow wiąże prawostronnie; oznacza to, że typy $\tau \rightarrow \sigma \rightarrow \theta$ oraz $\tau \rightarrow (\sigma \rightarrow \theta)$ będziemy uznawali za tożsame.

Typy proste ujęte Definicją 18 mają strukturę drzewa binarnego. Wysokość takiego drzewa będziemy nazywali *stopniem* typu. Precyzyjnie ujmuje to pojęcie poniższa definicja.

Definicja 19. (Stopień typu)

Stopniem typu nazywamy funkcję $\delta : \mathbf{T}_{\rightarrow} \longrightarrow \mathbb{N}$ taką, że

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

2.2 Pseudotermy

Niech V będzie przeliczalnie nieskończonym zbiorem zmiennych przedmiotowych x, y, \dots (indeksowanych być może liczbami naturalnymi). Elementy takiego zbioru będziemy nazywali *λ -zmiennymi*.

Definicja 20. (Pseudo-pretermy)

Pseudo-pretermami będziemy nazywali najmniej (w sensie mnogościowym) zbiór $\tilde{\Lambda}_T$ taki, że:

(PT1) Jeśli $x \in V$, to $x \in \tilde{\Lambda}_T$.

(PT2) Jeśli $M \in \tilde{\Lambda}_T$ i $N \in \tilde{\Lambda}_T$, to $(MN) \in \tilde{\Lambda}_T$.

(PT3) Dla dowolnych $x \in V$, $\sigma \in \mathbf{T}_\rightarrow$, $M \in \tilde{\Lambda}_T$ mamy, że $(\lambda x : \sigma. M) \in \tilde{\Lambda}_T$.

Wyrażenia postaci (PT2) nazywamy *aplikacjami* M do N , zaś wyrażenia postaci (PT3) – *λ -abstrakcjami*, gdzie o wszystkich podtermach termu M mówi się, że są w *zasięgu* λ -abstraktora, zaś o λ -zmiennej x mówi się, że jest nim *związana*.

Za zmienne metasyntaktyczne obieramy duże litery alfabetu łacińskiego M, N, \dots . Podobnie jak w podrozdziale 2.1 stosujemy konwencję o opuszczaniu najbardziej zewnętrznych nawiasów. Aplikacja termów wiąże lewostronnie; oznacza to, że będziemy utożsamiali ze sobą wyrażenia MNP oraz $(MN)P$.

Definicja 21. (Stwierdzenie, deklaracja, kontekst, sąd)

- (1) *Stwierdzeniami* (ang. *statement*) nazywamy napisy postaci $M : \sigma$, gdzie $M \in \Lambda_T$ i $\sigma \in \mathbb{T}$. W stwierdzeniu $M : \sigma$ λ -term M nazwamy *podmiotem* (ang. *subject*), zaś σ – *predykatem*.
- (2) *Deklaracjami* (ang. *declaration*) nazywamy stwierdzenia w których podmiot jest zmienną termową.
- (3) *Kontekstem* (ang. *context*) nazywamy skończony liniowo uporządkowany zbiór deklaracji, w którym wszystkie podmioty są wzajemnie różne.
- (4) *Sądami* (ang. *judgement*) nazywamy napisy postaci $\Gamma \vdash M : \sigma$, gdzie Γ jest kontekstem, zaś $M : \sigma$ – stwierdzeniem.

Definicja 22. (Poprawność) Preterm $M \in \Lambda_T$ nazywamy *poprawnym*, jeśli istnieje kontekst Γ i typ $\rho \in \mathbb{T}$ taki, że $\Gamma \vdash M : \rho$.

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.