

# 1 Rachunek $\lambda$

Niech  $V$  będzie przeliczalnie nieskończonym zbiorem zmiennych przedmiotowych  $x, y, \dots$  (indeksowanych być może liczbami naturalnymi). Elementy takiego zbioru będziemy nazywali  $\lambda$ -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 ?? można równoznacznie wyrazić przy pomocy notacji Backusa-Naura. Wówczas ma ona następującą, zwięzłą postać:

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

Elementy  $\tilde{\Lambda}$  będziemy oznaczali literami  $L, M, N, P, Q, R$  i ich wariantami z górnymi lub dolnymi indeksami. Wyrażenia postaci ?? nazywamy *aplikacjami*  $M$  do  $N$ . Symbol  $\lambda$  występujący w ?? nazywamy  $\lambda$ -abstraktorem, zaś wyrażenia powstałe przez zastosowanie tej reguły to  $\lambda$ -abstrakcje. W wyrażeniu postaci  $(\lambda x. M)$  preterm  $M$  jest w *zasięgu*  $\lambda$ -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  $PQR$ ,*
- *$\lambda$ -abstrakcja wiąże prawostronnie:  $\lambda x_1. (\lambda x_2. P)$  zapisujemy  $\lambda x_1. \lambda x_2. P$ ,*
- *następujące po sobie  $\lambda$ -abstrakcje postaci  $\lambda x_1. \lambda x_2. \dots \lambda x_n. P$  zapisujemy pod wspólnym  $\lambda$ -abstraktorem:  $\lambda x_1 x_2 \dots x_n. P$ .*

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

**Przykład 1.** Podajmy kilka przykładów  $\lambda$ -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):  $\lambda x. (xz), \lambda yz. x, \lambda x. (\lambda x. (xx))$ .

Podwyrażenia  $\lambda$ -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)  $\text{Sub}(MN) = \text{Sub}(M) \cup \text{Sub}(N) \cup \{MN\}$
- (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  $\lambda$ -pretermów.

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

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

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

(c)  $\text{FV}(\lambda xyz. xy) = \emptyset$

**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 (??) po podstawieniu są postaci  $N[y/L]$ .
  - b. Jeśli  $M \equiv y$ , to ponieważ  $x \neq y$  i  $x \notin \text{FV}(M)$ , po wykonaniu podstawienia po lewej stronie (??) 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 (??) 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 ?? 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 ?? widzimy, że obydwie strony (??) 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 ?? dowód przebiega przez indukcję strukturalną względem  $M$ . □

## 1.1 Wyrażenia $\lambda$

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  $\alpha$ -variantami lub że są ze sobą w relacji  $\alpha$ -konwersji.

**Definicja 5.** (Relacja  $\alpha$ -konwersji) Relacją  $=_\alpha$  ( $\alpha$ -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 ???. 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 ??? 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 ?? mamy, że  $M'[x/y][y/x] = M'$ . Zatem, na podstawie ???:

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

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

- v) Jeśli  $M =_{\alpha} N$  w konsekwencji ?? 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 ??  $N =_{\alpha} M$ .
- vi) Jeśli  $M =_{\alpha} N$  w konsekwencji ??, to postępujemy jak w przypadku ??.  $\square$

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

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

**Konwencja.** Wprowadzamy następujące konwencje notacyjne:

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

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

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

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

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

**Definicja 8.** (Zbiór zmiennych wolnych FV) Dla dowolnego  $\lambda$ -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 ??.

**Definicja 9.** (Podstawienie) Niech  $M = [M']_{=\alpha}$  i  $N = [N']_{=\alpha}$  i niech  $M'[x/N']$  będzie określone w myśl Definicji ??. 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 porządaną własność niezmienniczości  $\alpha$ -konwersji

względem podstawienia. Stąd w Definicji ?? 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 [Hindley:2008:LCI:1388400]. Po uwzględnieniu odpowiednich modyfikacji, Definicja ?? przyjmuje następującą postać:

**Definicja 4'.** (*Podstawienie'*)

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

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

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

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

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

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

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

przy czym w ?? wymagamy, aby zmienna  $z$  nie występowała wcześniej w termach  $N$  i  $P$  jako zmienna wolna, zaś dla ??-?? dodatkowo  $y \neq x$ .

*Uwaga 1.* Każde podstawienie  $[x/N]$  jest funkcją z  $\Lambda \rightarrow \Lambda$ , gdzie  $x \in V$  i  $N \in \Lambda$  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  $\iota$  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 [deBruijnIndices] notacja eliminuje bezpośrednie występowanie symboli zmiennych w  $\lambda$ -termach, zastępując je liczbą naturalną wyrażającą głębokość zagnieżdżenia odpowiedniej  $\lambda$ -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, eliminuje konieczność utożsamiania termów przez  $\alpha$ -konwersję, ale istotnie zmniejsza ich czytelność.

Szerszy komentarz dotyczący dotychczasowych prób uchwycenia operacji podstawienia można prześledzić w [txa:alpha-draft]. Nasze rozważania opierają się w tej materii przeważająco na [Urzyczyn2006]. Samo podejście do definiowania  $\lambda$ -termów przez operację  $\alpha$ -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  $\alpha$ -konwersji.

**Definicja 10.** (Podstawienie jednocześnie) Dla dowolnego  $M \in \mathbf{\Lambda}$ , ciągu  $\lambda$ -zmiennych  $\vec{x}$  i ciągu  $\lambda$ -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 ?? i Definicji ?? mogą, ale nie muszą, prowadzić do różnych rezultatów.

- a)  $(xy)[y/x][x/u] = uu,$                       b)  $(\lambda x. yx)[x/y][y/z] = \lambda x. zx,$   
 $(xy)[y/x, x/u] = ux.$                        $(\lambda x. yx)[x/y, y/z] = \lambda x. zx.$

## 1.2 Redukcja

Sens obliczeniowy  $\lambda$ -termom nadajemy przez określenie na  $\mathbf{\Lambda}$  operacji  $\beta$ - i  $\eta$ -redukcji. Pożądane jest, żeby operacje te wykonywane na podtermach pozostawały w zgodzie ze strukturą całego  $\lambda$ -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  $\lambda$ -zmiennnej  $x$ .
- (c2) Jeśli  $M\mathcal{R}N$ , to  $(MP)\mathcal{R}(NP)$ .

(c3) Jeśli  $MRN$ , 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  $??$  wyznacza praporzadek, który w odniesieniu do redukcji określonych na  $\Lambda$  można rozumieć jako graf skierowany (w przypadku  $\Lambda$  być może nieskończony) w którym krawędzie odpowiadają możliwym krokom obliczenia, zaś  $??$  – kongruencję, która znów w szczególnym odniesieniu do  $\lambda$ -termów, będzie dokonywała podziału w  $\Lambda$  ze względu na rezultat obliczenia.

**Definicja 12.** ( $\beta$ -redukcja)  $\beta$ -redukcją nazywamy najmniejszą (w sensie mnogościowym) zgodną na  $\Lambda$  relację binarną  $\rightarrow_{\beta}$  taką, że

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

$\beta$ -*redeksami* będziemy nazywali wyrażenia postaci  $(\lambda x. M)N$ , zaś rezultat ich  $\beta$ -redukcji w postaci termu  $M[x/N]$  –  $\beta$ -*reduktem*. *Ciągiem*  $\beta$ -redukcji nazywamy skończony lub nieskończony ciąg postaci

$$M_0 \rightarrow_{\beta} M_1 \rightarrow_{\beta} \dots$$

**Przykład 6.** Oznaczmy  $Y = \lambda f. (\lambda x. (f(xx))\lambda x. (f(xx)))$  i niech  $F$  będzie dowolnym  $\lambda$ -termem. Wówczas otrzymujemy nieskończony ciąg redukcji postaci

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

**Definicja 13.** ( $\eta$ -redukcja)  $\eta$ -redukcją nazywamy najmniejszą (w sensie mnogościowym) zgodną na  $\Lambda$  relację binarną  $\rightarrow_{\eta}$  taką, że

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



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

**Przykład 7.**  $\lambda x.((+1)x) =_{\eta} (+1)$ .

### 1.3 Kodowanie typów danych

Prosta składnia języka rachunku  $\lambda$  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 [Sussman:1975:IEL:889230].

Najpopularniejszym sposobem reprezentacji danych przez funkcje w rachunku  $\lambda$  oparty jest na kodowaniu liczb Peano za pomocą tzw. liczebników Churcha. Metoda ta, ze względu na wynikające zeń problemy natury złożonościowej [Koopman:2014:CED:2746325.2746330], 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 (ADT<sup>1</sup>) znanych ze współczesnych języków funkcyjnych [Jansen:2013:P9C:2941698.2941699], pozwalając tym samym zaimplementować te konstrukcje w paradygmacie imperatywnym. Fakt, że każdy typ danych można zastąpić tym sposobem odpowiadającą mu funkcją wskazuje na metodę konstruowania prostych języków funkcyjnych, ale przede wszystkim na uniwersalność rachunku  $\lambda$  jako języka przejściowego dla kompilatorów języków wysokiego poziomu [PeytonJones:1992:IFL:129390].

#### 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
```

---

<sup>1</sup>Skrót od angielskojęzycznego *Algebraic Data Types*; nie należy mylić z *Abstract Data Types*.

```

data Tuple a b    = Tuple a b
data Temperature = Fahrenheit Int
                  | Celsius Int
data Maybe a      = Nothing
                  | Just a
data Nat          = Zero
                  | Suc Nat
data List t       = Nil
                  | Cons t (List t)

```

Definicja typu rozpoczynają się od słowa kluczowego `data`<sup>2</sup> 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`.

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

### 1.3.2 Typy wyliczeniowe

Typy wyliczeniowe to typy, które reprezentują możliwe warianty przyjmowanej wartości. Przykładem takiego typu jest `Boolean` z dwoma wartościami: `True`, `False`. Praca z tego rodzaju typami wymaga mechanizmu dopasowywania wzorców (ang. pattern-matching), który pozwala na wybór częściowej definicji funkcji w zależności od zadanego konstruktora wartości. Ponieważ w rachunku  $\lambda$ , którym się tutaj zajmujemy wyrażenia nie mają typów (lub, przyjmując perspektywę systemów z typami: wszystkie mają ten sam typ), interesowało nas będzie nie bezpośrednio kodowanie typu, ale 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  $\lambda$  w

---

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

następujący sposób:

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

Wówczas funkcję `if` możemy reprezentować wyrażeniem  $if \equiv \lambda x. x$ . W ogólnym przypadku, mając następującą definicję ADT:

```
data type_constructor t1 t2 ... tk = C1 t11 ... t1n1
                                   | C2 t21 ... t2n1
                                   ...
                                   | Cm tm1 ... tnm
```

dla  $m, n \in \mathbb{N}$ , wiążemy z nią następującą reprezentację w rachunku  $\lambda$ :

$$\begin{aligned}C_1 &\equiv \lambda t_{11} t_{12} \dots t_{1n_1} f_1 f_2 \dots f_m. f_1 t_{11} t_{12} \dots t_{1n_1} \\ C_2 &\equiv \lambda t_{11} t_{12} \dots t_{1n_2} f_1 f_2 \dots f_m. f_2 t_{11} t_{12} \dots t_{1n_2} \\ &\vdots \\ C_m &\equiv \lambda t_{11} t_{12} \dots t_{1n_m} f_1 f_m \dots f_m. f_1 t_{11} t_{12} \dots t_{1n_m}\end{aligned}$$