Spis treści

1 Rachunek λ bez typów

1.1 Język rachunku λ i λ-pretermy

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Definicja 2. (Multizbiór Sub podtermów pretermu)

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

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

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

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

Przykład 2. Podtermy wybranych λ-pretermów.

$$\text{(a)}\quad \mathrm{Sub}\,(\lambda x.\,x\,x) = \left\{(\lambda x.\,x\,x)^1,\ (x\,x)^1,\ x^2\right\}$$

$$\text{(b)}\quad \mathrm{Sub}\,((\lambda x.\,x\,x)\ (\lambda x.\,x\,x)) =$$
$$= \left\{((\lambda x.\,x\,x)\,(\lambda x.\,x\,x))^1,\ (\lambda x.\,x\,x)^2,\ (x\,x)^2,\ x^4\right\}$$

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

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

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

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

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

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

(c) $\mathrm{FV}(\lambda x y z.\,x y) = \varnothing$

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

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

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

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

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

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

Operacja podstawienia wymaga jednak pewnej delikatności. Rozważmy następującą relację:

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

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

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

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

Definicja 4'. *(Podstawienie')*

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

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

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

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

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

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

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

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

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

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

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

i) M jest zmienną. Wówczas:

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

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

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

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

$$P[x/N][y/L] = P[y/L]\,[x/N\,[y/L]]\,,$$
$$Q[x/N][y/L] = Q[y/L]\,[x/N\,[y/L]]\,.$$

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

iii) Jeśli $M \equiv \lambda z.\, P$ oraz $z \equiv x$ lub $z \equiv y$, to z (S'5) widzimy, że obydwie strony (1) sa identyczne. Przypuśćmy, że $z \not\equiv x$ i $z \not\equiv y$ i $z \notin \mathrm{FV}(L)$. Wówczas na podstawie 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 $y \notin \mathrm{FV}(M)$, to $M[x/y][y/x]$ jest określone oraz $M[x/y][y/x] = M$.*

Dowód. Mając na uwadze Lemat 1 dowód przebiega przez indukcję strukturalną względem M. □

1.2 Wyrażenia λ i ich własności

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

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

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

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

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

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

Przykład 4.

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

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

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

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

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

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

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

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

v) Jeśli $M =_\alpha N$ w konsekwencji (α3) 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 (α3) $N =_\alpha M$.

vi) Jeśli $M =_\alpha N$ w konsekwencji ($\alpha 4$), to postępujemy jak w przypadku (v). □

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

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

Konwencja. Wprowadzamy następujące konwencje notacyjne:

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

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

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

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

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

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

ii) Jeśli M jest aplikacją, to wówczas $M \equiv P_0 P_1 \ldots P_m$, 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 postaci $M \equiv \lambda x_1\, x_2 \ldots x_n.\, P_0 P_1 \ldots P_m$, to wówczas mamy następujące przypadki:

   (a) Jeśli P_0 jest zmienną, to M jest postaci (1).

   (b) Jeśli P_0 jest aplikacją, to $P_0 \equiv P_0' P_0''$, gdzie P_0' jest albo zmienną (wówczas M jest postaci (1)) albo λ-abstrakcją (wówczas M jest postaci (2)).

   (c) Jeśli P_0 jest abstrakcją, to M jest postaci (2).

□

Definicja 7 (Postać HNF, WHNF)**.** Niech $M \in \mathbf{\Lambda}$. Powiemy, że M jest w:

1. *czołowej postaci normalnej* (ang. *head normal form*), jeśli

$$M \equiv \lambda y_1\, y_2\, \ldots\, y_n.\ x M_1 M_2 \ldots M_m \quad \text{dla } n, m \geq 0.$$

2. *słabej czołowej postaci normalnej* (ang. *weak head normal form*), jeśli dla $n \geq 0$ i $x \in V$ λ-term M jest postaci i) lub ii).

   i) $x\, P_1\, P_2 \ldots P_n$,
   ii) $\lambda x.\, P_1\, P_2 \ldots P_n$.

Z określenia HNF widzimy, że każdy λ-term w postaci HNF jest również w postaci WHNF, ale nie odwrotnie.

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

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

Definicja 8. (Multizbiór Sub podtermów λ-termu) Dla dowolnego λ-termu $M = [M']_{=_\alpha}$ określamy

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

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

Definicja 9. (Zbiór zmiennych wolnych FV) Dla dowolnego λ-termu $M = [M']_{=_\alpha}$ określamy zbiór $\mathrm{FV}(M)$ *zmiennych wolnych* w M

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

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

Definicja 10. (Podstawienie) Niech $M = [M']_{=_\alpha}$ i $N = [N']_{=_\alpha}$ i niech $M'[x/N']$ będzie określone w myśl Definicji 4. Wówczas

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

Definicja 11. (Podstawienie jednoczesne) Dla dowolnego $M \in \mathbf{\Lambda}$, nieskończonego ciągu λ-zmiennych $\vec{x}$ i nieskończonego ciągu λ-termów $\vec{N}$ określamy:

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

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

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

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

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

Przykład 6. Zauważmy, że podstawienia w myśl Definicji 4 i Definicji 11 mogą, ale nie muszą, prowadzić do różnych rezultatów.

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

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

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

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

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

Szerszy komentarz dotyczący dotychczasowych prób uchwycenia operacji podstawienia można prześledzić w [Alt02]. Nasze rozważania opierają się w tej materii przeważająco na [SU06]. Technikalia definiowania λ-termow jako klas α-konwersji są na ogół w literaturze pomijane.

1.3 Semantyka operacyjna

1.3.1 Redukcje

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

Definicja 12. (Relacja zgodna) Relację binarną $\mathcal{R}$ na zbiorze $\mathbf{\Lambda}$ nazywamy *zgodną*, jeśli dla dowolnych M, N, $P \in \mathbf{\Lambda}$ zachodzą następujące warunki:

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

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

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

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

(a) Symbolem $\mathcal{R}^+$ oznaczamy domknięcie relacji $\mathcal{R}$ względem przechodniości (*przechodnie* domknięcie).

(b) Symbolem $\mathcal{R}^*$ oznaczamy domknięcie relacji $\mathcal{R}^+$ względem zwrotności (*zwrotne* domknięcie).

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

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

Definicja 13. Niech $\to$ będzie relacją binarną w zbiorze A.

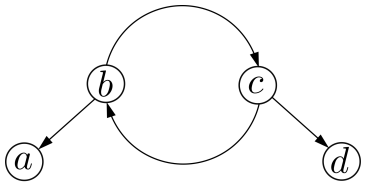
(CR) Powiemy, że $\to$ ma *własność Churcha-Rossera*, jeśli dla dowolnych a, b, $c \in A$ takich, że $a \to^* b$ oraz $a \to^* c$ istnieje $d \in A$ takie, że $b \to^* d$ i $c \to^* d$. Własność tę przedstawia poniższy diagram:

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

(WCR) Powiemy, że $\to$ ma *słabą własność Churcha-Rossera*, jeśli dla dowolnych a, b, $c \in A$ takich, że $a \to b$ oraz $a \to c$ istnieje $d \in A$ takie, że $b \to^* d$ i $c \to^* d$. Własność tę przedstawia poniższy diagram:

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

Widzimy, że własność CR pociąga za sobą własność WCR. Odwrotna zależność nie zachodzi (patrz Rysunek 1).



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

Definicja 14. (Postać normalna) Powiemy, że $x \in A$ jest *redukowalny*, jeśli istnieje $y \in A$ takie, że $x \to y$. W przeciwnym wypadku powiemy, że x jest w *postaci normalnej* i będziemy pisali $x \in \mathrm{NF}$.

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

Uwaga. Zbiór λ-termów w postaci normalnej względem β-redukcji będziemy oznaczali symbolem NF_β.

Definicja 15. Niech $\to$ będzie relacją binarną na zbiorze A.

(WN) Powiemy, że element $a \in A$ jest *słabo normalizowany* i pisali $a \in \mathrm{WN}$, jeśli istnieje $a' \in \mathrm{NF}$ taki, że $a \to^* a'$. Jeśli każdy element z dziedziny relacji $\to$ jest słabo normalizowalny, to o relacji $\to$ powiemy, że jest *słabo normalizująca.*

(SN) Powiemy, że element $a \in A$ jest *silnie normalizowalny* i pisali $a \in \mathrm{SN}$, jeśli dla pewnego $a' \in \mathrm{NF}$ każdy ciąg relacji $a \to a_1 \to a_2 \to \cdots \to a'$ jest skończony. Jeśli każdy element z dziedziny relacji $\to$ jest silnie normalizowalny, to o relacji $\to$ powiemy, że jest *silnie normalizująca.*

Uwaga. Zbiory λ-termów słabo i silnie normalizowalnych względem β-redukcji będziemy oznaczali symbolami WN_β i SN_β odpowiednio.

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

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

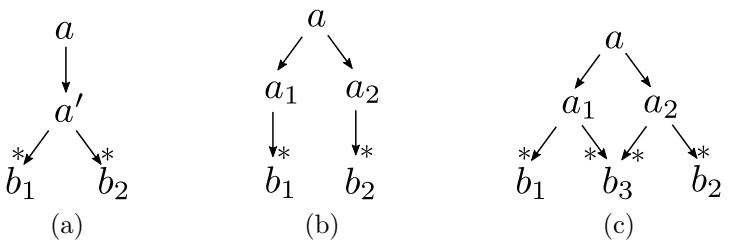
Jeśli A nie zawiera elementów niejednoznacznych, to twierdzenie zachodzi w sposób trywialny. Przypuśćmy, że $a \in A$ jest niejednoznaczny. Twierdzimy, że istnieje $a' \in A$ taki, że $a \to a'$ i a' jest niejednoznaczny. Niech $b_1, b_2 \in \mathrm{NF}$, $b_1 \neq b_2$ i $a \to^* b_1$ oraz $a \to^*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

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

Zachodzą więc dwa przypadki:

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

ii) $a_1 \neq a_2$ (Rysunek 2b). Wówczas z WCR istnieje $b_3 \in A$ takie, że $a_1 \to^* b_3$ oraz $a_2 \to^* b_3$ (Rysunek 2c). Przypuśćmy, że $b_3 \in \mathrm{NF}$. Ponieważ $b_1 \neq b_2$, więc $b_3 \neq b_1$ 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 skontruować nieskończony ciąg redukcji, wbrew założeniu, że relacja $\to$ jest SN. Zatem A nie zawiera elementów niejednoznacznych. □

Definicja 16. (β-redukcja) β-redukcją nazywamy najmniejszą (w sensie mnogościowym) zgodną na $\mathbf{\Lambda}$ relację binarną $\to_\beta$ taką, że

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

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

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

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

Przykład 7. Wszystkie pary λ-termów ze zbioru

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

są swoimi β-konwersami. Mamy:

$$(\lambda y.\, yv)z \to_\beta zv \ \leftarrow_\beta (\lambda x.\, zx)v,$$
$$(\lambda y.\, yv)z \leftarrow_\beta (\lambda x.\,(\lambda y.\, yx)z)v \to_\beta (\lambda x.\, zx)v.$$

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

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

Konwencja. Składnię rachunku λ często rozszerza się o wyrażenia let pozwalające konstruować β-redeksy w czytelny sposób. Rozszerzenie ma następującą postąć:

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

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*[1] (ang. *closure*). Nieformalnie, pozwalają one na przypisywanie wartości zmiennym o tzw. *zakresie leksykalnym*.

Definicja 17. (Strategia redukcji) Strategią redukcji nazywamy każde odwzorowanie $S : \mathbf{\Lambda} \to \mathbf{\Lambda}$, które dla $M \in \mathbf{\Lambda}$ spełnia następującą równość:

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

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

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

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

i w konsekwencji $\mathrm{Y}F =_\beta F(\mathrm{Y}F)$. Y nazywamy *kombinatorem punktu stałego*. Widzimy, że relacja β-redukcji w rachunku λ nie jest ani słabo, ani silnie normalizująca.

[1] Idiom ten w literaturze poświęconej językom programowania z rodziny Lisp występuje również pod nazwą *let-over-lambda*.

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

$$\Omega \to_\beta \Omega \to_\beta \Omega \to_\beta \dots$$

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

$$\Delta\Delta \to_\beta \Delta\Delta\Delta \to_\beta \Delta\Delta\Delta\Delta \to_\beta \dots$$

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

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

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

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

Definicja 18. (η-redukcja) η-redukcją nazywamy najmniejszą (w sensie mnogościowym) zgodną na $\mathbf{\Lambda}$ relację binarną $\to_\eta$ taką, że

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

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

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

Widzieliśmy, że konsekwentne β-redukowanie λ-termów nie zawsze prowadzi do uzyskania postaci normalnej. Fakt 1 i następujące po nim Wniosek 3 i Wniosek 4 stwierdzają, że jeśli tylko mamy pewność, że λ-term ma postać normalną, to jest ona wyznaczona jednoznacznie i doprowadzi nas do niej każda strategia normalizująca. Fakt 1 to klasyczne twierdzenie, którego dowód można znaleźć w [Bar84, Rozdział 3.2] i ze względu na jego obszerność pozwalamy sobie go pominąć.

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

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

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

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

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

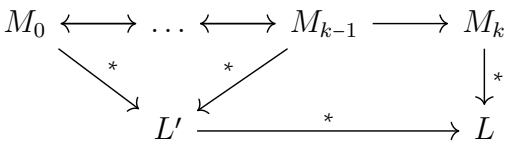
(1) Jeśli $n = 0$, to $M \equiv N$. Ustalając $L \equiv M (\equiv N)$ w oczywisty sposób $M \twoheadrightarrow_\beta^* L$ i $N \twoheadrightarrow_\beta^* L$.

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

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

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

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



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

□

Wniosek 4.

Każdy λ-term ma co najwyżej jedną postać normalną.

Dowód.

Przypuśćmy, że M ma dwie różne postacie normalne, N_1, N_2. Wówczas na podstawie Definicji 14, $M \twoheadrightarrow_\beta^* N_1$ i $M \twoheadrightarrow_\beta^* N_2$. Z Faktu 1 istnieje $L \in \mathbf{\Lambda}$ taki, że $N_1 \twoheadrightarrow_\beta^* L$ i $N_2 \twoheadrightarrow_\beta^* L$. Ponieważ $N_1, N_2 \in \mathrm{NF}_\beta$, to $N_1 \equiv L \equiv N_2$. □

1.3.2 Strategie redukcji

Przedmiotem tego podrozdziału jest przedstawienie najczęściej spotykanych klasyfikacji obliczeń w rachunku λ. Na gruncie tego formalizmu nakreślimy czym jest szeroko stosowany w funkcyjnych językach programowania *leniwy* sposób wykonywania programów.

Definicja 19. Strategię nazywamy:

1. *normalną* (ang. *normal-order*), gdy zawsze redukujemy redeksy pojedyńczo, rozpoczynając od najbardziej zewnętrznego redeksu od lewej strony wyrażenia; strategię tę nazywa się również *lewostronną najbardziej zewnętrzną* (ang. *leftmost outermost reduction*).

2. *aplikatywną* (ang. *applicative-order*), jeśli zawsze redukujemy redeksy pojedyńczo, rozpoczynając od najbardziej zagnieżdżonego redeksu występującego najbardziej po lewej stronie wyrażenia; strategię tę nazywa się również *lewostronną najbardziej wewnętrzną* (ang. *leftmost innermost reduction*).

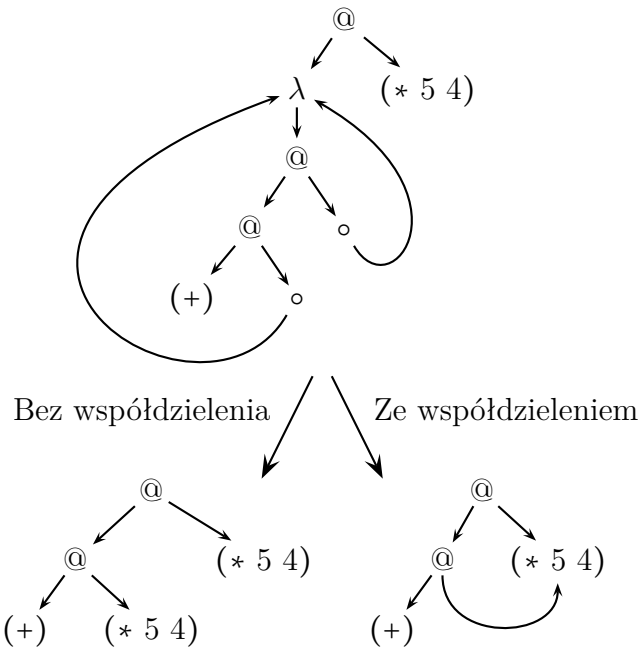
Strategia aplikatywna jest rodzajem strategii *ścisłej* (nazywanej również *zachłanną*, w angielskojęzycznej literaturze określanej odpowiednio *strict*, *gready* lub *eager*). Strategiami zachłannymi nazywamy wszystkie te redukcje, w których argumenty λ-abstrakcji są redukowane do postaci normalnej przed zaaplikowaniem ich. Strategie, które nie są ścisłe, nazywamy *strategiami nieścisłymi* (ang. *non-strict*) – argumenty λ-abstkracji w strategiach tego rodzaju mogą być redukowane dopiero wówczas, gdy jest to konieczne.

W przeciwieństwie do strategii aplikatywnej (patrz Przykład 8(d)), okazuje się, że strategia normalna jest strategią normalizującą [SU06, Rozdział 1.5]. Niestety, narzuca ona w pesymistycznym przypadku wykładniczą złożoność obliczeniową. Zobserwujmy następującą redukcję:

$$\begin{aligned}(\lambda x.(+\ x\ x))\,(*\ 5\ 4) &\to_\beta (+\ (*\ 5\ 4)(*\ 5\ 4)) \qquad (\blacktriangledown)\\ &\to_\beta (+\ 20\ (*\ 5\ 4))\\ &\to_\beta (+\ 20\ 20)\\ &\to_\beta 40.\end{aligned}$$

Widzimy, że redeksy są niepotrzebnie zwielokratniane, podczas gdy przy podejściu aplikatywnym zostałyby wpierw zredukowane. Obydwie strategie mają więc poważne wady; niekiedy stosuje się je naprzemiennie, ich efektywność zależy od wyrażenia[2]. Używając grafowej reprezentacji λ-termów możemy wprowadzić jednak

[2]Analiza wyrażeń m.in. pod kątem możliwości redukowania wyrażeń strategią aplikatywną to tzw. *strictness analysis*.



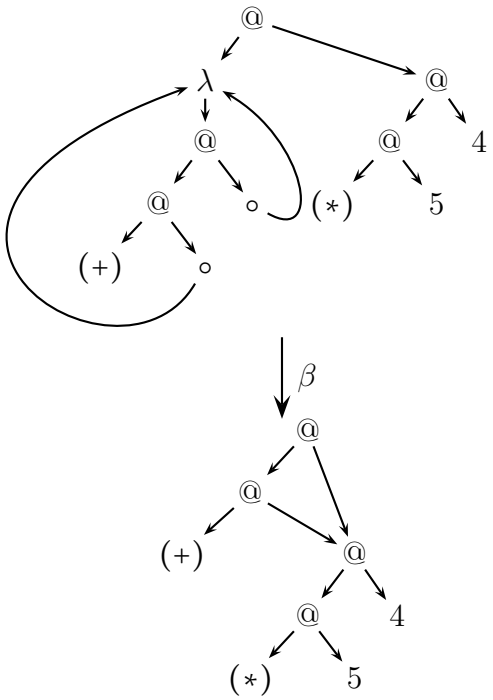
Rysunek 3: Schematyczne porównanie redukcji (▼) z użyciem redukcji grafów wyrażeń bez współdzielenia (po lewej) i ze współdzieleniem (po prawej).

pewną istotną optymalizację: w przypadku, gdy przy pomocy strategii normalnej redukujemy te same podwyrażenia, zamiast powielać wierzchołki odpowiadające β-reduktom, możemy dodać krawędzie prowadzące do tych podwyrażeń (Rysunek 3). Jest to nic innego jak ustawianie wskaźnika do innego, być może nieobliczonego jeszcze podwyrażenia[3] i współdzielenie wyników redukcji (Rysunek 4). Takie rozwiązanie pozwala na przeprowadzenie redukcji w skutek której współdzielone podwyrażenia są redukowane *dokładnie jeden raz*. Zauważmy jednak, że używając strategii zachłannych taka optymalizacja jest niemożliwa, gdyż podwyrażenia odpowiadające argumentom są już wstępnie redukowane do postaci normalnej. Współdzielenie wprowadza również właściwe sobie problemy, których omówienie można znaleźć w [Rin93, Rozdział 3.8.3].

Normalne strategie redukcji, które używają współdzielenia rezultatu redukcji (ang. *sharing*) nazywa się strategiami *call-by-need* albo strategiami *leniwymi*. Wykonywanie redukcji strategią call-by-need aż do uzyskania wyrażenia w słabej czołowiej postaci normalnej nazywamy *leniwą ewaluacją*. Ze względu na kolejność

[3]Na określenie podwyrażeń, których redukcja jest odłożona na później, używa się nazwy *thunk*.

wykonywania redukcji strategie leniwe są więc strategiami nieścisłymi[4]. Implementacja tej strategii odpowiada *leniwej redukcji grafu* [Pey87, Rozdział 12.1, str. 212] na przykład przy pomocy abstrakcyjnej maszyny STG, jak ma to miejsce współcześnie w przypadku kompilatora GHC języka Haskell. Szczegóły technicznie związane z formalizmem maszyny STG można znaleźć w [JLP92]; prezentacja nawet zrębów tej implementacji znacznie wykraczaja poza zamierzony zakres tej pracy.



Rysunek 4: Redukcja (▼) z użyciem redukcji grafu ze współdzieleniem.

Redukcja do słabej czołowej postaci normalnej ma szczególne znaczenie, bowiem pozwala unikać α-konwersji przy kolejnych β-redukcjach. Sensowne wyrażenia, które kodujemy w rachunku λ, są na ogół kombinatorami, tzn. nie zawierają zmiennych wolnych. Zauważmy, że jeśli redukujemy λ-term zamknięty strategią normalną, to nie jest możliwe aby wykonując podstawienie jakieś zmienne zostały dodatkowo związane λ-abstraktorem. Dopóki redukcja nie zostanie przeprowadzo-

[4]Haskell, jako jeden z nielicznych języków programowania, określany jest jako *nieścisły*; ma to jednak związek z *nieścisłą semantyką* języka, a nie strategią redukcji jego wyrażeń (które notabene najczęściej redukowane są strategią leniwą, a więc nieścisłą).

na wewnątrz λ-abstrakcji, to α-konwersja okazuje się zbędna.

Przykład 10. Rozważmy następujący λ-term:

$$(\lambda y.\,(\lambda x\,y.\,xy)\,y)\,(\lambda x\,y.\,xy)$$

Zauważmy, że wykonanie redukcji strategią aplikatywną (tzn. redukucja redeksu $(\lambda x\,y.\,xy)\,y)$ musi być poprzedzone α-konwersją. Dla porównania przeprowadźmy redukcję strategią normalną:

$$\begin{aligned}(\lambda y.\,(\lambda x\,y.\,xy)\,y)\,(\lambda x\,y.\,xy) &\to_\beta\\ (\lambda x\,y.\,xy)(\lambda x\,y.\,xy) &\to_\beta\\ \lambda y.\,(\lambda x\,y.\,xy)\,y\end{aligned}$$

Żadna zmienna nie została dodatkowo związana. Otrzymany λ-term jest w czołowej postaci normalnej. Przeprowadzenie kolejnej redukcji wewnątrz λ-abstrakcji wymagałoby już jednak uprzedniej α-konwersji.

Własność tę można zachować dla wyrażeń ze zmiennymi wolnymi, jeśli tylko wstępnie przemianujemy zmienne wolne na zmienne nie występujące w wyrażeniu.

1.4 Równościowa teoria rachunku λ

Rachunek λ możemy rozszerzyć o teorię równościową w stylu Hilberta. Składa się ona z następujących reguł i aksjomatów inferencji.

(a) Aksjomaty:

- (α) $\lambda x.\,M = \lambda y.\,M[x/y]$, jeśli $y \in \mathrm{FV}(M)$
- (β) $(\lambda x.\,M)N = M[x/N]$
- (ρ) $M = M$

(b) Reguły inferencji:

$$\frac{M = M'}{NM = NM'}\ \text{(l-con)} \qquad \frac{M = N \qquad N = P}{M = P}\ \text{(trans)}$$

$$\frac{M = M'}{MN = M'N}\ \text{(r-con)} \qquad \frac{M = N}{N = M}\ \text{(sym)}$$

$$\frac{M = M'}{\lambda x.\,M = \lambda x.\,M'}\ (\xi)$$

Powyższy system nazywamy *teorią* $\lambda\beta$; jest ona zbiorem *równości* między λ-termami.

Definicja 20. (Wyprowadzenie) Niech $M, N \in \mathbf{\Lambda}$ będą dowolnymi λ-termami. *Wyprowadzeniem* równości $M = N$ ze zbioru równości Γ w teorii $\lambda\beta$ będziemy nazywali drzewo $\mathcal{P}$ równości takie, że:

(i) liście $\mathcal{P}$ są aksjomatami albo należą do zbioru Γ,

(ii) korzeniem $\mathcal{P}$ jest równość $M = N$ (jest *wnioskiem*),

(iii) wszystkie pozostałe równości w $\mathcal{P}$ są uzyskane ze swoich dzieci (*przesłanek*) za pomocą reguł inferencji.

Jeśli istnieje wyprowadzenie $\mathcal{P}$ równości $M = N$ ze zbioru równości Γ, to $M = N$ nazywamy *wyprowadzalnym w kontekście* Γ i piszemy $\lambda, \Gamma \vdash M = N$. Jeśli Γ jest zbiorem pustym, to wyprowadzenie $\mathcal{P}$ nazywamy *dowodem* równości $M = N$, a o równości $M = N$ mówimy, że jest *dowodliwa*.

Następujący fakt ustala związek między równościami dowodliwymi w $\lambda\beta$ a λ-termami, które są swoimi β-konwersami.

Fakt 2. *([HS08, Lem. 6.4]) Niech* $M, N \in \mathbf{\Lambda}$. *Wówczas*

$$M =_\beta N \iff \lambda \vdash M = N$$

1.5 Semantyka denotacyjna

1.5.1 λ-modele

Ustalmy wpierw, że nie możemy naiwnie interpretować λ-termów jako funkcji i aplikacji argumentów do funkcji. Przypuśćmy bowiem, że w pewnej interpretacji $[\![M]\!] = f_M$, gdzie $f_M \in A \to B$ dla pewnych zbiorów A i B. Wówczas $[\![MM]\!] = f_M(f_M)$, a zatem $f_M \in A$. Oznacza to, że funkcja f_M jest elementem swojej własnej dziedziny i możliwe jest skonstruowanie nieskończonego zstępującego ciąg zbiorów:

$$f_M \ni (f_M, f_M(f_M)) \ni \{f_M\} \ni f_M \ni \dots$$

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

Celem wprowadzenia ustalmy szereg następujących pojęć.

Definicja 21. (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 22. (Ekstensjonalność) Strukturę aplikatywną $\mathcal{D} = (D, \bullet)$ nazywamy *ekstensjonalną*, jeśli dla dowolnych $a, b \in D$ spełnia ona warunek

$$(\forall d \in D\ a \bullet d = b \bullet d) \implies a = b$$

Definicja 23. (Ekstensjonala równoważność) Niech $(D, \bullet)$ będzie strukturą aplikatywną i niech $a, b \in D$. Określamy relację:

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

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

Ustalmy teraz co rozumiemy przez λ-model.

Definicja 24. (Wartościowanie λ-zmiennych) Każde odwzorowanie

$$\rho : V \to D,$$

gdzie V jest zbiorem λ-zmiennych, a D dowolnym niepustym zbiorem, nazywamy *wartościowaniem* (λ-zmiennych).

Jeśli ρ jest wartościowaniem, to symbolem $\rho[x_0/d_0]$ będziemy oznaczali następujące wartościowanie:

$$\rho[x_0/d_0](x) = \begin{cases} d_0, & \text{jeśli } x = x_0, \\ \rho(x) & \text{w przeciwnym wypadku.} \end{cases}$$

Definicja 25. (λ-model) Trójkę $\mathcal{M} = (D, \bullet, [\![\,]\!])$, gdzie

$$[\![\,]\!] : \mathbf{\Lambda} \times D^V \to D$$

oraz $(D, \bullet)$ jest strukturą aplikatywną, nazywamy λ-*modelem*, jeśli dla dowolnego wartościowania ρ spełnione są poniższe własności:

(i) $\forall x \in V\ \ ([\![x]\!]_\rho = \rho(x))$

(ii) $\forall M, N \in \mathbf{\Lambda}\ ([\![MN]\!]_\rho = [\![M]\!]_\rho \bullet [\![N]\!]_\rho)$

(iii) $\forall x \in V\ \forall M \in \mathbf{\Lambda}\ \forall d \in D\ (\,[\![\lambda x.\, M]\!]_\rho \bullet d = [\![M]\!]_{\rho[x/d]})$

(iv) Dla dowolnych wartościowań ρ, σ i dowolnego termu $M \in \mathbf{\Lambda}$, jeśli

$$\forall x \in \mathrm{FV}(M)\ (\rho(x) = \sigma(x)),$$

to $[\![M]\!]_\rho = [\![M]\!]_\sigma$

(v) $\forall M \in \mathbf{\Lambda}\, \forall x, y \in V\ (y \notin \mathrm{FV}(M) \implies [\![\lambda x.\, M]\!]_\rho = [\![\lambda y.\, M[x/y]]\!]_\rho)$

(vi) $\forall M, N \in \mathbf{\Lambda}\ \big((\forall d \in D\, [\![M]\!]_{\rho[x/d]} = [\![N]\!]_{\rho[x/d]}) \implies [\![\lambda x.\, M]\!]_\rho = [\![\lambda x.\, N]\!]_\rho\big)$

Omówmy pokrótce Definicję 25. Zauważmy na początku, że aplikacja w rachunku λ oznacza w istocie działanie binarne na λ-termach; dlatego na uniwersum interpretacji D wprowadzamy strukturę aplikatywną okreslając działanie $\bullet$ w myśl warunku (ii).

Określając działanie $\bullet$ wymagamy w warunku (iii), żeby było ono zgodne z β-redukcją. Zauważmy, że w konsekwencji $(\lambda x.\, M)N =_\beta M[x/N]$ mamy, że

$$\begin{aligned} [\![\lambda x.\, M]\!]_\rho \bullet [\![N]\!]_\rho &= [\![(\lambda x.\, M)N]\!]_\rho \\ &= [\![\lambda x.\, M]\!]_\rho \bullet b \\ &= [\![M]\!]_{\rho[x/b]} \\ &= [\![M[x/N]]\!]_\rho \qquad (\dagger) \end{aligned}$$

Ostatnia równość wymaga dowodu: przeprowadza się go indukcją strukturalną względem M. Szczegółowe uzasadnienia można znaleźć w [HS08, Tw. 15.10(a)]. Interpretacja rezultatu operacji podstawienia termu N za zmienną x nie różni się w tym wypadku niczym od zmiany wartościowania ρ dla zmiennej x na desygnat $[\![N]\!]_\rho$ (symbolicznie: $\rho[x/[\![N]\!]_\rho]$), bowiem w wyniku β-redukcji żadna nowa zmienna nie zostaje związana; przez N zastąpione zostają wszystkie wolne wystąpienia x w M. Zachodzi więc równość

$$[\![M[x/N]]\!]_\rho = [\![M]\!]_{\rho[x/[\![N]\!]_\rho]}.$$

W myśl warunku (iv) wymagamy, żeby interpretacja λ-termu zależała tylko od wartościowania występujących w nim zmiennych wolnych.

Warunek (v) zapewnia, że interpretacja λ-termów, które są swoimi α-konwersami, jest taka sama. Odpowiada on więc aksjomatowi (α) z Podrozdziału 1.4.

Warunek (vi) odpowiada regule (ξ) z Rodrozdziału 1.4. Zauważmy wpierw jednak, że zgodnie z warunkiem (iii)

$$(\forall d \in D\ [\![M]\!]_{\rho[x/d]} = [\![N]\!]_{\rho[x/d]}) \iff [\![\lambda x.\, M]\!]_\rho \sim [\![\lambda x.\, N]\!]_\rho \qquad (\dagger\,\dagger)$$

Warunek (vi) stwierdza więc, że jeśli dwie λ-abstrakcje $(\lambda x.M), (\lambda x.N) \in \mathbf{\Lambda}$ indukują dwie częściowe aplikacje działania $\bullet$

$$([\![\lambda x.M]\!]_\rho \bullet\ ),\ ([\![\lambda x.N]\!]_\rho \bullet\ ) :\ D \to D,$$

które są sobie równe, czyli

$$\forall d \in D\ [\![\lambda x.M]\!]_\rho \bullet d = [\![\lambda x.N]\!]_\rho \bullet d,$$

to interpretacje obu λ-abstrakcji również muszą być sobie równe. Zgodnie z dyskusją we wstępie do niniejszego rozdziału, nie możemy interpretować λ-abstrakcji jako funkcji $D \to D$, nic nie stoi jednak na na przeszkodzie, żeby scharakteryzować je czysto semantycznie przez operację w λ-modelu.

Zgodnie z († †) możemy przepisać warunek (vi) do następującej równoważnej postaci

$$\forall M, N \in \mathbf{\Lambda}\ (\llbracket \lambda x.\, M \rrbracket_\rho \sim \llbracket \lambda x.\, N \rrbracket_\rho \implies \llbracket \lambda x.\, M \rrbracket_\rho = \llbracket \lambda x.\, N \rrbracket_\rho) \qquad \text{(vi')}$$

Własność (vi') w literaturze nazywa się niekiedy warunkiem *słabej ekstensjonalności* λ-abstrakcji.

Definicja 26. (Spełnianie, prawdziwość) Niech $\mathcal{M} = (D, \bullet, \llbracket\ \rrbracket)$ będzie λ-modelem. Jeśli dla dowolnych λ-termów M, N i wartościowania ρ zachodzi

$$\llbracket M \rrbracket_\rho = \llbracket N \rrbracket_\rho,$$

to mówimy, że wartościowanie ρ *spełnia* równość $M = N$ i piszemy

$$\mathcal{M}, \rho \vDash M = N$$

.

Jeśli każde wartościowanie ρ spełnia równość $M = N$, to mówimy, że równość $M = N$ jest *prawdziwa* w $\mathcal{M}$ (lub $\mathcal{M}$ *spełnia* $M = N$) i piszemy $\mathcal{M} \vDash M = N$.

Twierdzenie 3. *Każdy λ-model spełnia wszystkie dowodliwe równania teorii $\lambda\beta$.*

Dowód. Dowód przeprowadzimy indukcją względem wyprowadzenia równania $M = N$. Niech $\lambda \vdash M = N$, $\mathcal{M}$ będzie λ-modelem i ρ dowolnym wartościowaniem. Rozważmy następujące przypadki:

1. Równanie $M = N$ jest aksjomatem w postaci:

   (α) Wówczas na podstawie warunku (v) mamy $\llbracket \lambda x.\, M \rrbracket_\rho = \llbracket \lambda y.\, M[x/y] \rrbracket_\rho$
   (β) Wówczas na podstawie (†) mamy $\llbracket (\lambda x.\, M)N \rrbracket_\rho = \llbracket \lambda y.\, M[x/N] \rrbracket_\rho$
   (ρ) Wynika ze zwrotności semantycznej równości.

   W każdym z powyższych przypadków $\mathcal{M} \vDash M = N$.

2. Równanie $M = N$ jest konkluzją reguły:

(l-con) Wówczas M i N są postaci PM' i PN' odpowiednio, dla pewnego $P \in \mathbf{\Lambda}$. Ponadto, z założenia indukcyjnego $\llbracket M' \rrbracket_\rho = \llbracket N' \rrbracket_\rho$. Wobec tego na podstawie warunku (ii) mamy

$$\llbracket PM' \rrbracket_\rho = \llbracket P \rrbracket_\rho \bullet \llbracket M' \rrbracket_\rho = \llbracket P \rrbracket_\rho \bullet \llbracket N' \rrbracket_\rho = \llbracket PN' \rrbracket_\rho$$

(r-con) Analogicznie do (l-con).

(ξ) Wówczas na podstawie warunku (vi) mamy $[\![\lambda x.\, M]\!]_\rho = [\![\lambda x.\, N]\!]_\rho$.

(trans) Wówczas z założenia indykcyjnego dla pewnego $P \in \mathbf{\Lambda}$ $[\![M]\!]_\rho = [\![P]\!]_\rho$ oraz $[\![P]\!]_\rho = [\![N]\!]_\rho$. Z przechodniości semantycznej równości mamy zatem, że $[\![M]\!]_\rho = [\![N]\!]_\rho$.

(sym) Wynika symetryczności semantycznej równości.

W każdym z powyższych przypadków $\mathcal{M} \vDash M = N$.

□

1.5.2 Model Scotta D_∞

W tym podrozdziale przybliżymy konstrukcję modelu Scotta D_∞ zgodnie z [HS08, Rozdział 16]. Rozwiązanie D. Scotta polega na tym, aby zamiast naiwnych prób interpretacji termów jako funkcji, 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 wówczas poprawnie określona:

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

Wszystkie symbole użyte w powyższym wtym prowadzeniu zostaną zdefiniowane w tym podrozdziale.

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 $\bot$. Niech $X \subset D$. *Ograniczeniem górnym* X nazywamy element $u \in D$ taki, że $x \sqsubseteq u$ dla wszystkich $x \in X$. *Kresem górny* zbioru X nazywamy element $\ell \in D$ taki, że ℓ jest ograniczeniem górnym X i $\ell \sqsubseteq u$ dla wszystkich ograniczeń górnych u zbioru X. Element taki, o ile istnieje, będziemy oznaczali symbolem $\bigsqcup X$. Podzbiór $X \subset D$ nazywamy *skierowanym*, jeśli $X \neq \varnothing$ i dla dowolnych $x, y \in X$ istnieje $z \in X$ taki, że $x \sqsubseteq z$ i $y \sqsubseteq z$.

Definicja 27. (Zupełny porządek częściowy) Porządek częściowy $(D, \sqsubseteq)$ taki, że

(a) posiada element najmniejszy oraz

(b) każdy skierowany poddzbiór $X \subset D$ ma kres górny,

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

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

Przykład 11. Ustalmy $\perp \notin \mathbb{N}$ i niech $\mathbb{N}^+ = \mathbb{N} \cup \perp$. Określmy na $\mathbb{N}^+$ następującą relację:

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

$\sqsubseteq$ jest oczywiście zwrotna, przechodnia i antysymetryczna. Widzimy, że $\mathbb{N}^+$ ma względem niej element najmniejszy (jest nim $\perp$). Podzbiory jednoelementowe $\mathbb{N}^+$ i podzbiory dwuelementowe zawierające $\perp$ to wszystkie skierowane podzbiory $\mathbb{N}^+$. Widzimy, że każdy z nich ma kres górny, a zatem $\mathbb{N}^+$ jest cpo.

Definicja 28. (Monotoniczność, ciągłość, ścisłość) Niech D i D' bedą cpo i $\varphi : D \to D'$.

(a) Powiemy, że φ jest *monotoniczna*, jeśli $\varphi(a) \sqsubseteq' \varphi(b)$ dla $a \sqsubseteq b$.

(b) Powiemy, że φ jest *ciągła (w sensie Scotta*[5]), jeśli $\varphi(\bigsqcup X) = \bigsqcup \varphi(X)$ dla wszystkich skierowanych podzbioru $X \subseteq D$.

(c) Powiemy, że φ jest *ścisła*[6] (ang. *strict*), jeśli $\varphi(\perp) = \perp'$. Funkcje, które nie są ścisłe, nazywamy niekiedy *leniwymi* (ang. *lazy*)

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

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

Twierdzenie 4. *Jeśli D i D' są cpo, to $[D \to D']$ jest cpo.*

Dowód. Określmy na $[D \to D']$ relację $\preceq$:

$$\varphi \preceq \psi \quad \Leftrightarrow \quad \forall d \in D \, (\varphi(d) \sqsubseteq' \psi(d)) \,,$$

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

[5]Funkcje te są ciągłe w topologicznym sensie względem topologii Scotta; taką topologię można określić na każdym cpo.

[6]Każdą semantykę, w której $\varphi(\perp) \neq \perp'$ dla pewnej funkcji φ, nazywamy *nieścisłą* (ang. *non-strict*).

Niech teraz Φ będzie skierowanym podzbiorem $[D \to D']$. Wówczas dla wszystkich $d \in D$ zbiór $\{\varphi(d) \mid \varphi \in \Phi\}$ jest skierowanym podzbiorem D'. 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 Φ jest skierowany, to istnieje φ_3 taki, że $\varphi_1 \preceq \varphi_3$ i $\varphi_2 \preceq \varphi_3$. Wówczas dla dowolnego $d \in D$ $\varphi_1(d) \sqsubseteq' \varphi_3(d)$ oraz $\varphi_1(d) \sqsubseteq' \varphi_3(d)$, a zatem zbiór $\{\varphi(d) \mid \varphi \in \Phi\}$ istotnie jest skierowany dla każdego $d \in D$. Ponieważ zaś D' jest cpo, to dla każdego $d \in D$ istnieje kres górny $\{\varphi(d) \mid \varphi \in \Phi\}$. Określmy więc funkcję $\Psi_\Phi : D \to D'$ następującym wzorem:

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

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

1. Pokażemy najpierw, że Ψ_Φ jest ciągła. Niech X będzie dowolnym skierowanym podzbiorem D. Wówczas:

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

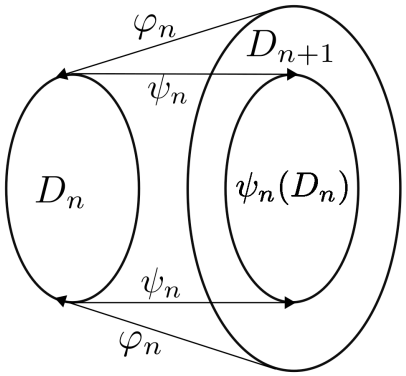
2. Niech $\varphi \in \Phi$. Wówczas dla dowolnego $d \in D$ mamy $\varphi(d) \sqsubseteq' \bigsqcup \Phi d = \Psi_\Phi(d)$. Zatem $\varphi \preceq \Psi_\Phi$ i Ψ_Φ jest ograniczeniem górnym Φ. Jeśli Ψ_Φ nie jest kresem górnym Φ, to istnieje χ takie, że $\chi \preceq \Psi_\Phi$ oraz χ jest kresem górnym Φ. Wówczas dla dowolnego $d \in D$ mamy, że $\chi(d)$ jest kresem górnym Φd oraz $\chi(d) \sqsubseteq' \Psi_\Phi(d)$. Ale $\Psi_\Phi(d) = \bigsqcup \Phi d$, a zatem $\chi(d)$ nie jest kresem górnym Φd. Stąd Ψ_Φ jest kresem górnym Φ.

Definicja 29. Dla danego cpo D_0 możemy skonstruować ciąg cpo $\{D_n\}_{n=0}^{\infty}$ określając $D_{n+1} = [D_n \to D_n]$ dla $n \geq 0$. Tak określony ciąg dla $D_0 = \mathbb{N}^+$ (patrz Przykład 11) nazywamy modelem Scotta.

Definicja 30. (Projekcja) Niech D i D' będą cpo. *Projekcją* z D' do D nazywamy parę (φ, ψ) funkcji $\varphi \in [D \to D']$, $\psi \in [D' \to D]$ takich, że

$$\psi \circ \varphi = I_D \quad \text{oraz} \quad \varphi \circ \psi \preceq I_{D'} \tag{2}$$

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



Rysunek 5: Projekcja (φ_n, ψ_n) z D_{n+1} do D_n

.

Okazuje się, że ciąg cpo $\{D_n\}_{n=0}^{\infty}$ skonstruowany w myśl Wniosku 29 można określić definiując dla każdego $n \in \mathbb{N}$ projekcję (φ_n, ψ_n) z D_{n+1} do D_n. Istotnie, spróbujmy skonstruować taką rodzinę projekcji. Wybierzmy $d \in D_0$ i niech κ_d oznacza funkcję stałą

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

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

$$\begin{aligned} \varphi_0(d) &= \kappa_d \quad \text{dla wszystkich } d \in D_0, \\ \psi_0(c) &= c(\bot_0) \quad \text{dla wszystkich } c \in D_1, \end{aligned}$$

gdzie $\bot_0$ jest elementem najmniejszym D_0. Widzimy, że $\varphi_0 : D_0 \to D_1$ i $\psi_0 : D_1 \to D_0$. Funkcje φ_0 i ψ_0 są ciągłe, zaś $\psi_0 \circ \varphi_0 = I_{D_0}$ (dowód pomijamy), zatem (φ_0, ψ_0) jest projekcją z D_1 do D_0.

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

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

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

Ponieważ projekcje (φ_n, ψ_n) przenoszą nas tylko pomiędzy następującymi po sobie cpo w ciągu $\{D_n\}_{n=0}^{\infty}$, (Rysunek 5) w Definicji 31 określamy złożenie pozwalające na projekcje między dowolnymi dwoma wyrazami ciągu.

Definicja 31. Dla $m, n \geq 0$ określamy $\varphi_{mn} : D_m \to D_n$ w następujący sposób:

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

W myśl Definicji 31, dla każdego $n \in \mathbb{N}$ para (φ_n, ψ_n) jest projekcją z D_{n+1} do D_n.

Konstrukcja D_∞ Mając zadane dwa cpo D, D' możemy zapytać czy istnieje włożenie jednego z nich w drugi. Zauważmy, że jeśli φ, ψ jest projekcją z D' do D, to ϕ 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 32. Niech D_∞ 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_∞ określamy relację $\sqsubseteq$ w nastepujący sposób:

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

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_∞ okazuje się być cpo [HS08, Tw. 16.36].

Dodatkowo, określamy projekcję z D_∞ do D_n dla każdego $n \in \mathbb{N}$. Wymaga to dowodu (patrz Fakt 3(i)), który pomijamy.

Definicja 33 (D_∞)**.** Dla $n \geq 0$ określamy funkcje $\varphi_{n\infty} : D_n \to D_\infty$ oraz $\varphi_{\infty n} : D_\infty \to D_n$, gdzie

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

Określona para funkcji spełnia poniższy szereg własności.

Fakt 3 ([HS08, Tw. 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_∞ *do* D_n*,*

(ii) $\varphi_{mn}(a_m) \sqsubseteq a_n$*,*

(iii) $\varphi_{m\infty}(a_m) \sqsubseteq \varphi_{n\infty}(a_n)$*,*

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

(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_∞. Oznacza to, że każdy $a \in D_n$ możemy utożsamiać z odpowiadającym mu $\varphi_{n\infty}(a) \in D_\infty$. Biorąc $a \in D_\infty$ i stosując tę odpowiedniość jako konwencję notacyjną, na podstawie 3(iii) mamy, że

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

zaś na podstawie 3(iv):

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

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

Kolejną istotną obserwacją jest fakt, że dziedzina każdej funkcji z D_n zawiera się w powyższym sensie również w D_n. Fakty 3(iv) i 3(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 $a_{n+1} \in D_{n+1} = [D_n \to D_n]$, to $a_{n+1}(b_n)$ jest określony. A zatem $a_{n+1}(b_n)$ jest pewnym przybliżeniem aplikacji a do b. Widzimy, że rozwiązuje to wymieniony wcześniej problem samoaplikacji.

Definicja 34. Dla $a, b \in D_\infty$ określamy:

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

Zauważmy, że na podstawie Faktu 3(v) D_∞ jest łańcuchem, a zatem zbiorem skierowanym. Ponieważ D_∞ 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 4. *([HS08, p. 16.55])* (D_∞ *jest* λ*-modelem.*

1.6 Kodowanie typów danych

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

Najpopularniejszym sposobem reprezentacji danych przez funkcje w rachunku λ oparty jest na kodowaniu liczb Peano za pomocą tzw. liczebników Churcha. Metoda ta, ze względu na wynikające zeń problemy natury złożonościowej [KPJ14], ma obecnie wyłącznie walory edukacyjne, dlatego w dalszej części pracy pokażemy tzw. kodowanie Scotta. Jest ona interesująca ze względu na praktyczną możliwość reprezentacji algebraicznych typów danych (ADT, ang. *Algebraic Data Types*[7]) znanych ze współczesnych języków funkcyjnych [Jan13], pozwalając tym samym zaimplementować te konstrukcje w dowolnym języku, w którym dostępne są wyrażenia λ. Fakt, że każdy typ danych można zastąpić tym sposobem odpowiadającą mu funkcją, wskazuje na metodę konstruowania prostych języków funkcyjnych [JKP06] oraz na uniwersalność rachunku λ jako języka przejściowego dla kompilatorów języków funkcyjnych [PL92, Rozdział 3]. Druga z tych idei znajduje dziś bardzo praktyczne zastosowanie w przypadku Systemu F – rozszerzenia rachunku λ, który będzie tematem Rozdziału **??** – i kompilatora GHC języka Haskell.

Szerokie omówienie struktur danych, które można wyrazić za pomocą rachunku λ bez typów można znaleźć w [Sel08, Rozdział 3]. Niniejszy rozdział opieramy na [Jan13]. Kodowanie Scotta, które jest jego tematem, jest stosunkowo mało popularne i nie spotyka się go w klasycznej literaturze przedmiotu.

1.6.1 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
```

[7] Nie należy mylić z *Abstract Data Types*.

```
                  | Just a
data Nat          = Zero
                  | Succ Nat
data List t       = Nil
                  | Cons t (List t)
```

Definicja typu rozpoczynają się od słowa kluczowego `data`[8] 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 λ bez typów. Przedstawione tutaj koncepcje w zaskakujący sposób przenoszą się do bardziej złożonych typowanych systemów rachunku λ.

1.6.2 Proste typy wyliczeniowe

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

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

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

$$\text{True} \equiv \lambda ab.\, a$$
$$\text{False} \equiv \lambda ab.\, b$$

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

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

1.6.3 Pary w rachunku λ

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

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

Tego rodzaju typy możemy reprezentować przez domknięcie. Standardowym sposobem reprezentacji pary w rachunku λ jest:

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

Uzywając wyrażeń let, powyższą reprezentację możemy przepisać w postaci:

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

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

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

Przykład 12. Wprowadzone konstrukcje pozwalają nam na definicję skończonych (w sensie liczby konstruktorów) typów. Rozważmy następujące przykłady:

a) Konstruktory wartości typu `Maybe` możemy reprezentować przez

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

Rozważmy następującą funkcję:

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

Odpowiadająca jej reprezentacja to

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

b) Rozważmy następującą funkcję

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

Ustalając reprezentację konstruktorów `Fahrenheit` i `Celsius`:

$$\text{Fahrenheit} \equiv \lambda t f c.\, f t$$
$$\text{Celsius} \equiv \lambda t f c.\, c t$$

otrzymujemy reprezentację funkcji `formTemperature` postaci:

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

1.6.4 Kodowanie rekurencji

Rozważmy następującą funkcję dodawania liczb Peano w języku Haskell:

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

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

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

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

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

Wówczas określamy

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

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

1.6.5 Kodowanie Scotta typów rekursywnych

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

$$\begin{aligned} \text{Zero} &\equiv \lambda zs.\, z & \qquad \text{Nil} &\equiv \lambda nc.\, n \\ \text{Succ} &\equiv \lambda nzs.\, sn & \qquad \text{Cons} &\equiv \lambda x\, x_s\, n\, c.\ c\, x\, x_s \end{aligned}$$

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

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

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

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

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

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

`foldl` może być przy pomocy kodowania Scotta zapisane jako

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

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

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

1.6.6 Kodowanie Churcha typów rekursywnych

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

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

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

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

Liczba naturalna n jest kodowana przez funkcję w której jej pierwszy argument jest aplikowany n razy do drugiego argumentu. Porównując je do kodowania Scotta widzimy, że różnica polega na aplikowaniu do kontynuacji termu $(n\ f\ x)$ w przypadku brania następnika. Da się pokazać [Hin05], że liczebniki Churcha są w istocie operacją `foldl` na argumentach `Succ` i Zero. Istotnie, niech nat $\equiv \lambda c.\ c$ Succ Zero. Wówczas nat $\bar{n} =_\beta \bar{n}$. Z tego powodu kodowanie operacji na liczebnikach Churcha, lub ogólnie – funkcji opartych na rekursji prostej po zbiorze liczb naturalnych – jest wyjątkowo proste przy użyciu tej metody. Przykładowo, używając metody Churcha, operację dodawania kodujemy w następujący sposób:

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

Dla porównania, używając kodowania Scotta:

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

1.6.7 Ogólny schemat kodowania Scotta typów ADT

W ogólnym przypadku, mając nastepującą definicję ADT:

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

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

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

Wówczas następującą definicję częściową funkcji `f`:

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

kodujemy przy za pomocą następujego λ-termu:

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

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

1.7 Podsumowanie

Istotą rachunku λ bez typów jest uchwycenie pojęcia aplikacji argumentu do funkcji. Kodując selektor `if` dla typu `Boolean` w 1.6.2 zauważmy, że nic nie powstrzymuje nas przed zaaplikowaniem do wyrażenia `if` dowolnego λ-termu. Analogiczna sytuacja ma miejsce, gdy określamy operacje na reprezentacji liczb naturalnych. Widzimy, że w ramach tak zakrojonego systemu nie mamy możlwiości uchwycenia które rezultaty są sensowne. Jak przekonamy się w Rozdziale **??**, problem ten eliminuje w pewnym stopniu rozszerzenie systemu rachunku λ o typy wyrażeń. 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 wyrażeń możliwych do zbudowania w rachunku λ nie jest poprawnych. W dziedzinie projektowania języków programowania pożądane zaś są bogate systemy typów, które jednocześnie nie ograniczają ekspresji (lub mówiąc bardziej obrazowo: pozwalałyby na określenie większej ilości poprawnie zbudowanych programówm, ograniczając ilość tych błędnych).

Literatura

[Alt02] Thorsten Altenkirch. "α-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.

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

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

[Hin05] Ralf Hinze. "THEORETICAL PEARL Church Numerals, Twice!" In: *J. Funct. Program.* 15.1 (Jan. 2005), pp. 1–13. ISSN: 0956-7968. DOI: 10.1017/S0956796804005313. URL: http://dx.doi.org/10.1017/S0956796804005313.

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

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

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

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

[JLP92] Peyton Jones, Simon L, and Simon Peyton Jones. "Implementing Lazy Functional Languages on Stock Hardware: The Spineless Tagless G-machine". In: *Journal of Functional Programming* 2 (July 1992), pp. 127–202. URL: https://www.microsoft.com/en-us/research/publication/implementing-lazy-functional-languages-on-stock-hardware-the-spineless-tagless-g-machine/.

[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.

[Pey87] Simon L. Peyton Jones. *The Implementation of Functional Programming Languages (Prentice-Hall International Series in Computer Science)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1987. ISBN: 013453333X.

[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.

[Rin93] Marko van Eekelen Rinus Plasmeijer. *Functional Programming and Parallel Graph Rewriting*. Addison-Wesley, 1993. ISBN: 0471935670. URL: https://clean.cs.ru.nl/Functional_Programming_and_Parallel_Graph_Rewriting.

[Sel08] Peter Selinger. "Lecture notes on the lambda calculus". In: *CoRR* abs/0804.3434 (2008). arXiv: 0804.3434. URL: http://arxiv.org/abs/0804.3434.

[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.