

# Spis treści

<b>1</b>	<b>Rachunek <math>\lambda</math> bez typów</b>	<b>3</b>
1.1	Wyrażenia $\lambda$ . . . . .	6
1.2	Redukcja . . . . .	10
1.3	Model Scotta $D_\infty$ . . . . .	16
1.4	Kodowanie typów danych . . . . .	24
1.4.1	Algebraiczne typy danych . . . . .	25
1.4.2	Proste typy wyliczeniowe . . . . .	25
1.4.3	Pary w rachunku $\lambda$ . . . . .	26
1.4.4	Kodowanie rekurencji . . . . .	27
1.4.5	Kodowanie Scotta typów rekursywnych . . . . .	28
1.4.6	Kodowanie Churcha typów rekursywnych . . . . .	29
1.4.7	Ogólny schemat kodowania Scotta typów ADT . . . . .	30
1.5	Podsumowanie . . . . .	30
<b>2</b>	<b>Rachunek <math>\lambda</math> z typami prostymi</b>	<b>31</b>
2.1	Typy proste . . . . .	31
2.2	Typowanie . . . . .	33
2.2.1	Rodzaje problemów . . . . .	35
2.3	Własności . . . . .	36
2.3.1	Uniwersalny polimorfizm . . . . .	40
2.3.2	Silna normalizacja . . . . .	40
2.4	Typy w stylu Churcha . . . . .	45
2.4.1	Składnia . . . . .	45
2.4.2	Typowanie . . . . .	46
2.5	Podsumowanie . . . . .	46
<b>3</b>	<b>System Girarda/Reynoldsa</b>	<b>47</b>
3.1	Termy zależne od typów . . . . .	47
3.2	Typowanie . . . . .	51
3.3	Redukcja . . . . .	54
3.4	Podsumowanie . . . . .	57

For a large class of cases of  
the employment of the word  
‘meaning’ – though not for all –  
this word can be explained in this  
way: the meaning of a word is its  
use in the language. [Wit53]

---

*L. Wittgenstein*

The meaning of a proposition  
is determined by (...) what counts  
as a verification of it. [Mar96]

---

*P. Martin-Löf*

## Wstęp

Pojęcia *funkcji* używa się na ogół mając na myśli jedno z dwóch znaczeń:

- (a) *funkcji* jako algorytmu, który zwraca wartość dla zadanego argumentu.
- (b) *funkcji* w rozumieniu teoriomnogościowym, jako zbiór par argument-wartość.

Ujęcie (a) nazywane jest *semantyką operacyjną*. Oddaje dynamiczny charakter procesu obliczania wartości funkcji jako ciągu wykonywanych w czasie elementarnych operacji na zadanym argumentcie. W kontekście teorii języków programowania przez operacje elementarne należy rozumieć wykonywanie podstawowych instrukcji procesora. W teorii obliczalności to samo rozumielibyśmy pod pojęciem *funkcji obliczalnej*, zaś algorytmiczny proces otrzymywania wartości nazwalibyśmy *efektywnym*.

Ujęcie (b) odpowiada rozumieniu funkcji jako ustalonego, statycznego zbioru przyporządkowań z którego możemy odczytać wartość. Przypisanie funkcjom takiego znaczenia nazywamy *semantyką denotacyjną*. Wymaga ono dostępu do pełnej informacji o funkcji. Niestety, spełnienie tego wymogu na ogół nie może być efektywne ze względu na złożoność pamięciową konieczną do przeprowadzenia takiego procesu (tzw. *memoizacji*).

# 1 Rachunek $\lambda$ bez typów

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 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  $\lambda$ -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  $\lambda$  występujący w (P3) 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:  $\lambda x_1 x_2 \dots x_n. P$ .
- $n$ -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  $\lambda$ -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):  $\lambda x. (x z), \lambda y z. x, \lambda x. (\lambda x. (x x))$ .

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)  $\text{Sub}(\lambda x. x x) = \{(\lambda x. x x)^1, (x x)^1, x^2\}$
- (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 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 (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 założenia indukcyjnego wnosimy, że

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

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 założenia indukcyjnego 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 $\lambda$

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

**Definicja 5.** ( $\alpha$ -konwersja) 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 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 założenia indukcyjnego  $N =_\alpha L$  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 założenia indukcyjnego  $N' =_\alpha M'$  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 założenia indukcyjnego 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  $\Lambda$   $\lambda$ -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:

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

**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  $\lambda$ -termach postaci (1) i (2) mówimy, że są w *czołowej postaci normalnej* (HNF, ang. *head normal form*) albo w *słabej czołowej postaci normalnej* (WHNF, ang. *weak head normal form*), odpowiednio.

**Dowód.** Z definicji  $\lambda$ -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 abstrakcją (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)).  $\square$

**Przykład 5.**  $\lambda x. (\lambda y. y) M$  jest w postaci WHNF, ale nie jest w postaci HNF, ponieważ zawiera redeks czołowy  $(\lambda y. y)M$ .

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 2.

**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 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  $\alpha$ -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)  $(\lambda y. P)[x/N] = \lambda y. P$ , jeśli  $x \notin \text{FV}(P)$

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

(S'7)  $(\lambda y. P)[x/N] = \lambda z. P[y/z][x/N]$ , gdzie  $x \in \text{FV}(P)$  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  $\iota$  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 [Bru72] 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 (np. rachunku SKI), eliminuje konieczność utożsamiania termów przez  $\alpha$ -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]. Technika definiowania  $\lambda$ -termów jako klas  $\alpha$ -konwersji są na ogół w literaturze pomijane.

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

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

$$(\vec{s}2) \ y[\vec{x}/\vec{N}] = y \text{ o ile dla dowolnego } i \in \mathbb{N}, y \neq x_i.$$

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

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

*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 6.** 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  $\lambda$ -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  $M\mathcal{R}N$ , 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 praporządek, 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  $\lambda$ -termów, będzie dokonywała podziału w  $\mathbf{\Lambda}$  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:



(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:



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.

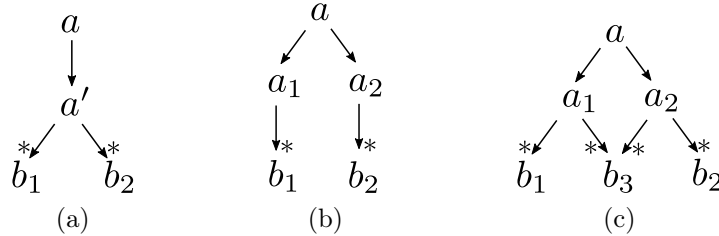
**Dowód.** Niech  $\rightarrow$  będzie relacją binarną na  $A$  o własności SN i WCR. Ponieważ  $\rightarrow$  jest SN, to każdy  $a$  jest normalizowalny.

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

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

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

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

Relację  $=_\beta$  nazywamy  $\beta$ -konwersją. Zauważmy, że  $M =_\beta N$  wtedy i tylko wtedy, gdy istnieje skończony ciąg  $\lambda$ -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  $0 \leq i < n$ .

**Przykład 7.** Wszystkie pary  $\lambda$ -termów ze zbioru

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

są swoimi  $\beta$ -konwersami. Mamy:

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

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

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

*Konwencja.* Składnię rachunku  $\lambda$  często rozszerza się o wyrażenia let pozwalające konstruować  $\beta$ -redeksy w czytelny sposób. Rozszerzenie ma następującą postać:

$$\text{let } x=N \text{ in } M \equiv (\lambda x. M)N$$

Jest to przykład tzw. *cukru syntaktycznego*, czyli wtórnych rozszerzeń języka, które ułatwiają jego użycie. Wyrażenia let w których  $M \equiv \lambda y. M'$  dla pewnego  $M' \in \mathbf{\Lambda}$  nazywamy *domknięciami*<sup>1</sup> (ang. *closure*). Nieformalnie, pozwalają one na przypisywanie wartości zmiennym o tzw. *zakresie leksykalnym*.

**Definicja 16.** (Strategia redukcji) Strategią redukcji nazywamy każde odwzorowanie  $S : \mathbf{\Lambda} \rightarrow \mathbf{\Lambda}$  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  $F^i(M) \equiv \underbrace{F(F(\dots(F(M))\dots))}_{i\text{-razy}} \in \text{NF}_\beta$ .

---

<sup>1</sup>Idiom ten w literaturze poświęconej językom programowania z rodziny Lisp powszechnie występuje pod nazwą wyrażen *let-over-lambda*.

**Przykład 8.** (a) Oznaczmy  $Y \equiv \lambda f. (\lambda x. (f(xx)) \lambda x. (f(xx)))$  i niech  $F$  będzie dowolnym  $\lambda$ -termem. Wówczas otrzymujemy nieskończony ciąg redukcji postaci

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

$Y$  nazywamy *kombinatorem punktu stałego*. Widzimy, że relacja  $\beta$ -redukcji w rachunku  $\lambda$  nie jest ani słabo, ani silnie normalizująca.

- (b) Niech  $\Omega \equiv (\lambda x. xx)(\lambda x. xx)$ .  $\Omega$  jest  $\beta$ -redexem, którego redukcja prowadzi do ponownego otrzymania termu  $\Omega$  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  $\lambda$ -termu posiadającego więcej niż jeden redex może prowadzić do różnych (choć  $\beta$ -równoważnych) reduktów. Zależy to od wyboru strategii redukcji. Rozważmy następujący term:  $(\lambda u. v) \Omega$ . Konsekwentne redukowanie podtermu  $\Omega$  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  $\Omega$  do  $(\lambda u. v)$  otrzymujemy natychmiastowo redex w postaci normalnej.

**Definicja 17.** ( $\eta$ -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 9.** Przypuśćmy, że  $(+1) \in \mathbf{\Lambda}$ . Wówczas  $\lambda x.((+1)x) =_{\eta} (+1)$ .

Widzieliśmy, że  $\beta$ -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  $\lambda$ -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 [Bar84, Rozdział 3.2] i ze względu na jego obszerność pozwalamy sobie go pominąć.

**Fakt 1.** (*Twierdzenie Churcha-Rossera*).  $\beta$ -redukcja ma własność CR.

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

**Dowód.** Niech  $M, N \in \mathbf{\Lambda}$  będą takie, że  $M =_{\beta} N$ . Wówczas istnieje ciąg  $\lambda$ -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

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

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  $\leftrightarrow_{\beta}$  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

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

- (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  $\lambda$ -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  $L \in \Lambda$  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 Model Scotta $D_\infty$

**Wprowadzenie** Ustalmy wpierw, że nie możemy naiwnie interpretować  $\lambda$ -termów jako funkcji i aplikacji argumentów do funkcji. Przypuśćmy bowiem, że w pewnej interpretacji  $\llbracket M \rrbracket = f_M$ , gdzie  $f_M \in A \rightarrow B$  dla pewnych zbiorów  $A$  i  $B$ . Wówczas  $\llbracket MM \rrbracket = f_M(f_M)$ , a zatem  $f_M \in A$ . Oznacza to, że funkcja  $f_M$  jest elementem swojej własnej dziedziny, czyli istnieje nieskończony zstępujący ciąg zbiorów

$$A \times f(A) \supset A \times f(A) \times f(A) \supset A \times f(A) \times f(A) \times f(A) \supset \dots$$

Istnienie takiego ciągu narusza aksjomat ufundowania na gruncie aksjomatyki ZF, a to wyklucza możliwość określenia takich modeli.

Rozwiązanie Scotta polega na tym, aby zamiast próbować interpretować termy jako funkcje, przypisać im nieskończone ciągi funkcji postaci

$$\varphi = (\varphi_0, \varphi_1, \varphi_2, \dots),$$

gdzie  $\varphi \in D_\infty$  i  $\varphi_n \in D_n$ . Określając na takiej strukturze aplikację następująco

$$\varphi \bullet \psi = (\varphi_1(\psi_0), \varphi_2(\psi_1), \varphi_3(\psi_2), \dots),$$

widzimy, że samoaplikacja jest poprawnie określona:

$$\varphi \bullet \varphi = (\varphi_1(\varphi_0), \varphi_2(\varphi_1), \varphi_3(\varphi_2), \dots).$$

W rozdziale tym przybliżymy konstrukcję modelu Scotta  $D_\infty$  zgodnie z [HS08, Rozdział 16]. Ze względu na obszerność materiału zaprezentujemy wyłącznie konstrukcję modelu bezsyntaktycznego (ang. *syntax-free*) milcząco traktując jej topologiczny charakter.

Celem wprowadzenia ustalmy szereg następujących pojęć.



**Definicja 18** (Struktura aplikatywna). Parę  $(D, \bullet)$ , gdzie  $D$  jest zbiorem zawierającym przynajmniej dwa elementy i w którym symbol  $\bullet$  oznacza działanie binarne na  $D$ , nazywamy *strukturą aplikatywną*.

*Konwencja.* Ponieważ nie zakładamy, że działanie  $\bullet$  jest łączne, to celem ograniczenia ilości nawiasów przyjmijmy, że  $\bullet$  wiąże lewostronnie, czyli  $a \bullet b \bullet c \bullet d \equiv ((a \bullet b) \bullet c) \bullet d$ .

**Definicja 19** (Ekstensjonalna równoważność). Niech  $(D, \bullet)$  będzie strukturą aplikatywną i niech  $a, b \in D$ . Określamy relację:

$$a \sim b \iff \forall d \in D (a \bullet d = b \bullet d).$$

Powiemy, że  $a$  jest *ekstensjonalnie równoważne*  $b$ , jeśli  $a \sim b$ .

**Definicja 20.** 1. Powiemy, że term  $M$  jest *kombinacją zmiennych*, jeśli  $M \equiv x$  dla pewnej zmiennej  $x$  albo  $M \equiv (PQ)$  dla pewnych kombinacji zmiennych  $P$  i  $Q$ .

2. Niech  $M$  będzie kombinacją zmiennych taką, że  $FV(M) = \{x_1, x_2, \dots, x_n\}$ . Względem  $M$  określamy funkcję  $f_M : D^n \rightarrow D$  następującym wzorem:

$$f_M(d_1, d_2, \dots, d_n) = \begin{cases} d_k, & \text{jeśli } M \equiv x_k, \\ f_Q(d_1, d_2, \dots, d_n) \bullet f_Q(d_1, d_2, \dots, d_n), & \text{jeśli } M \equiv (PQ). \end{cases}$$

3. Powiemy, że struktura aplikatywna  $(D, \bullet)$  jest *kombinatorycznie zupełna*, jeśli dla każdej kombinacji zmiennych  $M$  istnieje  $a_M \in D$  taki, że dla wszystkich  $d_1, d_2, \dots, d_n \in D$

$$a_M \bullet d_1 \bullet d_2 \bullet \dots \bullet d_n = f_M(d_1, d_2, \dots, d_n).$$

**Definicja 21** (Model bezsyntaktyczny). Modelem bezsyntaktycznym rachunku  $\lambda$  nazywamy trójkę  $(D, \bullet, \Lambda)$ , gdzie  $\mathbb{D} = (D, \bullet)$  jest strukturą aplikatywną,  $\Lambda : D \rightarrow D$  i spełnione są poniższe własności:

- (a)  $\mathbb{D}$  jest kombinatorycznie zupełny.
- (b)  $\Lambda(a) \sim a$  dla wszystkich  $a \in D$ .
- (c) Jeśli  $a \sim b$ , to  $\Lambda(a) = \Lambda(b)$  dla wszystkich  $a, b \in D$ .
- (d)  $\Lambda(a) = e \bullet a$  dla pewnego  $e \in D$  i wszystkich  $a \in D$ .

**Elementy teorii porządku** Niech  $(D, \sqsubseteq)$  będzie zbiorem częściowo uporządkowanym. Powiemy, że  $b \in D$  jest elementem *najmniejszym*, jeśli  $b \sqsubseteq d$  dla wszystkich  $d \in D$ . Element ten, o ile istnieje, wyznaczony jest jednoznacznie i będziemy oznaczać go symbolem  $\perp$ . Niech  $X \subset D$ . *Ograniczeniem górnym*  $X$  nazywamy element  $u \in D$  taki, że  $x \sqsubseteq u$  dla wszystkich  $x \in X$ . *Kresem górnym* zbioru  $X$  nazywamy element  $\ell \in D$  taki, że  $\ell$  jest ograniczeniem górnym  $X$  i  $\ell \sqsubseteq u$  dla wszystkich ograniczeń górnych  $u$  zbioru  $X$ . Element taki, o ile istnieje, będziemy oznaczali symbolem  $\sqcup X$ . Podzbiór  $X \subset D$  nazywamy *skierowanym*, jeśli  $X \neq \emptyset$  i dla dowolnych  $x, y \in X$  istnieje  $z \in X$  taki, że  $x \sqsubseteq z$  i  $y \sqsubseteq z$ .

**Definicja 22** (Zupełny porządek częściowy). Porządek częściowy  $(D, \sqsubseteq)$  taki, że

- (a) posiada element najmniejszy oraz
- (b) każdy skierowany podzbiór  $X \subset D$  ma kres górny,

nazywamy *zupełnym porządkiem częściowym* (w skrócie: *cpo*).

Ustalmy, że jeśli  $D', D'', \dots$  są cpo, to odpowiadające im porządki częściowe będziemy notowali symbolami  $\sqsubseteq', \sqsubseteq'', \dots$

**Przykład 10.** Ustalmy  $\perp \notin \mathbb{N}$  i niech  $\mathbb{N}^+ = \mathbb{N} \cup \perp$ . Określmy na  $\mathbb{N}^+$  następującą relację:

$$a \sqsubseteq b \iff (a = \perp \wedge b \in \mathbb{N}) \vee a = b$$

$\sqsubseteq$  jest oczywiście zwrotna, przechodnia i antysymetryczna. Widzimy, że  $\mathbb{N}^+$  ma względem niej element najmniejszy (jest nim  $\perp$ ) i oczywiście każdy podzbiór  $\mathbb{N}^+$  jest skierowany.

**Definicja 23** (Monotoniczność, ciągłość). Niech  $D$  i  $D'$  będą cpo i  $\varphi : D \rightarrow D'$ .

- (a) Powiemy, że  $\varphi$  jest *monotoniczna*, jeśli  $\varphi(a) \sqsubseteq' \varphi(b)$  dla  $a \sqsubseteq b$ .
- (b) Powiemy, że  $\varphi$  jest *ciągła* (w sensie Scotta<sup>2</sup>), jeśli  $\varphi(\sqcup X) = \sqcup \varphi(X)$  dla wszystkich skierowanych podzbioru  $X \subseteq D$ .

Symbolem  $[D \rightarrow D']$  oznaczamy zbiór wszystkich funkcji ciągłych ze zbioru  $D$  do  $D'$ .

*Uwaga.* Zauważmy, że jeśli  $\varphi$  jest ciągła, to jest również monotoniczna. Istotnie, jeśli  $a \sqsubseteq b$ , to w szczególności  $\{a, b\} \subseteq D$  jest skierowanym podzbiorem  $D$ . Wówczas  $\sqcup \{a, b\} = b$  i ponieważ  $\sqcup \varphi(\{a, b\}) = \varphi(\sqcup \{a, b\})$ , to otrzymujemy, że  $\varphi(a) \sqsubseteq \varphi(\sqcup \{a, b\}) = \varphi(b)$ .

**Twierdzenie 3.** Jeśli  $D$  i  $D'$  są cpo, to  $[D \rightarrow D']$  jest cpo.

<sup>2</sup>Funkcje te są ciągłe w topologicznym sensie względem topologii Scotta.

**Dowód.** Określmy na  $[D \rightarrow D']$  relację  $\leq$ :

$$\varphi \leq \psi \iff \forall d \in D (\varphi(d) \sqsubseteq' \psi(d)),$$

Ponieważ  $D'$  jest cpo, to aby wykazać, że  $[D \rightarrow D']$  ma element najmniejszy, wystarczy, że rozpatrzmy  $\perp(d) = \perp'$  dla  $d \in D$ .

Niech teraz  $\Phi$  będzie skierowanym podzbiorem  $[D \rightarrow D']$ . Wówczas dla wszystkich  $d \in D$  zbiór  $\{\varphi(d) \mid \varphi \in \Phi\}$  jest skierowanym podzbiorem  $D'$ .

Istotnie, wybierzmy  $y_1, y_2 \in \{\varphi(d) \mid \varphi \in \Phi\}$ . Wówczas dla pewnych  $\varphi_1, \varphi_2 \in \Phi$  mamy, że  $y_1 = \varphi_1(d)$  oraz  $y_2 = \varphi_2(d)$ . Ponieważ  $\Phi$  jest skierowany, to istnieje  $\varphi_3$  taki, że  $\varphi_1 \sqsubseteq \varphi_3$  i  $\varphi_2 \sqsubseteq \varphi_3$ . Z monotoniczność funkcji ciągłych i faktu, że  $d \sqsubseteq d$  widzimy, że  $\{\varphi(d) \mid \varphi \in \Phi\}$  jest skierowanym podzbiorem  $D'$ . Ponieważ zaś  $D'$  jest cpo, to dla każdego  $d \in D$  istnieje kres górny  $\{\varphi(d) \mid \varphi \in \Phi\}$ . Określmy więc funkcję  $\Psi_\Phi : D \rightarrow D'$  następującym wzorem:

$$\Psi_\Phi(d) = \bigsqcup \Phi d,$$

gdzie  $\Phi d = \{\varphi(d) \mid \varphi \in \Phi\}$ . Wystarczy teraz pokazać, że  $\Psi_\Phi$  jest ciągła i jest kresem górnym zbioru  $\Phi$ .

1. Pokażemy najpierw, że  $\Psi_\Phi$  jest ciągła. Niech  $X$  będzie dowolnym skierowanym podzbiorem  $D$ . Wówczas:

$$\begin{aligned} \Psi_\Phi(\bigsqcup X) &= \bigsqcup \Phi(\bigsqcup X) \\ &= \bigsqcup \{\varphi(\bigsqcup X) \mid \varphi \in \Phi\} \\ &= \bigsqcup \{\bigsqcup \{\varphi(d) \mid d \in X\} \mid \varphi \in \Phi\} \\ &= \bigsqcup \left\{ \bigcup_{\varphi \in \Phi} \{\varphi(d) \mid d \in X\} \right\} \\ &= \bigsqcup \left\{ \bigcup_{\varphi \in \Phi} \bigcup_{d \in X} \{\varphi(d)\} \right\} = \bigsqcup \left\{ \bigcup_{d \in X} \bigcup_{\varphi \in \Phi} \{\varphi(d)\} \right\} \\ &= \bigsqcup \left\{ \bigcup_{d \in X} \{\varphi(d) \mid \varphi \in \Phi\} \right\} \\ &= \bigsqcup \{\bigsqcup \{\varphi(d) \mid \varphi \in \Phi\} \mid d \in X\} \\ &= \bigsqcup \{\bigsqcup \Phi d \mid d \in X\} = \bigsqcup \Psi_\Phi(X). \end{aligned}$$

2.  $\Phi$  jest ograniczony w  $[D \rightarrow D']$ . Istotnie, przypuśćmy, że dla dowolnego  $f \in [D \rightarrow D']$  istnieje  $g \in \Phi$  taki, że  $f \leq g$ . Wówczas w szczególności  $\Psi_\Phi \leq g$  i stąd również

$$\bigsqcup \Phi d \sqsubseteq' g(d) \quad \text{dla wszystkich } d \in D.$$

Ponieważ  $g \in \Phi$ , to  $g(d) \subseteq \sqcup \{g(d) \mid g \in \Phi\} = \sqcup \Phi d$ . Zatem  $\Psi_\Phi$  jest ograniczeniem górnym  $\Phi$ .

Przypuśćmy teraz, że  $\Phi$  nie ma kresu górnego. Wówczas dla dowolnego ograniczenia górnego  $f$  zbioru  $\Phi$  istnieje  $g$  takie, że  $g \leq f$  i które jest również ograniczeniem górnym. Wówczas w szczególności mamy, że  $g \leq \Psi_\Phi$ , czyli dla dowolnego  $d \in D$

$$g(d) \subseteq' \sqcup \Phi d \in D'.$$

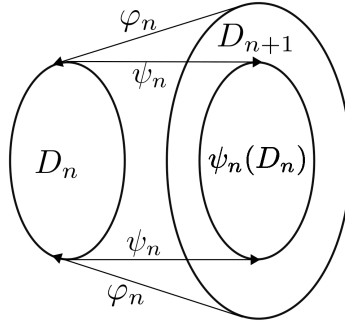
Ale  $\sqcup \Phi d$  jest kresem górnym  $\Phi d$  (wykazaliśmy bowiem, że  $\Phi d$  jest skierowany, zaś  $D'$  jest cpo), a zatem  $g(d)$  nie może być ograniczeniem  $\Phi d$ . Istnieje stąd  $z \in \Phi d$  taki, że  $g(d) \subseteq' z$ . Wówczas istnieje  $\tilde{\varphi} \in \Phi$  taki, że  $z = \tilde{\varphi}(d)$  oraz  $g(d) \subseteq' \tilde{\varphi}(d)$ . Z dowolności  $d$  mamy, że  $g \leq \tilde{\varphi}$ . Zatem  $g$  nie jest ograniczeniem górnym  $\Phi$ , co jest sprzeczne z założeniem. Stąd  $\Psi_\Phi$  musi być kresem górnym  $\Phi$ .  $\square$

**Wniosek 5.** Dla danego cpo  $D_0$  możemy skonstruować ciąg cpo  $\{D_n\}_{n=0}^\infty$  określając  $D_{n+1} = [D_n \rightarrow D_n]$  dla  $n \geq 0$ . Tak określony ciąg dla  $D_0 = \mathbb{N}^+$  (patrz Przykład 10) nazywamy modelem Scotta.

**Definicja 24** (Projekcja). Niech  $D$  i  $D'$  będą cpo. Projekcją z  $D'$  do  $D$  nazywamy parę  $(\varphi, \psi)$  funkcji  $\varphi \in [D \rightarrow D']$ ,  $\psi \in [D' \rightarrow D]$  takich, że

$$\psi \circ \varphi = I_D \quad \text{oraz} \quad \varphi \circ \psi \leq I_{D'} \quad (2)$$

gdzie przez  $I_D$  i  $I_{D'}$  oznaczamy funkcję identycznościową na zbiorze  $D$  i  $D'$ , odpowiednio.



Rysunek 3: Projekcja  $(\varphi_n, \psi_n)$  z  $D_{n+1}$  do  $D_n$

Okazuje się, że ciąg cpo  $\{D_n\}_{n=0}^\infty$  skonstruowany w myśl Wniosku 5 można określić definiując dla każdego  $n \in \mathbb{N}$  projekcję  $(\varphi_n, \psi_n)$  z  $D_{n+1}$  do  $D_n$ . Istotnie,

spróbujemy skonstruować taką rodzinę projekcji. Wybierzmy  $d \in D_0$  i niech  $\kappa_d$  oznacza funkcję stałą

$$\kappa_d(c) = d \quad \text{dla wszystkich } c \in D_0.$$

Określmy  $D_1 = [D_0 \rightarrow D_0]$ . Ponieważ  $\kappa_d$  jest funkcją ciągłą, to  $\kappa_d \in D_1$ . Niech teraz

$$\begin{aligned} \varphi_0(d) &= \kappa_d \quad \text{dla wszystkich } d \in D_0, \\ \psi_0(c) &= c(\perp_0) \quad \text{dla wszystkich } c \in D_1, \end{aligned}$$

gdzie  $\perp_0$  jest elementem najmniejszym  $D_0$ . Widzimy, że  $\varphi_0 : D_0 \rightarrow D_1$  i  $\psi_0 : D_1 \rightarrow D_0$ . Funkcje  $\varphi_0$  i  $\psi_0$  są ciągłe, zaś  $\psi_0 \circ \varphi_0 = I_{D_0}$  (dowód pomijamy), zatem  $(\varphi_0, \psi_0)$  jest projekcją z  $D_1$  do  $D_0$ .

Dla  $n > 0$  określamy teraz  $\varphi_n : D_n \rightarrow D_{n+1}$  i  $\psi_{n+1} : D_{n+1} \rightarrow D_n$  następującym wzorem:

$$\begin{aligned} \varphi_n(\sigma) &= \varphi_{n-1} \circ \sigma \circ \psi_{n-1} \quad \text{dla } \sigma \in D_n, \\ \psi_n(\tau) &= \psi_{n-1} \circ \varphi_{n-1} \quad \text{dla } \tau \in D_{n+1} \end{aligned}$$

Wówczas  $\varphi_n \in [D_n \rightarrow D_{n+1}]$ ,  $\psi_n \in [D_{n+1} \rightarrow D_n]$  i  $\psi_n \circ \varphi_n = I_{D_n}$  oraz  $\varphi_n \circ \psi_n \leq I_{D_{n+1}}$ . [HS08, Lemat 16.28]. A zatem  $(\varphi_n, \psi_n)$  jest projekcją z  $D_{n+1}$  do  $D_n$ .

Ponieważ projekcje  $(\varphi_n, \psi_n)$  przenoszą nas tylko pomiędzy następującymi po sobie cpo w ciągu  $\{D_n\}_{n=0}^\infty$ , (Rysunek 3) określamy następujące złożenie pozwalające na projekcje między dowolnymi dwoma wyrazami ciągu.

**Definicja 25.** Dla  $m, n \geq 0$  określamy  $\varphi_{mn} : D_m \rightarrow D_n$  w następujący sposób:

$$\varphi_{mn} = \begin{cases} \varphi_{n-1} \circ \varphi_{n-2} \circ \dots \circ \varphi_{m+1} \circ \varphi_m, & \text{jeśli } m \leq n, \\ I_{D_n}, & \text{jeśli } m = n, \\ \psi_n \circ \psi_{n+1} \circ \dots \circ \psi_{m-2} \circ \psi_{m-1} & \text{jeśli } m \geq n. \end{cases}$$

Ponieważ dla każdego  $n \in \mathbb{N}$  para  $(\varphi_n, \psi_n)$  jest projekcją z  $D_{n+1}$  do  $D_n$ , to zachodzi następujący szereg własności.

**Fakt 2** ([HS08, p. 16.33]). *Niech  $m, n \geq 0$ . Wówczas*

- (i)  $\varphi_{mn} \in [D_m \rightarrow D_n]$ ,
- (ii) *jeśli  $m \leq n$ , to  $\varphi_{nm} \circ \varphi_{mn} = I_{D_m}$ ,*
- (iii) *jeśli  $m > n$ , to  $\varphi_{nm} \circ \varphi_{mn} \leq I_{D_m}$ ,*
- (iv) *jeśli  $m < n$ , to  $(\varphi_{mn}, \varphi_{nm})$  jest projekcją z  $D_n$  do  $D_m$ ,*
- (v) *jeśli  $m < k < n$  lub  $n < k < m$ , to  $\varphi_{kn} \circ \varphi_{mk} = \varphi_{mn}$ .*

**Konstrukcja  $D_\infty$**  Mając zadane dwa cpo  $D, D'$  możemy zapytać czy istnieje włożenie jednego z nich w drugi. Zauważmy, że jeśli  $\varphi, \psi$  jest projekcją z  $D'$  do  $D$ , to  $\phi$  jest włożeniem  $D$  w  $D'$  (w sensie topologii Scotta). Ciąg  $\{D_n\}_{n=0}^\infty$  intuicyjnie przypomina więc wstępujący ciąg zbiorów. Formalizuje to następująca definicja.

**Definicja 26.** Niech  $D_\infty$  oznacza zbiór wszystkich nieskończonych ciągów postaci

$$d = (d_0, d_1, \dots)$$

takich, że dla wszystkich  $n \geq 0$  mamy, że  $d_n \in D_n$  oraz  $\psi_n(d_{n+1}) = d_n$ . Na zbiorze  $D_\infty$  określamy relację  $\sqsubseteq$  w następujący sposób:

$$(d_0, d_1, \dots) \sqsubseteq (d'_0, d'_1, \dots) \iff \forall n \geq 0 (d_n \sqsubseteq d'_n)$$

Przez  $d_n$  oznaczać będziemy  $n$ -ty element ciągu  $d$ . Jeśli  $X \subset D_\infty$ , to określamy  $X_n = \{d_n \mid d \in X\}$ .

Przy powyższym określeniu  $D_\infty$  okazuje się być cpo.

**Fakt 3** ([HS08, p. 16.36]). (i)  $D_\infty$  jest cpo.

(ii)  $D_\infty$  zawiera element najmniejszy  $\perp = (\perp_0, \perp_1, \dots)$ , gdzie przez  $\perp_n$  oznaczamy najmniejszy element  $D_n$ .

(iii) Kres górny każdego skierowanego zbioru  $X \subset D_\infty$  ma postać

$$\bigsqcup X = (\bigsqcup X_0, \bigsqcup X_1, \dots)$$

Dodatkowo, określamy projekcję z  $D_\infty$  do  $D_n$  dla każdego  $n \in \mathbb{N}$ . Wymaga to dowodu (patrz Fakt 4(i)), który pomijamy.

**Definicja 27** ( $D_\infty$ ). Dla  $n \geq 0$  określamy funkcje  $\varphi_{n\infty} : D_n \rightarrow D_\infty$  oraz  $\varphi_{\infty n} : D_\infty \rightarrow D_n$ , gdzie

$$\begin{aligned} \varphi_{n\infty}(d) &= (\varphi_{n0}(d), \varphi_{n1}(d), \dots) \text{ dla } d \in D_n, \\ \varphi_{\infty n}(d) &= d_n. \end{aligned}$$

Określona para funkcji spełnia poniższy szereg własności.

**Fakt 4** ([HS08, pp. 16.38, 16.39, 16.42]). Niech  $m, n \geq 0$ ,  $m \leq n$  i  $a, b \in D_\infty$ . Wówczas:

- (i)  $(\varphi_{n\infty}, \varphi_{\infty n})$  jest projekcją z  $D_\infty$  do  $D_n$ ,
- (ii)  $\varphi_{mn}(a_m) \sqsubseteq a_n$ ,
- (iii)  $\varphi_{m\infty}(a_m) \sqsubseteq \varphi_{n\infty}(a_n)$ ,

$$(iv) \ a = \sqcup_{n \geq 0} \varphi_{n\infty}(a_n),,$$

$$(v) \ \varphi_{n\infty}(a_{n+1}(b_n)) \sqsubseteq \varphi_{(n+1)\infty}(a_{n+2}(b_{n+1})).$$

Zauważmy, że  $\varphi_{n\infty}$  jest izomorficznym włożeniem  $D_n$  w  $D_\infty$ . Oznacza to, że każdy element  $a \in D_n$  możemy utożsamiać z odpowiednim  $\varphi_{n\infty}(a) \in D_\infty$ . Biorąc  $a \in D_\infty$  i stosując tę odpowiedniość jako konwencję notacyjną, na podstawie 4(iii) mamy, że

$$a_0 \sqsubseteq a_1 \sqsubseteq a_2 \sqsubseteq a_3 \sqsubseteq \dots,$$

zaś na podstawie 4(iv):

$$a = \sqcup \{a_0, a_1, a_2, a_3, \dots\}.$$

Zatem wyrazy  $a_0, a_1, a_2, a_3, \dots$  możemy traktować jako kolejne przybliżenia elementu  $a$ . Zauważmy, że na podstawie Faktu (v) ciąg ten jest zawsze rosnący. Traktując nieformalnie  $D_\infty$  jako pewną przestrzeń informacji i w niej: element  $\perp$  jako brak informacji, zaś relację  $\sqsubseteq$  odczytując jako „więcej informacji”, kolejne przybliżenia możemy odczytać jako proces poznawczy prowadzący do stanu pełnej informacji o jakimś fakcie.

Fakty 4(iv) i 4(v) sugerują metodę określenia struktury aplikatywnej na  $D$ . Przypuśćmy bowiem, że  $a, b \in D_\infty$ . Wówczas dla każdego  $n \in \mathbb{N}$  wyrazy  $a_n$  i  $b_n$  są przybliżeniami  $a$  i  $b$ , odpowiednio. Ponieważ  $a_{n+1} \in D_{n+1} = [D_n \rightarrow D_n]$ , to  $a_{n+1}(b_n)$  jest określony. A zatem  $a_{n+1}(b_n)$  jest pewnym przybliżeniem aplikacji  $a$  do  $b$ . Widzimy jednocześnie, że rozwiązuje to problem samoaplikacji wymieniony na wstępie.

**Definicja 28.** Dla  $a, b \in D_\infty$  określamy:

$$a \bullet b = \sqcup \{\varphi_{n\infty}(a_{n+1}(b_n)) \mid n \geq 0\}$$

Zauważmy, że na podstawie Faktu 4(v)  $D_\infty$  jest łańcuchem, a zatem zbiorem skierowanym. Ponieważ  $D_\infty$  jest cpo, to supremum takiego zbioru zawsze istnieje. Zatem działanie  $\bullet$  jest poprawnie określone. Dowód, że operacja ta nie wyprowadza poza zbiór funkcji ciągłych pomijamy.

**Fakt 5** ([HS08, p. 14.29]). *Struktura aplikatywna  $(D, \bullet)$  jest kombinatorycznie zupełna wtedy i tylko wtedy, gdy istnieją takie  $k, s \in D$ , że dla wszystkich  $a, b, c \in D$  mamy:*

$$k \bullet a \bullet b = a \quad \text{oraz} \quad s \bullet a \bullet b \bullet c = a \bullet c \bullet (b \bullet c).$$

**Fakt 6** ([HS08, p. 15.30]). *Niech  $(D, \bullet)$  będzie kombinatorycznie zupełną strukturą aplikatywną taką, że dla dowolnych  $a, b \in D$  jeśli  $a \sim b$ , to  $a = b$ . Określmy  $\Lambda(a) = a$  dla wszystkich  $a \in D$ . Wówczas  $(D, \bullet, \Lambda)$  jest modelem bezsyntaktycznym.*

Ostatecznie, na podstawie Faktu 5 i Faktu 6 zachodzi następujące twierdzenie.

**Twierdzenie 4.**  $(D, \bullet, \Lambda)$  jest modelem bezsyntaktycznym.

**Dowód.** Określmy:

$$\begin{aligned} k &= (\perp_0, I_{D_0}, k_2, k_3, \dots), \\ s &= (\perp_0, I_{D_0}, \psi_2(s_3), s_3, s_4, \dots). \end{aligned}$$

Okazuje się [HS08, Tw. 16.51, 16.53], że  $k, s \in D_\infty$  oraz, że dla dowolnych  $a, b, c \in D_\infty$  mamy

$$k \bullet a \bullet b = a \quad \text{oraz} \quad s \bullet a \bullet b \bullet c = a \bullet c \bullet (b \bullet c).$$

Struktura  $(D, \bullet)$  jest ekstensjonalna. Istotnie, niech  $a, b \in D_\infty$  takie, że  $a \sim b$  i  $m \geq 0$ . Wybierzmy  $d \in D_m$  oraz ustalmy  $c = \varphi_{m\infty}(d)$ . Wówczas [HS08, Tw. 16.54]

$$(a \bullet c)_m = a_{m+1}(d) \quad \text{oraz} \quad (b \bullet c)_m = b_{m+1}(d).$$

Stąd  $a_{m+1}(d) = (a \bullet c)_m = (b \bullet c)_m = b_{m+1}(d)$ , a zatem  $a_n = b_n$  dla  $n > 0$ . Wówczas również  $a_0 = \psi_0(a_1) = \psi_0(b_1) = b_0$  i stąd  $a = b$ .

A zatem na podstawie Faktu 6  $(D, \bullet, \Lambda)$  jest modelem bezsyntaktycznym.  $\square$

## 1.4 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. 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  $\lambda$  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 (ADT, ang. *Algebraic Data Types*<sup>3</sup>) znanych ze współczesnych języków funkcyjnych [Jan13], pozwalając tym samym zaimplementować te konstrukcje w dowolnym języku, w którym dostępne są wyrażenia  $\lambda$ . Fakt, że każdy typ danych można zastąpić tym sposobem odpowiadającą mu funkcją, wskazuje na metodę konstruowania prostych języków funkcyjnych

<sup>3</sup>Nie należy mylić z *Abstract Data Types*.



[JKP06] oraz na uniwersalność rachunku  $\lambda$  jako języka przejściowego dla kompilatorów języków funkcyjnych [PL92, Rozdział 3]. Druga z tych idei znajduje dziś bardzo praktyczne zastosowanie w przypadku Systemu F – rozszerzenia rachunku  $\lambda$ , który będzie tematem Rozdziału 3 – i kompilatora GHC języka Haskell.

### 1.4.1 Algebraiczne typy danych

Algebraiczne typy danych są podstawowym środkiem służącym do określania struktur danych we współczesnych funkcyjnych językach programowania. Na potrzeby prezentacji poszczególnych kodowań posłużymy się intuicjami o ADT zbudowanymi 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`<sup>4</sup> po którym występuje *konstruktor typu*. Na wzór notacji BNF, typy przyjmują jedną z *wartości* odzielonych 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  $\lambda$ .

### 1.4.2 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

---

<sup>4</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`.

konstruktora wartości. Ponieważ w rachunku  $\lambda$  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  $\lambda$  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  $\eta$ -reduktem:  $\lambda c. c$ .

### 1.4.3 Pary w rachunku $\lambda$

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 domknięcie. Standardowym sposobem reprezentacji pary w rachunku  $\lambda$  jest:

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

Używając wyrażeń `let`, powyższą reprezentację możemy przepisać w postaci:

$$\text{let } a = a \text{ } b = b \text{ in } f$$

Aplikując `Tuple` tylko do dwóch termów (*domykając* term `Tuple`) otrzymujemy reprezentację pary. Argument  $f$  nazywamy *kontynuacją*, gdyż aplikując  $(\text{Tuple } x \ y)$  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:

$$\begin{aligned}\text{fst} &\equiv \lambda t. t(\lambda ab. a) \\ \text{snd} &\equiv \lambda t. t(\lambda ab. b)\end{aligned}$$

**Przykład 11.** Wprowadzone konstrukcje pozwalają nam na definicję skończonych (w sensie liczby konstruktorów) typów. Rozważmy następujące przykłady:

a) Konstruktory wartości typu `Maybe` możemy reprezentować przez

$$\begin{aligned}\text{Nothing} &\equiv \lambda n j. n \\ \text{Just} &\equiv \lambda a n j. j a\end{aligned}$$

Rozważmy następującą funkcję:

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

Odpowiadająca jej reprezentacja to

$$\text{maybe} \equiv \lambda b f t. t b (\lambda a. f a)$$

b) Rozważmy następującą funkcję

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

Ustalając reprezentację konstruktorów `Fahrenheit` i `Celsius`:

$$\begin{aligned}\text{Fahrenheit} &\equiv \lambda t f c. f t \\ \text{Celsius} &\equiv \lambda t f c. c t\end{aligned}$$

otrzymujemy reprezentację funkcji `fromTemperature` postaci:

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

#### 1.4.4 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  $\lambda$  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  $\lambda$  nie pozwala na okreslanie nowych nazw i rekurencyjne odnoszenie się przez nie do nich samych. Standardową techniką w rachunku  $\lambda$  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

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

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

#### 1.4.5 Kodowanie Scotta typów rekursywnych

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

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

Zwróćmy uwagę, że konstruktory **Nat** i **Maybe** są swoimi  $\alpha$ -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  $\text{undef}$  jest stałą o którą rozszerzamy rachunek  $\lambda$  celem sygnalizowania błędnej aplikacji.

Celem lepszego porównania kodowania Churcha i Scotta podamy reprezentacje funkcji  $\text{foldl}$  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]

$$f = \text{foldl } \varphi a \iff \begin{cases} f \text{ Zero} = a \\ f (\text{Succ } n) = \varphi (f n) \end{cases} \quad (3)$$

#### 1.4.6 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 oznaczają 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:

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

### 1.4.7 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 częściowej funkcji  $f$ :

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

kodujemy przy za pomocą następującego  $\lambda$ -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  $i \in \mathbb{N}$ .

## 1.5 Podsumowanie

Istotą rachunku  $\lambda$  bez typów jest uchwycenie pojęcia aplikacji argumentu do funkcji. Kodując selektor `if` dla typu `Boolean` w 1.4.2 zauważmy, że nic nie powstrzymuje nas przed zaaplikowaniem do wyrażenia `if` dowolnego  $\lambda$ -termu. Analogiczna sytuacja ma miejsce, gdy określamy operacje na reprezentacji liczb naturalnych. Widzimy, że w ramach tak zakrojonego systemu nie mamy możliwości uchwycenia które rezultaty są sensowne. Jak przekonamy się w Rozdziale 2, problem ten eliminuje w pewnym stopniu rozszerzenie systemu rachunku  $\lambda$  o typy wyrażen. Wówczas aplikacja argumentu do funkcji wymaga wcześniejszej *weryfikacji* typu, zaś typy argumentów oraz rezultatu funkcji są z góry określone (z dokładnością do podstawienia). Niestety, w rezultacie otrzymujemy system w którym wiele sensownych wyrażen możliwych do zbudowania w rachunku  $\lambda$  nie jest

poprawnych. Szukanie bogatych systemów typów, które jednocześnie nie ograniczałyby ekspresji (lub mówiąc bardziej obrazowo językiem informatyki: pozwalałyby na określenie większej ilości poprawnie zbudowanych programów ograniczając ilość tych błędnych) jest stale pojawiającym się tematem w dziedzinie teorii typów.

## 2 Rachunek $\lambda$ 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 29.** (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  $\sigma$  i  $\tau$  są typami prostymi, to  $(\sigma \rightarrow \tau)$  jest typem prostym.

Zmienne typowe nazywa się w literaturze niekiedy *stałymi typowymi*<sup>5</sup>. 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 29 można równoznacznie wyrazić przy pomocy notacji Backusa-Naura. Wówczas ma ona następującą, zwięzłą postać:

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

Późniejsze litery alfabetu greckiego ( $\sigma, \tau, \rho, \dots$ ), być może z indeksami, 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  $\sigma \rightarrow \tau \rightarrow \rho$  oraz  $\sigma \rightarrow (\tau \rightarrow \rho)$  będziemy uznawali za tożsame.

Zauważmy, że obiekty skonstruowane w myśl Definicji 29 mają strukturę drzewa binarnego. Wysokość takiego drzewa będziemy nazywali *stopniem* typu.

**Definicja 30.** (Stopień typu) Stopniem typu nazywamy następująco określoną funkcję  $\delta : \mathbb{T} \rightarrow \mathbb{N}$

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

---

<sup>5</sup>Ta raczej nieszcześliwa konwencja podkreśla fakt, że abstrakcja nie może odbywać się po zmiennych typowych.

**Definicja 31.** (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  $M : \sigma$   $\lambda$ -term  $M$  nazywamy *podmiotem* (ang. *subject*), zaś  $\sigma$  – *predykatem*<sup>6</sup>.
- (2) *Deklaracją* (ang. *declaration*) nazywamy każde stwierdzenie w którym podmiot 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 każdy napis postaci  $\Gamma \vdash M : \sigma$ , gdzie  $\Gamma$  jest kontekstem, zaś  $M : \sigma$  – stwierdzeniem.

**Definicja 32.** (1) Jeśli  $\Gamma = (x_1 : \sigma_1, \dots, x_n : \sigma_n)$ , to liniowo uporządkowany zbiór  $\text{dom } \Gamma = (x_1, \dots, x_n)$  nazywamy *dziędziną* kontekstu  $\Gamma$ , zaś  $\text{rg } \Gamma$  – *zakresem* kontekstu  $\Gamma$  oraz

$$\begin{aligned}\text{dom } \Gamma &= \{M \in \mathbf{\Lambda} \mid (x : \sigma) \in \Gamma\}, \\ \text{rg } \Gamma &= \{\sigma \in \mathbb{T} \mid (x : \sigma) \in \Gamma\}.\end{aligned}$$

- (2) Kontekst  $\Gamma'$  nazywamy *podkontekstem*  $\Gamma$  i piszemy  $\Gamma' \subseteq \Gamma$ , jeśli wszystkie deklaracje występujące w  $\Gamma'$  występują również w  $\Gamma$  z zachowaniem tego samego porządku.
- (3) Kontekst  $\Gamma'$  nazywamy *permutacją* kontekstu  $\Gamma$ , jeśli wszystkie deklaracje w  $\Gamma'$  występują w  $\Gamma$  i odwrotnie.
- (4) Jeśli  $\Gamma$  jest kontekstem i  $\Phi$  jest zbiorem  $\lambda$ -zmiennych, wówczas *projekcją*  $\Gamma$  na  $\Phi$  (symbolicznie  $\Gamma \upharpoonright \Phi$ ) nazywamy podkontekst  $\Gamma'$  kontekstu  $\Gamma$  taki, że  $\text{dom } \Gamma' = (\text{dom } \Gamma) \cap \Phi$ .
- (5) Dla kontekstów  $\Gamma = (x_1 : \sigma_1, \dots, x_k : \sigma_k)$  i  $\Gamma' = (y_1 : \rho_1, \dots, y_n : \rho_n)$  takich, że  $\text{dom}(\Gamma) \cap \text{dom}(\Gamma') = \emptyset$  *konkatenacją*  $\Gamma$  i  $\Gamma'$  nazywamy kontekst

$$\Gamma \# \Gamma' = (x_1 : \sigma_1, \dots, x_k : \sigma_k, y_1 : \rho_1, \dots, y_n : \rho_n).$$

**Przykład 12.** Niech  $\Gamma \equiv (y : \sigma, x_1 : \rho_1, x_2 : \rho_2, z : \tau, x_3 : \rho_3)$ . Wówczas:

- (1)  $\text{dom } \Gamma = (y, x_1, x_2, z, x_3)$ .
- (2)  $\emptyset \subseteq (x_1 : \rho_1, z : \tau) \subseteq \Gamma$
- (3)  $(x_1 : \rho_1, x_2 : \rho_2, x_3 : \rho_3, y : \sigma, z : \tau)$  jest permutacją  $\Gamma$ .
- (4)  $\Gamma \upharpoonright \{z, u, x_1\} = (x_1 : \rho_1, z : \tau)$ .

---

<sup>6</sup>Nazwy te historycznie sięgają prac nad semantyką formalną języków naturalnych R. Montague'a.



## 2.2 Typowanie

Wprowadzamy następujące reguły wyprowadzania typu (relacji typowalności):

$$\begin{array}{c} \Gamma \vdash x : \sigma \text{ (var),} \qquad \text{jeśli } x : \sigma \in \Gamma, \\[10pt] \frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash (MN) : \tau} \text{ (app),} \\[10pt] \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x. M) : \sigma \rightarrow \tau} \text{ (abs).} \end{array}$$

W systemie tym mamy do czynienia z wyraźnym podziałem na obiekty dwóch rodzajów:  $\lambda$ -termy i typy.  $\lambda$ -termy możemy przekształcać dwoma dualnymi operacjami:  $\lambda$ -abstrakcją i aplikacją. Rezultat operacji zależy od wyboru zmiennej wolnej, którą chcemy wyabstrahować z termu albo wyboru termu, który chcemy zaaplikować do innego termu, odpowiednio. Dlatego mówimy, że w rachunku  $\lambda$  z typami prostymi termy *zależą* od termów. Ponieważ abstrahowanie przebiega wyłącznie po zbiorze  $\lambda$ -zmiennych, mówimy, że zależność (abstrakcja) jest *pierwszego rzędu*.

**Definicja 33.** (Typowalność)

Mówimy, że  $\lambda$ -term  $M$  jest typu  $\sigma$  w kontekście  $\Gamma$ , 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  $\Gamma \vdash x : \sigma$ .
- (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  $\Gamma$ ) i pisali  $\Gamma \vdash_{\top} M : \sigma$ . O sądzie  $\Gamma \vdash M : \sigma$  będziemy wówczas mówili, że jest *wyprowadzalny*.

**Przykład 13.** (a) Niech  $\Gamma = (x : \sigma, y : \tau)$ . Pokażemy, że  $K = \lambda xy. x$  ma typ  $\sigma \rightarrow \tau \rightarrow \sigma$ . Istotnie,

$$\begin{array}{c} \frac{x : \sigma, y : \tau \vdash x : \sigma}{x : \sigma \vdash \lambda y. x : \tau \rightarrow \sigma} \text{ (abs)} \\ \frac{x : \sigma \vdash \lambda y. x : \tau \rightarrow \sigma}{\vdash \lambda xy. x : \sigma \rightarrow \tau \rightarrow \sigma} \text{ (abs)} \end{array}$$

(b) Niech  $\Gamma = (x : \tau \rightarrow \rho, y : \sigma \rightarrow \tau, z : \sigma)$ . Wówczas:

$$\begin{array}{c}
\frac{\Gamma \vdash y : \sigma \rightarrow \tau \quad \Gamma \vdash z : \sigma}{\Gamma \vdash yz : \tau} \text{ (app)} \\
\frac{\Gamma \vdash x : \tau \rightarrow \rho \quad \Gamma \vdash yz : \tau}{\Gamma \vdash x(yz) : \rho} \text{ (app)} \\
\frac{\Gamma \vdash x(yz) : \rho}{x : \tau \rightarrow \sigma, y : \sigma \rightarrow \rho \vdash \lambda z. x(yz) : \sigma \rightarrow \rho} \text{ (abs)} \\
\frac{x : \tau \rightarrow \sigma, y : \sigma \rightarrow \rho \vdash \lambda z. x(yz) : \sigma \rightarrow \rho}{x : \tau \rightarrow \rho \vdash \lambda yz. x(yz) : (\sigma \rightarrow \tau) \rightarrow \sigma \rightarrow \rho} \text{ (abs)} \\
\frac{x : \tau \rightarrow \rho \vdash \lambda yz. x(yz) : (\sigma \rightarrow \tau) \rightarrow \sigma \rightarrow \rho}{\vdash \lambda xyz. x(yz) : (\tau \rightarrow \rho) \rightarrow (\sigma \rightarrow \tau) \rightarrow \sigma \rightarrow \rho} \text{ (abs)}
\end{array}$$

- (c) Nie wszystkie  $\lambda$ -termy są typowalne. Niech  $\omega \equiv (\lambda x. xx)$  i przypuśćmy, że  $\omega$  ma typ  $\sigma$  w kontekście  $\Gamma$ . Zauważmy, że wówczas w  $x : \sigma \rightarrow \sigma \in \Gamma$ . Ponieważ  $\omega$  zawiera w sobie podterm  $(xx)$ , to w wyprowadzeniu musiał on zostać otrzymany przez zastosowanie reguły (app). Wówczas  $x : \sigma \in \Gamma$  i  $x : \sigma \rightarrow \sigma \in \Gamma$ , co nie jest możliwe, bo wszystkie deklaracje w  $\Gamma$  muszą mieć różne podmioty.

*Uwaga.* Notacja dowodowa zaproponowana w Przykładzie 13 wprowadza wiele redundancji, która utrudnia zorientowanie się w dłuższych wyprowadzeniach, zaś zlinearyzowanie dowodów pozwala na notowanie kolejnych sądów w arbitralnym porządku. Nic nie stoi na przeszkodzie abyśmy ograniczyli się tylko do rozpatrywania wyprowadzeń, w których zależności pomiędzy sądami ustalają ścisły porządek częściowy, tzn. takich, że:

- żaden sąd nie poprzedza sam siebie (antyzwrotność),
- jeśli jeden sąd poprzedza drugi, to drugi nie poprzedza pierwszego (antysymetryczność),
- jeśli sąd  $J_k$  poprzedza  $J_l$  i  $J_l$  poprzedza  $J_m$ , to  $J_k$  poprzedza  $J_m$  (przechodniość).

Taki charakter dowodów oddaje wariant notacji w postaci drzew wprowadzony przez Pravitzą i proponowany w [HS08]. Powtarzanie się kontekstów eliminuje się w niej przez wprowadzanie każdej deklaracji do dowodu przed użyciem i wykreślanie jej po użyciu. W dalszej części pracy będziemy korzystać z analogicznej pod wieloma względami notacji Fitcha (tzw. notacji flagowej). Poniżej pokazujemy wprowadzenia typu z Przykładu 13 w tej notacji.

(a)

$$\begin{array}{lcl}
1 & \left| \begin{array}{l} x : \sigma \end{array} \right. & \text{(var)} \\
2 & \left| \begin{array}{l} \left| y : \sigma \right. \end{array} \right. & \text{(var)} \\
3 & \left| \begin{array}{l} \lambda y. x : \tau \rightarrow \sigma : \sigma \end{array} \right. & \text{(abs) 1} \\
4 & \lambda xy. x : \sigma \rightarrow \tau \rightarrow \sigma & \text{(abs) 3}
\end{array}$$

(b)

1	$x : \tau \rightarrow \rho$	(var)
2	$y : \sigma \rightarrow \tau$	(var)
3	$z : \sigma$	(var)
4	$yz : \tau$	(app) 2 3
5	$x(yz) : \rho$	(app) 1 4
6	$\lambda z. x(yz) : \sigma \rightarrow \rho$	(abs) 5
7	$\lambda yz. x(yz) : (\sigma \rightarrow \tau) \rightarrow \sigma \rightarrow \rho$	(abs) 6
8	$\lambda xyz. x(yz) : (\tau \rightarrow \rho) \rightarrow (\sigma \rightarrow \tau) \rightarrow \sigma \rightarrow \rho$	(abs) 7

**Definicja 34.** (Poprawność, typowalność)  $\lambda$ -term  $M \in \mathbf{\Lambda}$  nazywamy *poprawnym* (ang. *legal*) lub *typowalnym* (ang. *typable*), jeśli istnieje wyprowadzenie  $\Gamma \vdash M : \rho$  dla pewnego kontekstu  $\Gamma$  i typu  $\rho \in \mathbb{T}$ .

### 2.2.1 Rodzaje problemów

W teorii typów spotykamy trzy rodzaje problemów dotyczące sądów:

1. *Problem typowalności* (ang. *well-typedness, typability*)

Polega na rozstrzygnięciu czy zadany term jest poprawny, czyli znalezieniu kontekstu oraz wyprowadzenia typu względem tego kontekstu dla zadanego termu. Symbolicznie:

$$? \vdash \text{term} : ?$$

Problem typowalności przy zadanym kontekście nazywamy problemem *przy-  
pisania typu* (ang. *type assignment*). Ma on następującą postać:

$$\text{kontekst} \vdash \text{term} : ?$$

2. *Problem weryfikacji typu* (ang. *type checking*)

Polega na sprawdzeniu czy term ma zadany typ względem danego kontekstu.

$$\text{kontekst} \stackrel{?}{\vdash} \text{term} : \text{typ}$$

3. *Problem inhabitacji* (ang. *inhabitation, term finding*)

Polega na skonstruowaniu termu (lub przynajmniej wykazaniu istnienia takiego termu), który miałby zadany typ względem danego kontekstu.

$$\text{kontekst} \vdash ? : \text{typ}$$

W wielu systemach problem typowalności można sprowadzić do problemu weryfikacji typu. Istotnie, przypuśćmy, że  $M$  jest poprawnym termem i  $FV(M) = \{x_1, \dots, x_n\}$ . Zauważmy, że  $M$  jest typowalny, jeśli wyprowadzalny jest sąd

$$x_0 : p \vdash Kx_0(\lambda x_1 \dots x_n. M) : p,$$

gdzie  $p \in U$  jest zmienną typową, zaś kombinatory  $K \equiv \lambda xy. x$ .

Wszystkie wymienione problemy są w rachunku  $\lambda$  z typami prostymi są rozstrzygalne<sup>7</sup>, tzn. istnieją efektywnie obliczalne metody ich rozwiązywania. Czytelnika zainteresowanego szczegółami odsyłamy do [SU06, Twierdzenie 3.2.7] i [Bar92, Rozdział 4.4].

## 2.3 Własności

Przedstawimy teraz szereg lematów ustalających między innymi związki między rachunkiem  $\lambda$  bez typów wprowadzonym w Rozdziale 1, a rachunkiem  $\lambda$  z typami prostymi.

**Lemat 3.** (*O generowaniu*)

- (1) Jeśli  $\Gamma \vdash_{\mathbb{T}} x : \sigma$ , to  $x : \sigma \in \Gamma$ .
- (2) Jeśli  $\Gamma \vdash_{\mathbb{T}} MN : \tau$ , to  $\Gamma \vdash_{\mathbb{T}} M : \sigma \rightarrow \tau$  i  $\Gamma \vdash_{\mathbb{T}} N : \sigma$  dla pewnego  $\sigma \in \mathbb{T}$ .
- (3) Jeśli  $\Gamma \vdash_{\mathbb{T}} \lambda x. M : \tau$  i  $x \notin \text{dom } \Gamma$ , to  $\tau \equiv \tau_1 \rightarrow \tau_2$  oraz  $\Gamma, x : \tau_1 \vdash_{\mathbb{T}} N : \tau_2$ .

**Dowód.** Wynika natychmiast z postaci  $\lambda$ -termu. □

**Lemat 4.** (*O podtermie*) Podterm poprawnego  $\lambda$ -termu jest poprawny.

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

- (a) Jeśli  $J$  jest konsekwencją reguły *var*, to  $\text{Sub}(M) = \{M\}$  (Definicja 2.1), a zatem teza jest trywialnie spełniona.
- (b) Jeśli  $J$  jest konsekwencją reguły *app*, to  $M \equiv PQ$  dla  $P, Q$  dla których twierdzenie zachodzi. Ponieważ  $\text{Sub}(M) = \text{Sub}(P) \cup \text{Sub}(Q) \cup \{PQ\}$  (Definicja 2.3), 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  $\text{Sub}(\lambda x. M) = \text{Sub}(M) \cup \{\lambda x. M\}$ , (Definicja 2.5) to teza zachodzi również w tym przypadku.

---

<sup>7</sup>Nie jest to bynajmniej oczywiste dla innych systemów typów; za przykład wystarczy wziąć słynny wynik J. B. Wellsa [Wel99], który stwierdza, że problemy typowalności i weryfikacji typu w Systemie F są nierozstrzygalne. Stąd w praktyce uzasadnione jest zainteresowanie mniej ekspresywnymi systemami typów, np. systemem Hindleya-Milnera.

□

**Lemat 5.** *(O zmiennych wolnych) Jeśli sąd  $J : \Gamma \vdash L : \sigma$  jest wyprowadzalny, to  $\text{FV}(L) \subseteq \text{dom } \Gamma$ .*

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

- (a) Jeśli  $J$  jest konsekwencją reguły *var*, to  $L \equiv x$  dla pewnej  $\lambda$ -zmiennnej  $x$ . Wobec tego  $x : \sigma \in \Gamma$ , a zatem  $\text{FV}(x) \subseteq \text{dom } \Gamma$ .
- (b) Jeśli  $J$  jest konsekwencją reguły *app*, to  $J$  musi mieć postać  $\Gamma \vdash MN : \sigma$ . Z założenia indukcyjnego:  $\text{FV}(M) \subseteq \text{dom } \Gamma$  i  $\text{FV}(N) \subseteq \text{dom } \Gamma$ . Z Definicji 3:  $\text{FV}(MN) = \text{FV}(M) \cup \text{FV}(N)$ . Stąd  $\text{FV}(MN) \subseteq \text{dom } \Gamma$ .
- (c) Jeśli  $J$  jest konsekwencją reguły *abs*, to  $J$  musi mieć postać  $\Gamma \vdash \lambda x. M : \sigma$ . Z założenia indukcyjnego  $\text{FV}(M) \subseteq \text{dom } \Gamma$ . Ponieważ  $\text{FV}(\lambda x. M) = \text{FV}(M) \setminus \{x\} \subseteq \text{FV}(M)$  (z Definicji 3), to  $\text{FV}(M) \subseteq \text{dom } \Gamma$ .

□

**Lemat 6.** *(1) Niech  $\Gamma'$  i  $\Gamma''$  będą kontekstami takimi, że  $\Gamma' \subseteq \Gamma''$ . Jeśli  $\Gamma' \vdash M : \sigma$ , to  $\Gamma'' \vdash M : \sigma$ .*

*(2) Jeśli  $\Gamma \vdash M : \sigma$ , to  $\Gamma \upharpoonright \text{FV}(M) \vdash M : \sigma$ .*

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

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

*Uwaga.* Zauważmy, że (6.(3)) rozluźnia konieczność eliminowania maksymalnej deklaracji z kontekstu przy stosowaniu reguły *abs*. W tekstach wprowadzających typy proste zazwyczaj przez kontekst rozumie się po prostu skończony zbiór wzajemnie różnych deklaracji [Bar92; SU06; HS08]. Ponieważ branie dowolnej permutacji listy nie wpływa na typowanie, to widzimy, że obydwie znaczenia możemy stosować zamiennie, bowiem jest to nic innego jak traktowanie kontekstu jako zbioru deklaracji.

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

*(a)  $\Gamma_1, x : \sigma, \Gamma_2 \vdash M : \rho$*

*(b)  $\Gamma_1 \vdash N : \sigma$*

*Wówczas  $\Gamma_1, \Gamma_2 \vdash M[x/N] : \rho$ .*

**Dowód.** Niech  $\Gamma = \Gamma_1 \# \Gamma_2$ . Korzystając z części (3) Lematu 6 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  $\Gamma, x : \sigma \vdash P : \tau \rightarrow \rho$  oraz  $\Gamma, x : \sigma \vdash Q : \tau$ . Z założenia indukcyjnego mamy, że  $\Gamma \vdash P[x/N] : \tau \rightarrow \rho$  oraz  $\Gamma \vdash Q[x/N] : \tau$ . Wówczas stosując regułę *app* mamy:

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

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  $\rho \equiv \sigma \rightarrow \tau$ ,  $y \neq x$ . Z założenia indukcyjnego istnieje wyprowadzenie  $\Gamma', y : \tau \vdash P[x/N] : \rho$ , gdzie  $\Gamma' = \Gamma \# (x : \sigma)$ . Wówczas, stosując regułę *abs* mamy:

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

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

□

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

- (i)  $\Gamma \vdash_{\mathbb{T}} M : \sigma$
- (ii)  $M \rightarrow_{\beta}^* N$

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

**Dowód.** Pokażemy, że twierdzenie zachodzi dla jednego kroku redukcji  $\rightarrow_{\beta}$ . Dowód zwrotności jest trywialny, zaś aby pokazać przechodniość wystarczy skorzystać z indukcji względem długości ciągu redukcji.

Niech  $M \rightarrow_{\beta} N$ . 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  $\lambda$ -zmiennnej  $x \in V$ . 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  $\Gamma \vdash P : \tau \rightarrow \sigma$  oraz  $\Gamma \vdash Q : \sigma$ . Ponadto zakładamy, że dla pewnych  $P', Q' \in \mathbf{\Lambda}$  mamy  $P \rightarrow_\beta P'$  i  $Q \rightarrow_\beta Q'$ . Istnieją dwie możliwości redukcji  $M \rightarrow_\beta N$ :

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

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

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  $\sigma \equiv \rho \rightarrow \tau$  oraz istnieje wyprowadzenie sądu  $\Gamma, x : \rho \vdash P : \tau$ . Ponadto zakładamy, że dla pewnego  $P' \in \mathbf{\Lambda}$  mamy  $P \rightarrow_\beta P'$ .  $\beta$ -redukcja  $M \rightarrow_\beta N$  musi prowadzić w tym wypadku do  $N \equiv \lambda x. P'$ . Ponieważ  $\Gamma, x : \rho \vdash P' : \tau$  (założenie indukcyjne), to możemy zastosować regułę *abs*:

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

Stąd teza. □

**Lemat 9.** (Zachowawczość  $\eta$ -redukcji) Załóżmy, że

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

(ii)  $M \rightarrow_\eta^* N$

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

**Dowód.** Dowód przeprowadzamy analogicznie do Lematu 8. □

**Twierdzenie 5.** (Własność Churcha-Rossera) Relacja  $\rightarrow_\beta$  określona na typowych  $\lambda$ -termach ma własność CR.

**Dowód.** Wynika to bezpośrednio z Twierdzenia 1 i Lematu 8. □

### 2.3.1 Uniwersalny polimorfizm

Istnieje wiele wariantów wprowadzania typów prostych. Przedstawiony w tym rozdziale rachunek powszechnie nazywany jest *stylem Currego*. Charakteryzuje go interesująca własność: poprawne termy mają jednoznacznie wyznaczony typ z dokładnością do podstawienia. Oznacza to, że zmienne występujące w typie każdego poprawnego termu są w istocie kwantyfikowane po zbiorze wszystkich typów prostych. Każdy taki typ nazywamy *typem (uniwersalnie parametrycznie) polimorficznym*.

**Definicja 35.** (Podstawienie typu) Podstawienie typu  $\tau$  za zmienną typową  $p$  w typie  $\sigma$  nazywamy następującą funkcję :

$$\begin{aligned} p[p/\tau] &= \tau, \\ q[p/\tau] &= q, \text{ jeśli } q \neq p, \\ (\sigma_1 \rightarrow \sigma_2)[p/\tau] &= \sigma_1[p/\tau] \rightarrow \sigma_2[p/\tau]. \end{aligned}$$

Jeśli  $\Gamma$  jest kontekstem, to przez  $\Gamma[p/\tau]$  oznaczamy podstawienie  $\tau$  za zmienną  $p$  dla wszystkich typów występujących w  $\Gamma$ .

**Lemat 10.** Jeśli  $\Gamma \vdash_{\mathbb{T}} M : \sigma$ , to  $\Gamma[p/\tau] \vdash_{\mathbb{T}} M : \sigma[p/\tau]$  dla dowolnego  $\tau \in \mathbb{T}$  i zmiennej  $p \in U$ .

**Dowód.** Dowód przebiega przez indukcję względem długości wyprowadzenia  $\Gamma \vdash M : \sigma$ . Szczegóły pomijamy.  $\square$

Mając na uwadze Lemat 10 możemy wnioskować o wielu własnościach funkcji reprezentowanych przez  $\lambda$ -termy tylko na podstawie typu. Na przykład typowi  $\sigma \rightarrow \sigma$  odpowiada dokładnie jeden (z dokładnością do  $\alpha$ -konwersji) term  $\lambda x.x$  i reprezentuje on funkcję identycznościową; typom  $\sigma \rightarrow \rho \rightarrow \sigma$  i  $\sigma \rightarrow \rho \rightarrow \rho$  odpowiednio przypisać możemy wyłącznie projekcje  $\text{fst}$  i  $\text{snd}$  (określone w 1.4.3), zaś typowi  $(\rho \rightarrow \tau) \rightarrow (\sigma \rightarrow \rho) \rightarrow \sigma \rightarrow \tau$  term  $\lambda g f x.g(fx)$  reprezentujący złożenie funkcji (Przykład 13 (b)), co do której wiemy z kolei, że jest łączna. Jest to wraz ogólnej zależności: dysponując dowolnym typem polimorficznym otrzymujemy *twierdzenie za darmo* [Wad89] dotyczące termów, które mają ten typ.

### 2.3.2 Silna normalizacja

Pokażemy, że wszystkie typowalne  $\lambda$ -termy redukują się do postaci  $\beta$ -normalnej przez skończony ciąg  $\beta$ -redukcji. Oznacza to, że nie ma możliwości otrzymania nieskończonego ciągu  $\beta$ -redukcji, tak jak to miało miejsce w Przykładzie 8 (a) (b) i to bez względu na przyjętą strategię redukcji.



Ponieważ wszystkie  $\lambda$ -termy samoreplikujące się przy  $\beta$ -redukcji nie są typowalne, nie jest możliwe w rachunku  $\lambda$  z typami prostymi reprezentowanie rekurencyjnych typów ADT w myśl podrozdziału 1.4.4. Wynika to z faktu, że dodanie typów prostych do rachunku  $\lambda$  bez typów znacznie zmniejsza ekspresywność systemu, uniemożliwiając wyrażenie operacji rekursji prostej. Okazuje się, że stosując reprezentację Churcha dla liczb naturalnych i utożsamiając  $\lambda$ -termy za pomocą  $\beta$ -konwersji, rachunek  $\lambda$  z typami prostymi równoważny jest zbiorowi *wielomianów rozszerzonych* [Zak07]. Liczbnikom Churcha odpowiada wówczas typ postaci  $(\sigma \rightarrow \sigma) \rightarrow \sigma \rightarrow \sigma$  i możliwe jest określenie na nich dodawania i mnożenia.

Ponieważ wszystkie ciągi redukcji są w tym systemie skończone, to relacja  $\beta$ -konwersji jest rozstrzygalna, wystarczy bowiem sprowadzić jej argumenty do postaci normalnej. Podejście to rodzi jednak nietrywialne problemy natury złożonościowej [SU06, Podrozdział 3.7].

Opracowany tutaj dowód pochodzi z [HS08, Dodatek A3]. Polega on na:

1. Konstrukcji interpretacji dla typów prostych: termów redukowalnych.
2. Wykazaniu, że każdy term redukowalny jest silnie normalizowalny.
3. Wykazaniu, że każdy typowalny term jest redukowalny.

Rozumowanie przedstawione w tym dowodzie, tzw. *computability method* oryginalnie przypisywane W. Taitowi [Tai67], z odpowiednimi zmianami stosuje się w dowodach własności silnej normalizacji dla innych systemów typów [SU06, Podrozdział 11.5].

**Definicja 36.** (Termy redukowalne) Niech  $\Gamma \vdash_{\mathbb{T}} M : \sigma$ . Powiemy, że  $M$  jest *redukowalny* (także *silnie obliczalny*), jeśli spełnia poniższe warunki:

(R1) Jeśli  $\sigma$  jest zmienną typową, to  $M$  jest silnie normalizowalny. Określamy:

$$\llbracket \sigma \rrbracket = \text{SN}.$$

(R2) Jeśli  $\sigma$  jest typem funkcyjnym postaci  $\sigma \equiv \rho \rightarrow \tau$ , to dla wszystkich termów redukowalnych  $N$  takich, że  $\Gamma' \vdash_{\mathbb{T}} MN : \tau$ ,  $MN$  jest redukowalny. Określamy:

$$\llbracket \rho \rightarrow \tau \rrbracket = \{M \mid \forall N (N \in \llbracket \rho \rrbracket) \implies MN \in \llbracket \tau \rrbracket\}.$$

**Lemat 11.** Niech  $\tau \in \mathbb{T}$  będzie dowolnym typem prostym. Wówczas:

- (1)  $\llbracket \tau \rrbracket \subseteq \text{SN}$ .
- (2) Jeśli  $N_1, N_2, \dots, N_k \in \text{SN}$ , to  $xN_1N_2 \dots N_k \in \llbracket \tau \rrbracket$ .

**Dowód.** Dowód przeprowadzimy przez indukcję strukturalną względem typu  $\tau$ . Mamy do rozważenia następujące dwa przypadki:

(a)  $\tau$  jest zmienną typową.

- (1) Wynika bezpośrednio z definicji  $\llbracket \tau \rrbracket \in \text{SN}$ .
- (2) Niech  $N_1, N_2, \dots, N_k \in \text{SN}$ . Wówczas  $N_1, N_2, \dots, N_k \in \text{SN}$ . Z definicji  $\llbracket \tau \rrbracket$  mamy, że  $xN_1N_2 \dots N_k \in \llbracket \tau \rrbracket$ .

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

- (1) Niech  $M \in \llbracket \sigma \rightarrow \rho \rrbracket$  i niech  $x$  będzie dowolną  $\lambda$ -zmienną. Z części (2) założenia indukcyjnego mamy  $x \in \llbracket \sigma \rrbracket$ , zatem z definicji  $\llbracket \sigma \rightarrow \rho \rrbracket$  mamy  $Mx \in \llbracket \rho \rrbracket$ . Ponieważ z części (1) założenia indukcyjnego  $\llbracket \rho \rrbracket \in \text{SN}$ , to  $Mx \in \text{SN}$  i w konsekwencji  $\llbracket \sigma \rightarrow \rho \rrbracket \subseteq \text{SN}$ .
- (2) Niech  $P \in \llbracket \sigma \rrbracket$ . Wówczas z części (1) założenia indukcyjnego  $P \in \text{SN}$ . Chcemy pokazać, że  $xN_1N_2 \dots N_k \in \llbracket \rho \rrbracket$ . Z części (2) założenia indukcyjnego

$$xN_1N_2 \dots N_k N_{k+1} \in \llbracket \rho \rrbracket.$$

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

□

**Lemat 12.** *Założmy, że:*

- (a)  $M[x/N_0]N_1 \dots N_k \in \text{SN}$ ,
- (b)  $N_0 \in \text{SN}$ .

Wówczas  $(\lambda x. M)N_0N_1 \dots N_k \in \text{SN}$ .

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

$$P_0 \rightarrow P_1 \rightarrow \dots$$

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

$$P_n \equiv (\lambda x. M')N'_0N'_1 \dots N'_k \rightarrow_\beta M'[x/N'_0]N'_0N'_1 \dots N'_k \equiv P_{n+1},$$

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

**Lemat 13.** *Założmy, że:*

- (a)  $M[x/N_0]N_1 \dots N_k \in \llbracket \tau \rrbracket$ ,
- (b)  $N_0 \in \text{SN}$ .

Wówczas  $(\lambda x. M)N_0N_1 \dots N_k \in \llbracket \tau \rrbracket$ .

**Dowód.** Dowód przebiega przez indukcję strukturalną względem  $\tau$ . Rozważmy następujące przypadki:

- (a) Jeśli  $\tau$  jest zmienną typową, to  $\llbracket \tau \rrbracket = \text{SN}$ . Wobec tego problem sprowadza się do Lematu 12.
- (b) Przypuśćmy, że  $\tau \equiv \sigma \rightarrow \rho$  i niech  $M[x/N_0]N_1 \dots N_k \in \llbracket \sigma \rightarrow \rho \rrbracket$ . Wybierzmy dowolny  $P \in \llbracket \sigma \rrbracket$ . Wówczas  $M[x/N_0]N_1 \dots N_k N_{k+1} \in \llbracket \rho \rrbracket$ . Z założenia indukcyjnego mamy jednak, że  $(\lambda x. M)N_0N_1 \dots N_k N_{k+1} \in \llbracket \rho \rrbracket$ . Wystarczy więc przyjąć  $N_{k+1} \equiv P$  i z definicji  $\llbracket \sigma \rightarrow \rho \rrbracket$  mamy, że  $(\lambda x. M)N_0N_1 \dots N_k \in \llbracket \sigma \rightarrow \rho \rrbracket$ .

□

**Definicja 37.** Powiemy, że kontekst  $\Gamma = \{x_1 : \sigma_1, x_2 : \sigma_2, \dots, x_n : \sigma_n\}$  spełnia stwierdzenie  $M : \sigma$  i będziemy pisali  $\Gamma \models M : \sigma$ , jeśli dla dowolnych  $N_1 \in \llbracket \sigma_1 \rrbracket$ ,  $N_2 \in \llbracket \sigma_2 \rrbracket$ ,  $\dots$ ,  $N_n \in \llbracket \sigma_n \rrbracket$  mamy, że:

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

**Lemat 14.** Jeśli  $\Gamma \vdash_{\mathbb{T}} M : \tau$ , to  $\Gamma \models M : \tau$ .

**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  $N_1 \in \llbracket \tau_1 \rrbracket$ ,  $N_2 \in \llbracket \tau_2 \rrbracket$ ,  $\dots$ ,  $N_n \in \llbracket \tau_n \rrbracket$ . 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  $M[\vec{x}/\vec{N}] = x_i[x_i/N_i] = N_i \in \llbracket \tau \rrbracket$ . Z dowolności  $N_i$ ,  $\Gamma \models M : \tau$ .
- (b)  $J$  jest konsekwencją reguły *app*. Wówczas  $J$  jest postaci  $\Gamma \vdash PQ : \tau$ . Z założenia indukcyjnego istnieje  $\sigma \in \mathbb{T}$  takie, że  $\Gamma \models P : \sigma \rightarrow \tau$  i  $\Gamma \models Q : \sigma$ . Wobec tego  $P[\vec{x}/\vec{N}] \in \llbracket \sigma \rightarrow \tau \rrbracket$  i  $Q[\vec{x}/\vec{N}] \in \llbracket \sigma \rrbracket$ . Z definicji jednoczesnego podstawienia (Definicja 10) mamy:

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

Z definicji  $\llbracket \sigma \rightarrow \tau \rrbracket$  wówczas  $M \in \llbracket \tau \rrbracket$ .

- (c)  $J$  jest konsekwencją reguły *abs*. Wówczas  $J$  jest postaci  $\Gamma \vdash \lambda y. P : \sigma \rightarrow \rho$ , gdzie  $y \notin \text{dom}\Gamma$ . Z założenia indukcyjnego mamy, że  $\Gamma, y : \sigma \models P : \rho$ . Oznacza to, że dla dowolnych  $N_1 \in \llbracket \tau_1 \rrbracket$ ,  $N_2 \in \llbracket \tau_2 \rrbracket$ , ...,  $N_n \in \llbracket \tau_n \rrbracket$  mamy

$$\forall N \in \llbracket \sigma \rrbracket \left( P[\vec{x}, y/\vec{N}, N] \in \llbracket \rho \rrbracket \right) \quad (*)$$

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

$$\forall N \in \llbracket \sigma \rrbracket \left( P'[y'/N] \in \llbracket \rho \rrbracket \right)$$

Ustalmy  $N_0 \in \llbracket \sigma \rrbracket$ . Wówczas z części (1) Lematu 11  $N_0 \in \text{SN}$ . Wobec tego z Lematu 13 wnioskujemy, że:

$$(\lambda y'. P')N_0 \in \llbracket \rho \rrbracket \quad (**)$$

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

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

Z (\*\*) i (\*\*\*) otrzymujemy

$$((\lambda y. P)[\vec{x}/\vec{N}])N_0 \in \llbracket \rho \rrbracket.$$

Ponieważ  $N_0 \in \llbracket \sigma \rrbracket$ , to z definicji  $\llbracket \sigma \rightarrow \rho \rrbracket$  mamy, że

$$(\lambda y. P)[\vec{x}/\vec{N}] \in \llbracket \sigma \rightarrow \rho \rrbracket.$$

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

□

**Twierdzenie 6.** (*O silnej normalizacji*) Jeżeli  $\Gamma \vdash_{\mathbb{T}} M : \tau$ , to  $M \in \text{SN}_{\beta}$ .

**Dowód.** Na podstawie Lematu 14, jeśli  $\Gamma \vdash_{\mathbb{T}} M : \tau$ , to  $M \in \llbracket \tau \rrbracket$ . Stosując Lemat 11 otrzymujemy tezę. □

Natychmiast widzimy, że własność silnej normalizacji pociąga za sobą własność słabej normalizacji, dlatego pomijamy dowód tej drugiej.

## 2.4 Typy w stylu Churcha

Przypisanie typu  $\lambda$ -termowi rozpoczynamy zawsze od określenia typów dla  $\lambda$ -zmiennych. Zasadniczo możemy to rozwiązać na dwa sposoby:

1. Możemy przypisać unikalny typ każdej  $\lambda$ -zmiennej przed jej wprowadzeniem. Takie podejście nazywamy *stylem Churcha* albo typowaniem *explicite*, ponieważ deklaracje typowe zmiennych występują jawnie w składni  $\lambda$ -termów. W konsekwencji w podejściu tym nie spotykamy problemu typowalności. Stąd systemy w tym stylu nazywa się również systemami typowanymi (ang. *typed systems*).
2. Inny sposób polega na nie ustalaniu typów zmiennych. Składnia  $\lambda$ -termów nie ulega wówczas zmianie, zaś o typie rozstrzyga algorytm rekonstrukcji typu. Typy w tym stylu były przedmiotem Rozdziału 2. W literaturze powszechnie nazywa się to podejście *stylem Currego* albo typowaniem *implicit*, zaś systemy w takim stylu określa się systemami *przypisywania typu* (ang. *type assignment systems*).

Obydwa podejścia dają w rezultacie takie same systemy typów [SU06, Rozdział 3.4]. Rozwiązaniem kompromisowym jest tzw. typowanie w *stylu de Brujina*<sup>8</sup> [BDS13, 1A.33] w którym nie ustala się typu wszystkich zmiennych, jednak adnotacje typowe są częścią składni (tak jak w stylu Churcha) i zależą od ustalonego kontekstu.

Zaprezentujemy teraz alternatywną składnię oraz reguły wyprowadzania typów dla systemu typów prostych w stylu Churcha. Wszystkie określenia oraz twierdzenia występujące dotychczas w Rozdziale 2 mają swoje odpowiedniki dla systemu w stylu Churcha [NG14, Rozdział 2.10]. Wyjątek stanowi Lemat 10, które jest zastąpione w tym systemie Lematem 15 o jednoznaczności typu.

### 2.4.1 Składnia

Zbiór typów  $\mathbb{T}$  definiujemy w myśl Definicji 29. Niech  $U, V$  będą przeliczalnie nieskończonymi zbiorami zmiennych przedmiotowych, odpowiednio: zmiennych typowych i ( $\lambda$ -zmiennych). Celem zdefiniowania  $\lambda$ -termów w stylu Churcha przeprowadzamy konstrukcję analogiczną do tej przedstawionej w Rozdziale 1: określamy zbiór pretermów  $\tilde{\Lambda}_{\mathbb{T}}$ , a następnie definiujemy  $\lambda$ -termy jako klasy abstrakcji  $\alpha$ -konwersji.

$$\begin{aligned}\mathbb{T} &\leftarrow U \mid (\mathbb{T} \rightarrow \mathbb{T}) \\ \tilde{\Lambda}_{\mathbb{T}} &\leftarrow V \mid (\tilde{\Lambda}_{\mathbb{T}} \tilde{\Lambda}_{\mathbb{T}}) \mid (\lambda V : \mathbb{T}. \tilde{\Lambda}_{\mathbb{T}})\end{aligned}$$

---

<sup>8</sup>W [SU06] nazywa się to podejście *nieortodoksyjnym stylem Churcha*

Zauważmy, że  $\lambda$ -termy w stylu Churcha różnią się od stylu Currego tylko w wypadku  $\lambda$ -abstrakcji. Z tą jedną modyfikacją definicje zbioru zmiennych wolnych, podstawienia,  $\beta$ - i  $\eta$ -redukcji, kontekstu i wyprowadzenia są analogiczne do tych z Rozdziałów 1 i 2.

## 2.4.2 Typowanie

$$\begin{array}{c} \Gamma \vdash x : \sigma \text{ (var),} \qquad \text{jeśli } x : \sigma \in \Gamma, \\[10pt] \frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash (MN) : \tau} \text{ (app),} \\[10pt] \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma. M) : \sigma \rightarrow \tau} \text{ (abs).} \end{array}$$

Zauważmy, że mając zadany kontekst, typ każdego poprawnego  $\lambda$ -termu jest jednoznacznie określony. Jest to istotna różnica, którą wprowadza styl Churcha. W systemach w stylu Currego termy poprawne są zamknięte ze względu na podstawienie typu. Własność, którą wyraża Lemat 15 zachodzi w nich z dokładnością do podstawienia.

**Lemat 15.** (*O jednoznaczności*) *Jeśli  $\Gamma \vdash M : \sigma$  i  $M \rightarrow_{\beta}^* N$ , to  $\Gamma \vdash N : \sigma$ .*

**Dowód.** Dowód przebiega przez indukcję względem długości wyprowadzenia  $\Gamma \vdash M : \sigma$ . Szczegóły pomijamy.  $\square$

**Przykład 14.** Zauważmy, że nie istnieje jeden typ dla reprezentacji funkcji identycznościowej. Jeśli *nat* jest stałą typową, którą reprezentujemy liczby naturalne, to identyczność na zbiorze liczb naturalnych będziemy reprezentowali termem  $\lambda x : \text{nat}. x$ , na zbiorze funkcji  $\mathbb{N} \rightarrow \mathbb{N}$ ,  $\lambda x : \text{nat} \rightarrow \text{nat}. x$  i tak dalej. Aby określić ogólną postać identyczności, musimy móc abstrahować po zbiorze typów, czyli parametryzować postać termu typem.

$$\lambda \sigma : *. \lambda x : \sigma. x,$$

gdzie symbolem  $*$  oznaczamy typ obiektów będących typami (szczegóły omówimy w Rozdziale 3). Własność tę (polimorfizm parametryczny) miał w pewnym sensie rachunek  $\lambda$  w stylu Currego (Podrozdział 2.3.1).

## 2.5 Podsumowanie

System typów, który był przedmiotem Rozdziału 2 jest najbardziej elementarnym przypadkiem typowanego rachunku  $\lambda$ <sup>9</sup>. W literaturze często spotyka się być

<sup>9</sup>Patologicznym przypadkiem jest rachunek  $\lambda$  bez typów, jeśli przyjmiemy, że wszystkie wyrażenia mają w nim dokładnie jeden typ. Argument ten często podejmowany jest na rzecz statycznie

może jeszcze prostszy, równoważny wariant typów prostych, w którym wszystkie typy buduje się wyłącznie z jednej stałej typowej. Pod pojęciem typów prostych rozumie się także szereg rozszerzeń przedstawionego przez nas systemu.

Mają one na ogół szczególny cel praktyczny: na przykład rozszerzenie o typ dla par umożliwiającą elegancką prezentację analogii między intuicjonistycznym rachunkiem zdań, typowanym rachunkiem  $\lambda$  i kategoriami kartezjańsko domkniętymi, znanej szerzej jako izomorfizm Currego-Howarda-Lambeka [GTL89, Rozdział 3.1]. Nie wpływa to jednak na samą istotę typowania, która polega na zależności termów od termów. Jak przekonamy się w Rozdziale 3 zależność tę można rozszerzyć, pozwalając typom na decydowanie o postaci termu.

### 3 System Girarda/Reynoldsa

W rozdziale tym przedstawimy rachunek  $\lambda$  drugiego rzędu w stylu Churcha. System ten wprowadzony został przez J.-Y. Girarda jako System F i w literaturze szerzej znany jest pod tą nazwą.

#### 3.1 Termy zależne od typów

**Definicja 38.** (Typy  $\mathbb{T}2$ ) Niech  $\mathbb{V}$  będzie przeliczalnie nieskończonym zbiorem zmiennych przedmiotowych. Zmienne te będziemy nazywali *zmiennymi typowymi* i oznaczali literami alfabetu greckiego ( $\alpha, \beta, \gamma, \dots$ ). Zbiór typów  $\mathbb{T}2$  Systemu F określamy w notacji BNF następującym zapisem:

$$\mathbb{T}2 \leftarrow \mathbb{V} \mid (\mathbb{T}2 \rightarrow \mathbb{T}2) \mid (\Pi \mathbb{V} : *. \mathbb{T}2)$$

Za oznaczenia metazmiennych przebiegających zbiorów typów  $\mathbb{T}2$  posłużą nam późniejsze litery alfabetu greckiego:  $\sigma, \tau, \rho, \dots$  lub następujące litery alfabetu łacińskiego:  $A, B, T$ .

**Definicja 39.** (Pretermy  $\tilde{\Lambda}_{\mathbb{T}2}$ ) Niech  $V$  będzie przeliczalnie nieskończonym zbiorem zmiennych przedmiotowych. Zmienne te będziemy nazywali *zmiennymi termowymi* i oznaczali literami alfabetu łacińskiego ( $x, y, z, \dots$ ). Zbiór pretermów  $\tilde{\Lambda}_{\mathbb{T}2}$  Systemu F określamy w notacji BNF następującym zapisem:

$$\tilde{\Lambda}_{\mathbb{T}2} \leftarrow V \mid (\tilde{\Lambda}_{\mathbb{T}2} \tilde{\Lambda}_{\mathbb{T}2}) \mid (\tilde{\Lambda}_{\mathbb{T}2} \mathbb{T}2) \mid (\lambda V : \mathbb{T}2. \tilde{\Lambda}_{\mathbb{T}2}) \mid (\lambda V : *. \tilde{\Lambda}_{\mathbb{T}2})$$

Wyrażenia postaci  $(\lambda V : *. \tilde{\Lambda}_{\mathbb{T}2})$  i  $(\tilde{\Lambda}_{\mathbb{T}2} \mathbb{T}2)$  nazywamy *polimorficzną abstrakcją* i *polimorficzną aplikacją*, odpowiednio. O zmiennej (termowej lub typowej) występującej bezpośrednio po znaku  $\lambda$  powiemy, że jest związana.

---

typowanych języków programowania.

*Uwaga.* Zakładamy, że  $V \cap \mathbb{V} = \emptyset$ .

*Konwencja.* Stosujemy standardowe konwencje notacyjne:

- Opuszczamy najbardziej zewnętrzne nawiasy,
- Aplikacja wiąże prawostronnie,
- Aplikacja  $i \rightarrow$  wiąże mocniej niż  $\lambda$ – i  $\Pi$ –abstrakcja,
- Kolejne  $\lambda$ – i  $\Pi$ –abstrakcje zmiennych tych samych typów mogą występować pod wspólnym znakiem i wiąże prawostronnie,
- Konstruktor typu  $\rightarrow$  wiąże prawostronnie.

Przykładowo:  $\Pi\alpha\beta : *. \alpha \rightarrow \beta \rightarrow \alpha \equiv \Pi\alpha : *. (\Pi\beta : *. (\alpha \rightarrow (\beta \rightarrow \alpha)))$ .

Wyrażenia  $\lambda$  ( $\lambda$ -termy) w Systemie F to klasy abstrakcji  $\alpha$ -konwersji. Konstrukcja ta jest analogiczna do zaprezentowanej w Rozdziale 1.1. Za oznaczenia metazmiennych przebiegające zbiór wyrażeń Systemu F posłużą nam późniejsze litery alfabetu łacińskiego:  $M, N, P, \dots$

Oczywistemu rozszerzeniu ulega szereg pojęć z Rozdziału 2. Poniżej umieszczamy najistotniejsze z nich.

**Definicja 40.** (Zbiór FV zmiennych wolnych) Przez  $FV(M)$  oznaczamy zbiór wszystkich wolnych zmiennych termowych i typowych występujących w  $M \in \Lambda_{T2}$ .

$$\begin{aligned} FV(x) &= x, \\ FV(\lambda x : \sigma. M) &= FV(\sigma) \cup (FV(M) \setminus \{x\}), \\ FV(MN) &= FV(M) \cup FV(N), \\ FV(\lambda\alpha : *. M) &= FV(M) \setminus \{\alpha\}, \\ FV(M\sigma) &= FV(M) \cup FV(\sigma). \end{aligned}$$

**Definicja 41.** (Multizbiór Sub podtermów pretermu)

- (1)  $Sub(x) = \{x\}$
- (2)  $Sub(MN) = Sub(M) \cup Sub(N) \cup \{M N\}$
- (3)  $Sub(M\sigma) = Sub(M) \cup \{M\sigma\}$
- (4)  $Sub(\lambda x : \alpha. M) = Sub(M) \cup \{\lambda x : \alpha. M\}$
- (5)  $Sub(\lambda\sigma : *. M) = Sub(M) \cup \{\lambda\sigma : *. M\}$

**Definicja 42.** (Podstawienie) Rozszerzamy definicję podstawienia o reguły obej-



mujące zmienne typowe.

$$\begin{aligned}
x[x/P] &= P, \\
y[x/P] &= y, \\
(MN)[x/P] &= M[x/P]N[x/P], \\
(\lambda y : \sigma. M)[x/P] &= \lambda y : \sigma. M[x/P], \text{ gdzie } y \notin FV(P) \cup \{x\}, \\
(M\sigma)[x/P] &= M[x/P]\sigma, \\
(\lambda\beta : *. M)[x/P] &= \lambda\beta : *. M[x/P], \text{ gdzie } p \notin FV(P), \\
x[\alpha/\sigma] &= x, \\
(MN)[\alpha/\sigma] &= M[\alpha/\sigma]N[\alpha/\sigma], \\
(\lambda y : \sigma. M)[\alpha/\sigma] &= \lambda y : \sigma. M[\alpha/\sigma], \\
(M\rho)[\alpha/\sigma] &= M\rho[\alpha/\sigma], \\
(\lambda\beta : *. M)[\alpha/\sigma] &= \lambda\beta : *. M[\alpha/\sigma], \text{ gdzie } \beta \notin FV(\sigma) \cup \{\alpha\}.
\end{aligned}$$

**Definicja 43.** ( $\alpha$ -konwersja) Relacją  $=_\alpha$  ( $\alpha$ -konwersji) nazywamy najmniejszą w sensie mnogościowym zgodną relację równoważności na  $\tilde{\Lambda}_{\mathbb{T}2}$  taką, że

- ( $\alpha 1$ )  $\lambda x : \sigma. M =_\alpha \lambda y : \sigma. M[x/y]$ , jeśli  $M[x/y]$  jest określone i  $y \notin FV(M)$ .
- ( $\alpha 2$ )  $\lambda\alpha : *. M =_\alpha \lambda\beta : *. M[\alpha/\beta]$ , jeśli  $\beta$  nie występuje w  $M$ .
- ( $\alpha 3$ )  $\Pi\alpha : *. M =_\alpha \Pi\beta : *. M[\alpha/\beta]$ , jeśli  $\beta$  nie występuje w  $M$ .

**Przykład 15.** Rozważmy następujące przykłady  $\alpha$ -konwertów:

$$\begin{array}{ll}
\text{a)} & \lambda\alpha : *. \lambda x : \alpha. x =_\alpha \\
& \lambda\beta : *. \lambda x : \beta. x =_\alpha \\
& \lambda\beta : *. \lambda y : \beta. y. \\
\text{b)} & \Pi\alpha : *. \alpha \rightarrow \alpha =_\alpha \\
& \Pi\beta : *. \beta \rightarrow \beta =_\alpha \\
& \Pi\beta : *. \beta \rightarrow \beta.
\end{array}$$

Odpowiednim modyfikacjom ulegają również pojęcia wprowadzone w Definicji 31 i Definicji 32.

**Definicja 44.** (Stwierdzenie, deklaracja)

1. Stwierdzeniem nazywamy każdy napis postaci  $M : \sigma$ , gdzie  $M \in \Lambda_{\mathbb{T}2}$  i  $\sigma \in \mathbb{T}2$  lub  $\sigma : *$ , gdzie  $\sigma \in \mathbb{T}2$ .
2. Deklaracją nazywamy każde stwierdzenie ze zmienną typową lub zmienną termową w miejscu podmiotu.

**Definicja 45.** ( $\lambda 2$ -kontekst, dziedzina, zakres)

(E1)  $\emptyset$  jest  $\lambda 2$ -kontekstem; oznaczamy go parą nawiasów  $()$  lub pomijamy, jeśli nie prowadzi to do niejednoznaczności.

(E2) Jeśli:

- (a)  $\Gamma$  jest  $\lambda 2$ -kontekstem,
- (b)  $\alpha \in \mathbb{V}$  jest zmienną typową taką, że  $\alpha \notin \text{dom}(\Gamma)$ ,

to  $\Gamma, \alpha : *$  jest  $\lambda 2$ -kontekstem, gdzie

$$\begin{aligned}\text{dom}(\Gamma, \alpha : *) &= (\text{dom}(\Gamma), \alpha), \\ \text{rg}(\Gamma, \alpha : *) &= \text{rg}(\Gamma) \cup \{*\}.\end{aligned}$$

(E3) Jeśli:

- (a)  $\Gamma$  jest  $\lambda 2$ -kontekstem,
- (b)  $\rho \in \mathbb{T}2$  jest typem takim, że  $\alpha \in \text{dom}(\Gamma)$  dla wszystkich  $\alpha \in \text{FV}(\rho)$ ,
- (c)  $x \in V$  jest zmienną termową taką, że  $x \notin \text{dom} \Gamma$ ,

to  $\Gamma, x : \rho$  jest  $\lambda 2$ -kontekstem, gdzie

$$\begin{aligned}\text{dom}(\Gamma, x : \rho) &= (\text{dom}(\Gamma), x), \\ \text{rg}(\Gamma, x : \rho) &= \text{rg}(\Gamma) \cup \{\rho\}.\end{aligned}$$

$\text{dom}(\Gamma)$  i  $\text{rg}(\Gamma)$  nazywamy odpowiednio *dziedziną* i *zakresem*  $\lambda 2$ -kontekstu  $\Gamma$ .

Jesli  $\Gamma = (a_{11} : a_{12}, \dots, a_{n1} : a_{n2})$  jest  $\lambda 2$ -kontekstem, to przez  $\Gamma[\alpha/\sigma]$  oznaczamy  $\lambda 2$ -kontekst, w którym jesli  $a_{i1} \in V$ , to  $a_{i2}$  zamieniamy na  $x_{i2}[\alpha/\sigma]$  dla  $1 \leq i \leq n$ .

*Uwaga.* Zauważmy, że przy powyższych rozstrzygnięciach definicyjnych nie zachodzi odpowiednik Twierdzenia 6 (3): nie możemy rozpatrywać dowolnych permutacji kontekstów w sądach  $\Gamma \vdash M : T$ , ponieważ w myśl Definicji 45 deklaracje zmiennych termowych w poprawnie zbudowanych  $\lambda 2$ -kontekstach są uzależnione od poprzedzających je deklaracji typowych.

Naturalnie zachodzi natomiast Lemat 6 (1), który oznacza tyle, że w rozszerzonym kontekście jesteśmy w stanie wyprowadzić co najmniej tyle, ile w pierwotnym. Fakt ten wyraża Lemat 16.

**Lemat 16.** *(O zwięźaniu) Jeśli*

- (a)  $\Gamma'$  i  $\Gamma''$  są kontekstami takimi, że  $\Gamma' \subseteq \Gamma''$
- (b)  $\Gamma' \vdash M : \sigma$

to  $\Gamma'' \vdash M : \sigma$ .

**Dowód.** Dowód przeprowadzamy przez indukcję względem długości wyprowadzenia sądu. Szczegóły pomijamy  $\Gamma' \vdash M : \sigma$ .  $\square$

**Przykład 16.** (a)  $\emptyset$  jest  $\lambda 2$ -kontekstem na podstawie (E1).

(b)  $\alpha : *$  jest  $\lambda 2$ -kontekstem na podstawie (E2).

(c)  $\alpha : *, x : \alpha \rightarrow \alpha$  jest  $\lambda 2$ -kontekstem na podstawie (E3). Zauważmy, że deklaracja  $\alpha : *$  wysepuje w kontekście przed  $x : \alpha \rightarrow \alpha$ .

(d)  $\alpha : *, x : \alpha \rightarrow \alpha, \beta : *$  jest  $\lambda 2$ -kontekstem na podstawie (E2).

(e)  $\Gamma \equiv (\alpha : *, x : \alpha \rightarrow \alpha, \beta : *, y : (\alpha \rightarrow \alpha) \rightarrow \beta)$  jest  $\lambda 2$ -kontekstem na podstawie (E3). Wówczas  $\text{dom}(\Gamma) = (\alpha, x, \beta, y)$  i  $\text{rg}(\Gamma) = \{*, \alpha \rightarrow \alpha, \beta, (\alpha \rightarrow \alpha) \rightarrow \beta\}$ .

## 3.2 Typowanie

Wprowadzamy następujące reguły wyprowadzania typu:

$$\begin{array}{ll}
\text{(var)} & \Gamma \vdash x : \sigma, \quad \text{jeśli } x : \sigma \in \Gamma \\
\\
\text{(app)} & \frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} \\
\\
\text{(abs)} & \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma. M) : \sigma \rightarrow \tau} \\
\\
\text{(form)} & \Gamma \vdash B : *, \quad \text{jeśli } B \in \mathbb{T}2 \text{ i } \text{FV}(B) \subseteq \text{rg } \Gamma \\
\\
\text{(\Pi-e)} & \frac{\Gamma \vdash M : (\Pi \alpha : *. A) \quad \Gamma \vdash B : *}{\Gamma \vdash MB : A[\alpha/B]} \\
\\
\text{(\Pi-i)} & \frac{\Gamma, \alpha : * \vdash M : A}{\Gamma \vdash \lambda \alpha : *. M : \Pi \alpha : *. A}
\end{array}$$

**Definicja 46.** (Poprawność, typowalność) Powiemy, że term  $M \in \mathbf{\Lambda}_{\mathbb{T}2}$  jest *poprawny* lub *typowalny*, jeśli istnieje  $\lambda 2$ -kontekst  $\Gamma$  i typ  $\rho \in \mathbb{T}2$  taki, że  $\Gamma \vdash M : \rho$ .

**Przykład 17.** (a) Niech  $\perp \equiv \Pi \sigma : *. \sigma$  i  $\Gamma = (\beta : *, x : \perp)$ .

1	$\beta : *$	(form)
2	$\mid x : \perp$	(var)
3	$x\beta : \beta$	( $\Pi$ -e) 2 1
4	$\lambda\beta : *. x\beta : \underbrace{\Pi\beta : *. \beta}_{\perp}$	( $\Pi$ -i) 3

(b)  $\Gamma = (\beta : *, y : \beta, x : \perp)$ .

1	$\beta : *$	(form)
2	$\mid y : \beta$	(var)
3	$\mid \mid x : \perp$	(var)
4	$\mid \lambda x : \perp . y : \perp \rightarrow \beta$	(abs) 2
5	$\lambda y : \beta . x : \perp . y : \perp \rightarrow \beta$	(abs) 4
6	$\lambda\beta : *. \lambda y : \beta . x : \perp . y : \perp \rightarrow \beta$	( $\Pi$ -i) 1

(c) Przykład 14 ilustrował, że typy proste nie pozwalają na określenie polimorficznej identyczności. W Systemie F nie sprawia to problemów.

1	$\alpha : *$	(form)
2	$\mid x : \alpha$	(var)
3	$\lambda x : \alpha . x : \alpha \rightarrow \alpha$	(abs) 2
4	$\lambda\alpha : *. \lambda x : \alpha . x : \Pi\alpha : *. \alpha \rightarrow \alpha$	( $\Pi$ -i) 3

(d) W systemie Girarda/Reynoldsa możemy określić polimorficzne liczebniki Churcha ustalając:

$$\begin{aligned}
\text{Nat} &\equiv \Pi\alpha : *. (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha \\
\bar{0} &\equiv \lambda\alpha : *. \lambda f : \alpha \rightarrow \alpha . x : \alpha . x \\
\bar{1} &\equiv \lambda\alpha : *. \lambda f : \alpha \rightarrow \alpha . x : \alpha . fx \\
&\vdots \\
\bar{n} &\equiv \lambda\alpha : *. \lambda f : \alpha \rightarrow \alpha . x : \alpha . \underbrace{f \dots (fx)}_{\text{n-razy}}
\end{aligned}$$

Dla przykładu rozważmy typowanie  $\bar{2}$ :

1	$\alpha : *$	(form)
2	$f : \alpha \rightarrow \alpha$	(var)
3	$x : \alpha$	(var)
4	$fx : \alpha$	(app) 2 3
5	$f(fx) : \alpha$	(app) 2 1
6	$\lambda x : \alpha. f(fx) : \alpha \rightarrow \alpha$	(abs) 5
7	$\lambda f : \alpha \rightarrow \alpha. \lambda x : \alpha. f(fx) : (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha$	(abs) 6
8	$\lambda \alpha : *. \lambda f : \alpha \rightarrow \alpha. \lambda x : \alpha. f(fx) : \underbrace{\Pi \alpha : *. (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha}_{\text{Nat}}$	( $\Pi$ -i) 7

Za pomocą łatwej indukcji względem długości wyprowadzenia możemy przekonać się, że w istocie wszystkie polimorficzne liczebniki Churcha są typu Nat.

(e) Typować możemy również polimorficzne złożenie funkcji.

1	$\gamma : *$	(form)
2	$\beta : *$	(form)
3	$\alpha : *$	(form)
4	$g : \beta \rightarrow \gamma$	(var)
5	$f : \alpha \rightarrow \beta$	(var)
6	$x : \alpha$	(var)
7	$fa : \beta$	(app) 5 6
8	$g(fa) : \gamma$	(app) 4 7
9	$\lambda x : \alpha. g(fa) : \alpha \rightarrow \gamma$	(abs) 8
10	$\lambda f : \alpha \rightarrow \beta x : \alpha. g(fa) :$ $(\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \gamma$	(abs) 9
11	$\lambda g : \beta \rightarrow \gamma f : \alpha \rightarrow \beta x : \alpha. g(fa) :$ $(\beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \gamma$	(abs) 10
12	$\lambda \alpha : *. \lambda g : \beta \rightarrow \gamma f : \alpha \rightarrow \beta x : \alpha. g(fa) :$ $\Pi \alpha : *. (\beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \gamma$	(abs) 11
13	$\lambda \beta \alpha : *. \lambda g : \beta \rightarrow \gamma f : \alpha \rightarrow \beta x : \alpha. g(fa) :$ $\Pi \beta \alpha : *. (\beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \gamma$	(abs) 12
14	$\lambda \gamma \beta \alpha : *. \lambda g : \beta \rightarrow \gamma f : \alpha \rightarrow \beta x : \alpha. g(fa) :$ $\Pi \gamma \beta \alpha : *. (\beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \gamma$	(abs) 13

Jak widzimy procedura wyprowadzania typu odpowiada w istocie rekonstrukcji wyrażenia. Problem typowania w (ortodoksyjnym) stylu Churcha jest trywialny, gdyż zgodnie z Rozdziałem 2.4 założyliśmy, że mamy pełną informację typową o zmiennych przedmiotowych. Czytając wyprowadzenia od dołu do góry rekonstruujemy również kontekst. Mówimy, że składnia Systemu F w stylu Churcha *koduje typowanie*.

### 3.3 Redukcja

**Definicja 47.** ( $\beta$ -redukcja)  $\beta$ -redukcją nazywamy najmniejszą (w sensie mnogościowym) zgodną na  $\Lambda_{T2}$  relację binarną  $\rightarrow_\beta$  taką, że

$$\begin{aligned} (\lambda x : \sigma. M)N &\rightarrow_\beta M[x/N]. \\ (\lambda \alpha : *. M)T &\rightarrow_\beta M[\alpha/T] \end{aligned}$$

Lemat 17 (podobnie jak Lemat 15) zapewnia nas, że typ termu zachowuje się przy  $\beta$ -redukcji. Praktyczny sens tego twierdzenia polega na tym, że relacja posiadania tego samego typu jest zamknięta na  $\beta$ -konwersję i aby rozstrzygnąć o typie danego termu wystarczy, że znamy typ któregośkolwiek jego  $\beta$ -konwersa.

**Lemat 17.** (*O redukcji podmiotu*) Jeśli  $\Gamma \vdash M : \tau$  i  $M \rightarrow_{\beta}^* N$ , to  $\Gamma \vdash N : \tau$ .

**Dowód.** Dowód przeprowadzamy analogicznie do dowodu Lematu 8, przez indukcję względem długości wyprowadzenia sądu  $\Gamma \vdash M : \tau$ . Szczegóły pomijamy.  $\square$

**Przykład 18.** Wróćmy do Przykładu 17 (d). Wyprowadziliśmy w nim sąd:

$$\vdash \lambda\alpha : *. \lambda f : \alpha \rightarrow \alpha. \lambda x : \alpha. f(fx) : \Pi\alpha : *. (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha$$

Stosując lemat o zwężaniu (Lemat 16), dla dowolnego  $\lambda 2$ -kontekstu  $\Gamma$  wyprowadzalny jest również sąd:

$$\Gamma \vdash \lambda\alpha : *. \lambda f : \alpha \rightarrow \alpha. \lambda x : \alpha. f(fx) : \Pi\alpha : *. (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha \quad (1)$$

Niech  $\Gamma$  będzie takim  $\lambda 2$ -kontekstem, że

$$\{\text{nat} : *, \text{zero} : \text{nat}, \text{succ} : \text{nat} \rightarrow \text{nat}\} \subseteq \Gamma,$$

gdzie przez  $\text{nat}$ ,  $\text{zero}$  i  $\text{succ}$  oznaczamy dowolnie wybrane zmienne termowe i typowe, odpowiednio. Stosując regułę ( $\Pi$ -e) do (1) i  $\Gamma \vdash \text{nat} : *$  otrzymujemy:

$$\Gamma \vdash \lambda f : \text{nat} \rightarrow \text{nat}. \lambda x : \text{nat}. f(fx) \text{ nat} : (\text{nat} \rightarrow \text{nat}) \rightarrow \text{nat} \rightarrow \text{nat} \quad (2)$$

Dalej, stosując regułę ( $\text{app}$ ) do (2) i  $\Gamma \vdash \text{succ} : \text{nat} \rightarrow \text{nat}$  mamy, że

$$\Gamma \vdash \lambda x : \text{nat}. f(fx) \text{ nat succ} : \text{nat} \rightarrow \text{nat} \quad (3)$$

Ponownie aplikując (3) do  $\Gamma \vdash \text{zero} : \text{nat}$  mamy:

$$\Gamma \vdash f(fx) \text{ nat succ zero} : \text{nat} \quad (4)$$

Na podstawie lematu o redukcji podmiotu (Lemat 17) widzimy, że każdy kolejny redukt w poniższym ciągu ma typ  $\text{nat}$ .

$$\begin{aligned} \lambda\alpha : *. \lambda f : \alpha \rightarrow \alpha. \lambda x : \alpha. f(fx) \text{ nat succ zero} &\rightarrow_{\beta} \\ \lambda f : \text{nat} \rightarrow \text{nat}. \lambda x : \text{nat}. f(fx) \text{ succ zero} &\rightarrow_{\beta} \\ (\lambda x : \text{nat}. \text{succ}(\text{succ } x)) \text{ zero} &\rightarrow_{\beta} \\ \text{succ}(\text{succ zero}) & \end{aligned}$$

Odpowiedniej modyfikacji ulega również pojęcie  $\eta$ -redukcji.

**Definicja 48.** ( $\eta$ -redukcja)  $\eta$ -redukcją nazywamy najmniejszą (w sensie mnożącym) zgodną na  $\Lambda_{\mathbb{T}2}$  relację binarną  $\rightarrow_\beta$  taką, że

$$\begin{aligned}\lambda x : \sigma. Mx &\rightarrow_\eta M, \\ \lambda \alpha : *. M\alpha &\rightarrow_\eta M.\end{aligned}$$

Zachodzi oczywiście również odpowiednik Lematu 9:

**Lemat 18.** (*Zachowawczość  $\eta$ -redukcji*) Jeśli  $\Gamma \vdash M : \tau$  i  $M \rightarrow_\eta^* N$ , to  $\Gamma \vdash N : \tau$ .

**Dowód.** Dowód przeprowadzamy przez indukcję względem długości wyprowadzenia sądu  $\Gamma \vdash M : \tau$ . Szczegóły pomijamy.  $\square$

**Przykład 19.** Zwróćmy ponownie uwagę na Przykład 17(a). Ponieważ  $\lambda\beta : *. x\beta \rightarrow_\eta x$ , to ze względu Lemat 18 wyprowadzalny jest również  $\Gamma \vdash x : \perp$ .

W odpowiednich wersjach zachodzą również Lemat 3 o generowaniu, Lemat 4 o podtermie, Lemat 5 o zmiennych wolnych i Lemat 7 o podstawieniu. We wszystkich przypadkach metoda dowodowa nie odbiega znacząco od odpowiedników z Rozdziałów 1 i 2.

Kolejnym ważnym twierdzeniem, które zachodzi dla Systemu F jest silna normalizacja  $\beta$ -redukcji.

**Twierdzenie 7.** (*O silnej normalizacji*) Jeżeli  $\Gamma \vdash_{\mathbb{T}} M : \tau$ , to  $M \in \text{SN}_\beta$ .

**Dowód.**

**Wniosek.** Ponieważ wszystkie typowalne termny w Systemie F mają skończone ciągi redukcji, to kombinatory punktu stałego

$$\text{fix} : \Pi \alpha : *. (\alpha \rightarrow \alpha) \rightarrow \alpha$$

nie jest typowalny w tym systemie.

Jako Fakt 7 przedstawimy własność Churcha-Rossera upewniającą nas, że  $\beta$ -redukcja, bez względu na przyjętą strategię, zawsze prowadzi do tych samych rezultatów, a zatem ma sens obliczeniowy. Fakt ten otrzymujemy jako wniosek z lematu Newmana 2, wykazując, że  $\beta$ -redukcja ma słabą własność Churcha-Rossera (WCR) [SU06, Twierdzenie 11.2.12].

**Fakt 7.** (*Własność Churcha-Rossera*) Relacja  $\beta$ -redukcji ma własność CR.



### 3.4 Podsumowanie

```

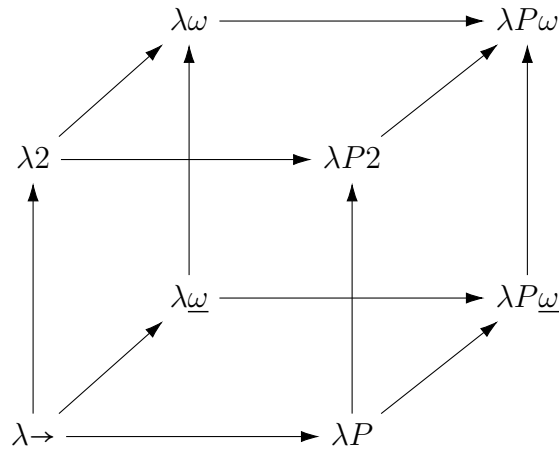
module Fnord where
  comp = \g f x -> g (f x)

comp
  :: forall t_aqz t1_aqB t2_aqD.
    (t_aqz -> t1_aqB) -> (t2_aqD -> t_aqz) -> t2_aqD -> t1_aqB

comp =
  \ (@ t_aqz)
    (@ t1_aqB)
    (@ t2_aqD)
    (g_aqg :: t_aqz -> t1_aqB)
    (f_aqh :: t2_aqD -> t_aqz)
    (x_aqi :: t2_aqD) ->
    g_aqg (f_aqh x_aqi)

```

Na kanwie zaproponowanej przez H. P. Barendregta w [Bar92, Rozdział 5] klasyfikacji rozszerzeń rachunku  $\lambda$  z typami prostymi (tzw. *kostki*  $\lambda$ , Rysunek 4), rozdział ten poświęcimy omówieniu wzajemnych zależności jakie mogą łączyć  $\lambda$ -termy i typy. Zajmować będziemy się wyłącznie systemami w stylu Churcha.



Rysunek 4: Poszczególne systemy klasyfikacji H. Barendregta; kierunek krawędzi  $\rightarrow$  oznacza relację  $\subseteq$ .

## Literatura

- [Alt02] Thorsten Altenkirch. “ $\alpha$ -conversion is easy”. Under Revision. 2002. URL: <https://www.cs.nott.ac.uk/~psztxa/publ/alpha-draft.pdf>.
- [Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Elsevier, 1984.
- [Bar92] H. P. Barendregt. “Lambda Calculi with Types”. In: vol. 2. Jan. 1992, pp. 117–309. ISBN: 0198537611.
- [BDS13] Henk Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in Logic. Cambridge University Press, 2013. DOI: 10.1017/CB09781139032636.
- [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.
- [GTL89] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. New York, NY, USA: Cambridge University Press, 1989. ISBN: 0-521-37181-3.
- [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  $\lambda$ -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](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>.
- [Mar96] P Martin Lof. “On the Meanings of the Logical Constants and the Justifications of the Logical Laws”. In: *Nordic Journal of Philosophical Logic* 1 (Jan. 1996).
- [NG14] Rob Nederpelt and Herman Geuvers. *Type Theory and Formal Proof: An Introduction*. Cambridge University Press, 2014. DOI: 10.1017/CB09781139567725.
- [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.
- [Tai67] W. W. Tait. “Intensional Interpretations of Functionals of Finite Type I”. In: *Journal of Symbolic Logic* 32.2 (1967), pp. 198–212. DOI: 10.2307/2271658.
- [Wad89] Philip Wadler. “Theorems for Free!” In: *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture*. FPCA ’89. Imperial College, London, United Kingdom: ACM, 1989, pp. 347–359. ISBN: 0-89791-328-0. DOI: 10.1145/99370.99404. URL: <http://doi.acm.org/10.1145/99370.99404>.
- [Wel99] J. B. Wells. “Typability and type checking in system F are equivalent and undecidable”. English. In: *Annals of Pure and Applied Logic* 98.1-3 (June 1999), pp. 111–156. ISSN: 0168-0072.
- [Wit53] Ludwig Wittgenstein. *Philosophical Investigations*. Wiley-Blackwell, 1953.

- [Zak07] Mateusz Zakrzewski. “Definable functions in the simply typed lambda-calculus”. In: *CoRR* abs/cs/0701022 (2007). arXiv: cs/0701022. URL: <http://arxiv.org/abs/cs/0701022>.