Instytut Informatyki

Programowanie funkcyjne

Wykład 10. Podstawy rachunku lambda

Zdzisław Spławski

- Formalizacje pojęcia obliczalności
- Pierwsze ważniejsze twierdzenia, dotyczące obliczalności
- Teza Churcha-Turinga
- Historia i niektóre zastosowania rachunku lambda
- Gramatyka i konwencje notacyjne
- Zmienne wolne i związane
- Reguły wnioskowania
- Konteksty
- Semantyka operacyjna: redukcje i postacie normalne
- Strategie i grafy redukcji
- Twierdzenie Churcha-Rossera
- Niesprzeczność i zupełność
- Twierdzenie o standardyzacji
- Programowanie funkcyjne
- Literatura
- Zadania kontrolne

Matematyczne modele algorytmu

- Rachunek kombinatorów — M.Schönfinkel (1924), H.Curry (1930)
- Rachunek lambda (λ-rachunek) — A.Church (1932/33)
- Rachunek równań rekurencyjnych — K.Gödel wykorzystując ideę J.Herbranda (1934)
- Funkcje (częściowo) rekurencyjne — S.C.Kleene (1936)
- Maszyny Turinga — A.M.Turing (1936)
- Systemy (produkcje) Posta — E.L.Post (1943)
- Normalne algorytmy Markova — A.A.Markov (1951)
- Maszyny z nieograniczonym rejestrami — J.C.Shepherdson, H.E.Sturgis (1963)
- Język `WHILE` (z danymi w stylu języka `LISP`) — N.D.Jones (1997)
- i inne.

Pierwsze ważniejsze twierdzenia, dotyczące obliczalności

- T.Skolem (1923) zdefiniował funkcje pierwotnie rekurencyjne. Ackermann (1928) pokazał, że istnieje funkcja intuicyjnie obliczalna, która nie jest pierwotnie rekurencyjna.
- Church i Kleene (1936) udowodnili równoważność funkcji rekurencyjnych i λ-obliczalności.
- Church (1936) wysunął hipotezę, że (nieformalne) pojęcie obliczalności można utożsamić z (formalnym) pojęciem λ-definiowalności.
- Turing (1937) udowodnił równoważność obliczalności na maszynach Turinga i λ-obliczalności.
- Turing (1937) wysunął hipotezę że (nieformalne) pojęcie obliczalności można utożsamić z (formalnym) pojęciem obliczalności na maszynach Turinga.

Później udowodniono równoważność wszystkich zaproponowanych do tej pory matematycznych modeli algorytmu.

Teza Churcha [Churcha-Turinga].
Każda funkcja obliczalna w nieformalnym sensie jest λ-definiowalna (rekurencyjna).

- Rachunek lambda (λ-rachunek) jest teorią funkcji rozumianych konstruktywnie jako reguły obliczania, tj. przekształcania argumentu w wynik.
- λ-rachunek został zaproponowany w latach trzydziestych ubiegłego wieku przez Alonzo Churcha jako część systemu formalnego, stanowiącego alternatywną formalizację podstaw matematyki. Chociaż cały system okazał się sprzeczny, nie dotyczy to λ-rachunku.
- Wcześniej, w latach dwudziestych, Moses Schönfinkel zaproponował inną teorię funkcji, opartą na kombinatorach, gdzie nie występuja zmienne związane. W 1927r. Haskell Curry niezależnie wprowadził kombinatory, rozszerzył teorię Schönfinkela oraz pokazał, że jest ona równoważna rachunkowi lambda. Mniej więcej w tym czasie udowodniono równoważność rachunku lambda, funkcji rekurencyjnych i maszyn Turinga.

- Pod koniec lat pięćdziesiątych John McCarthy, zainspirowany rachunkiem lambda, opracował język programowania LISP.
- We wczesnych latach sześćdziesiątych Peter Landin pokazał, jak można zinterpretować Algol-60 w rachunku lambda, dla którego zdefiniował semantykę operacyjną. Zaprojektował też prototypowy język ISWIM (If you See What I Mean) oraz maszynę abstrakcyjną (wirtualną) SECD (Stack-Environment-Control-Dump). ISWIM wywarł duży wpływ na projektantów zarówno języków funkcyjnych, jak i imperatywnych. Wśród współczesnych języków programowania najbliższy językowi ISWIM jest Scheme. Współczesną wersją maszyny SECD jest CAM (Categorial Abstract Machine) dla języka OCaml.

- Wykorzystując te rezultaty Christopher Strachey położył podstawy semantyki denotacyjnej języków programowania. Techniczne problemy rozwiązał w roku 1969 amerykański logik Dana Scott, opracowując teorię dziedzin, która stanowi ważny rozdział informatyki teoretycznej.
- Curry i niezależnie Howard zauważyli odpowiedniość między rachunkiem lambda z typami a dowodami matematycznymi (izomorfizm Curry'ego-Howarda).
- Pod koniec lat siedemdziesiątych David Turner pokazał, że kombinatory również mogą być używane jako efektywne kody maszynowe dla programów funkcyjnych.
- W latach osiemdziesiątych bardzo wiele uwagi poświęcono typom w językach funkcyjnych, co wywarło znaczny wpływ na inżynierię oprogramowania.

- W ten sposób wywodzące się z logiki matematycznej i stworzone jeszcze przed skonstruowaniem pierwszych komputerów rachunek lambda i teoria kombinatorów wywierają coraz większy wpływ na ważne dziedziny informatyki, m.in. podstawy informatyki, projektowanie i semantykę języków programowania, inżynierię oprogramowania (specyfikacje, poprawność ...).

Gramatyka i konwencje notacyjne

Definicja. *Zbiór λ-termów* Λ definiuje się przy użyciu nieskończonego, przeliczalnego zbioru zmiennych $\mathcal{V} = \{v, v', v'', \ldots\}$ i dwóch podstawowych operacji — aplikacji i abstrakcji funkcyjnej.

$$\mathcal{V} ::= v \mid \mathcal{V}'$$
$$\Lambda ::= \mathcal{V} \mid (\Lambda\Lambda) \mid (\lambda\mathcal{V}.\Lambda)$$

Dla uproszczenia zapisu stosuje się następujące konwencje notacyjne.

- Małe litery (np. x, y, x_1) oznaczają zmienne.
- Wielkie litery (np. M, N, P) oznaczają λ-termy.
- $\lambda x_1 \ldots x_n.M$ oznacza $(\lambda x_1(\lambda x_2(\ldots(\lambda x_n(M))\ldots)))$.
  (Abstrakcja wiąże w prawo.)
- $M_1 \ldots M_n$ oznacza $(\ldots(M_1 M_2)\ldots M_n)$.
  (Aplikacja wiąże w lewo.)

Zmienne wolne i związane

Definicja. *Zbiór zmiennych wolnych termu* M, oznaczany przez $FV(M)$, i *zbiór zmiennych związanych*, oznaczany przez $BV(M)$, definiuje się przez indukcję po strukturze termu:

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

$$\begin{aligned} BV(x) &= \emptyset \\ BV(MN) &= BV(M) \cup BV(N) \\ BV(\lambda x.M) &= BV(M) \cup \{x\} \end{aligned}$$

Przykład. $(\lambda x.\underline{y}\ \overline{x})\ (\lambda y.\underline{x}\ \overline{y})$.
$\underline{z}$: wolne wystąpienie zmiennej z.
$\overline{z}$: związane wystąpienie zmiennej z.

Kombinatory

Definicja.

(i) M jest *termem stałym*, *termem zamkniętym* lub *kombinatorem* (ang. ground term, closed term, combinator) jeśli $FV(M) = \emptyset$.

(ii) $\Lambda^0 = \{M \in \Lambda \mid FV(M) = \emptyset\}$.

Najbardziej znane i najczęściej używane są kombinatory:
$\mathsf{I} = \lambda x.x, \mathsf{K} = \lambda xy.x, \mathsf{S} = \lambda xyz.xz(yz)$.
Są one charakteryzowane za pomocą następujących aksjomatów:

$$\mathsf{I}\,x = x \tag{ax$\mathsf{I}$}$$
$$\mathsf{K}\,x\,y = x \tag{ax$\mathsf{K}$}$$
$$\mathsf{S}\,x\,y\,z = x\,z\,(y\,z) \tag{ax$\mathsf{S}$}$$

Kombinator I można zdefiniować następująco: $\mathsf{I}{=}\mathsf{SKK}$
i wyprowadzić jego aksjomat:

$$\mathsf{I}\,x = \mathsf{SKK}\,x = \mathsf{K}\,x\,(\mathsf{K}\,x) = x.$$

Podstawienie za zmienną wolną

$M \equiv N$ oznacza tekstową identyczność termów M i N z dokładnością do zamiany nazw zmiennych związanych.

Definicja. Wynik podstawiania N za wolne wystąpienia zmiennej x w termie M, oznaczany przez $M[x := N]$ lub $M[N/x]$, można zdefiniować indukcyjnie następująco:

$$y[x := N] \equiv \begin{cases} N, & \text{gdy } y \equiv x \\ y, & \text{gdy } y \not\equiv x \end{cases}$$

$$(P\,Q)[x := N] \equiv (P[x := N])(Q[x := N])$$

$$(\lambda y.P)[x := N] \equiv \begin{cases} \lambda x.P, & \text{jeśli } y \equiv x \\ \lambda y.(P[x := N]), & \text{jeśli } y \not\equiv x \text{ i } y \notin FV(N) \\ & \text{lub } x \notin FV(P) \\ \lambda z.(P[y := z][x := N]), & \text{jeśli } y \not\equiv x \text{ i } y \in FV(N) \\ & \text{i } x \in FV(P), \text{ gdzie } z \text{ jest dowolną} \\ & \text{„świeżą'' zmienną, tzn. } z \notin FV(N) \cup \\ & BV(N) \cup FV(P) \cup BV(P) \end{cases}$$

Przykład.

$$(\lambda y.x(\lambda x.xy))[x := yz] \equiv \lambda w.yz(\lambda x.xw)$$

Reguły wnioskowania dla rachunku lambda I

$$\frac{}{(\lambda x.M)\ N\ =\ M[x := N]}\ \beta$$

$$\frac{}{\lambda x.Mx\ =\ M}\ \eta \quad \text{gdy } x \notin FV(M)$$

$$\frac{}{M\ =\ M}\ \text{Refl} \qquad\qquad \frac{N\ =\ M}{M\ =\ N}\ \text{Sym}$$

$$\frac{K\ =\ M \qquad M\ =\ N}{K\ =\ N}\ \text{Trans}$$

$$\frac{K\ =\ L \qquad M\ =\ N}{K\ M\ =\ L\ N}\ \text{MonApp} \qquad\qquad \frac{M\ =\ N}{\lambda x.M\ =\ \lambda x.N}\ \text{MonAbs}$$

Reguły wnioskowania dla rachunku lambda II

Powyższy rachunek nosi nazwę rachunku $\lambda\eta$ (lub $\lambda\beta\eta$). Jeśli pominiemy regułę (η), to otrzymamy teorię λ (lub $\lambda\beta$). Jeśli w systemie λ można wyprowadzić równość $M = N$, to piszemy $\lambda \vdash M = N$.

Reguły wnioskowania dla rachunku lambda III

Przykład. $\lambda \vdash (\lambda xy.x)\ (\lambda z.z) = (\lambda x.x)\ (\lambda yz.z)$.

$$\dfrac{\dfrac{}{(\lambda xy.x)\ (\lambda z.z) = \lambda yz.z}\ \beta \qquad \dfrac{\dfrac{}{(\lambda x.x)\ (\lambda yz.z) = \lambda yz.z}\ \beta}{\lambda yz.z = (\lambda x.x)\ (\lambda yz.z)}\ \text{Sym}}{(\lambda xy.x)\ (\lambda z.z) = (\lambda x.x)\ (\lambda yz.z)}\ \text{Trans}$$

W celu sformalizowania zamiany zmiennych związanych Church wprowadził poniższą regułę (α) .

$$\dfrac{}{\lambda x.M \ = \ \lambda y.M[x := y]}\ \alpha \qquad \text{gdy } y \notin FV(M) \cup BV(M)$$

Reguły wnioskowania dla rachunku lambda IV

Semantyka termów, różniących się tylko zmiennymi związanymi jest identyczna, więc zwykle reguła (α) przenoszona jest do metajęzyka, a termy w rachunku lambda rozważane są z dokładnością do α-kongruencji. My również przyjmiemy taką konwencję.

Relacja równoważności "$\sim$" jest kongruencją ze względu na f, jeśli z $x \sim y$ wynika $f(x) \sim f(y)$, czyli kongruencja jest monotoniczną relacją równoważności.

Definicja (Konteksty).

(i) *Kontekst* $C[\,]$ jest termem z "dziurami". Formalnie:
x (dowolna zmienna) jest kontekstem,
$[\,]$ jest kontekstem,
jeśli $C_1[\,]$ i $C_2[\,]$ są kontekstami, to $C_1[\,]C_2[\,]$ i $\lambda x.C_1[\,]$ są też kontekstami.

(ii) Jeśli $C[\,]$ jest kontekstem i $M \in \Lambda$, to $C[M]$ oznacza wynik włożenia (nie podstawienia — nie ma tu zmiany zmiennych związanych) M w dziury kontekstu $C[\,]$. Przy tym zmienne wolne termu M mogą zostać związane w $C[M]$, tzn. konteksty *nie* są rozpatrywane modulo α-kongruencja. Np. jeśli $C[\,] \equiv \lambda x.x(\lambda y.[\,])$ i $M \equiv xy$, to $C[M] \equiv \lambda x.x(\lambda y.xy)$.

Lemat 1 (Przezroczystość referencyjna). Niech $C[\,]$ będzie kontekstem. Wówczas (z dokładnością do nazw zmiennych związanych)

$$M = N \Rightarrow C[M] = C[N].$$

Dowód. Przez indukcję po strukturze $C[\,]$. ■

β-redukcja

W zbiorze lambda termów Λ definuje się relację beta-redukcji jako najmniejszą relację $\to_\beta$ (β-*redukcja w jednym kroku* lub *kontrakcja*) taką, że:

- (podstawa) $(\lambda x.M)N \to_\beta M[x := N]$
- (kompatybilność) jeśli $M \to_\beta N$, to $ZM \to_\beta ZN$, $MZ \to_\beta NZ$ oraz $(\lambda x.M) \to_\beta (\lambda x.N)$.

Relacja β-*redukcji* $\twoheadrightarrow_\beta$ jest zwrotnym i przechodnim domknięciem relacji $\to_\beta$.
Relacja β-*konwersji* $=_\beta$ jest relacją równoważności (a nawet kongruencji) generowaną przez $\to_\beta$.

Twierdzenie 2. $\lambda \vdash M = N \Leftrightarrow M =_\beta N$.
Dowód.
($\Rightarrow$) Przez indukcję po strukturze drzewa wywodu.
($\Leftarrow$) Przez indukcję po sposobie generowania relacji $=_\beta$.

Postać normalna lambda termu I

Niech $M \in \Lambda$.

- M jest w postaci β-normalnej (β-NF, ang. normal form), jeśli nie zawiera β-redeksu (ang. redex = reducible expression), tj. podtermu $(\lambda x.P)Q$.
- M jest w postaci $\beta\eta$-normalnej ($\beta\eta$-NF), jeśli nie zawiera β- ani η-redeksu, tj. podtermów $(\lambda x.P)Q$ ani $\lambda x.Px$, gdzie $x \notin FV(P)$.
- M jest w czołowej postaci normalnej (HNF, ang. head-normal form), jeśli $M \equiv \lambda x_1 \ldots x_n.yN_1 \ldots N_m$ dla $m, n \geq 0$.
- M jest w słabej czołowej postaci normalnej (WHNF, ang. weak head-normal form), jeśli $M \equiv \lambda x.N$ lub $M \equiv yN_1 \ldots N_m$ dla $m \geq 0$.
- M ma R-NF, jeśli $\exists N.M = N$ i N jest w R-NF, gdzie R oznacza dowolną redukcję.

Postać normalna lambda termu II

Termy w postaci normalnej są wartościami.

Przykład. $\lambda x.((\lambda y.\lambda z.fzy)x)$ nie jest w β-NF, ani w HNF, jest w WHNF.

Lemat 3. Niech M będzie w β-NF. Wówczas

$$M \twoheadrightarrow_\beta N \Rightarrow M \equiv N$$

Dowód. Jeśli M jest w β-NF, to nie zawiera redeksu. Wobec tego nigdy $M \to_\beta N$. Jeśli więc $M \twoheadrightarrow_\beta N$, to tylko dlatego, że $M \equiv N$.

Strategie redukcji I

Zgodnie z powyższymi definicjami lambda term może zawierać kilka redeksów. Na przykład term:

$$\underline{(\lambda x.xyxx)(\underline{(\lambda z.z)w})}$$

zawiera dwa β-redeksy $(\lambda x.xyxx)((\lambda z.z)w)$ oraz $(\lambda z.z)w$. Można przeprowadzać kontrakcje redeksów zgodnie z wybraną strategią.

Strategie redukcji II

- *Redukcja normalna* (ang. normal-order reduction, NOR) polega na kontrakcji lewostronnego zewnętrznego redeksu, tj. redeksu, który zaczyna się najbardziej na lewo i nie jest zawarty w żadnym innym redeksie.
- *Redukcja aplikatywna* (ang. applicative-order reduction, AOR) polega na kontrakcji lewostronnego wewnętrznego redeksu, tj. lewostronnego redeksu, nie zawierającego innych redeksów.
- Są też inne, mniej ważne strategie redukcji.

Term jest *silnie normalizowalny* (ang. strongly normalizing), jeśli każda strategia redukcji doprowadza do postaci normalnej.

Strategie redukcji III

Poniższe slogany ułatwiają zapamiętanie istoty najważniejszych strategii redukcji.

- **Redukcja normalna**: wartościuj każdy argument tyle razy, ile trzeba (być może wcale).
- **Redukcja aplikatywna**: wartościuj każdy argument dokładnie raz.

Strategie redukcji — przykłady I

$$
\begin{aligned}
\underline{(\lambda x.xyxx)((\lambda z.z)w)} &\to (\underline{(\lambda z.z)w})y((\lambda z.z)w)((\lambda z.z)w) \\
&\to wy(\underline{(\lambda z.z)w})((\lambda z.z)w) \\
&\to wyw(\underline{(\lambda z.z)w}) \\
&\to wyww \qquad\qquad \text{NOR}
\end{aligned}
$$

$$
\begin{aligned}
(\lambda x.xyxx)(\underline{(\lambda z.z)w}) &\to \underline{(\lambda x.xyxx)w} \\
&\to wyww \qquad\qquad \text{AOR}
\end{aligned}
$$

Strategie redukcji — przykłady II

Niech $\omega \equiv \lambda x.xx$ oraz $\Omega \equiv \omega\omega$.

$\Omega \equiv \omega\omega \equiv \underline{(\lambda x.xx)(\lambda x.xx)} \to \underline{(\lambda x.xx)(\lambda x.xx)} \to \ldots$

$\underline{(\lambda x.xxy)(\lambda x.xxy)} \to \underline{(\lambda x.xxy)(\lambda x.xxy)}y \to \ldots$

$\underline{(\lambda x.y(\lambda z.z))(\omega\omega)} \to y(\lambda z.z)$ NOR

$(\lambda x.y(\lambda z.z))(\underline{\omega\omega}) \to (\lambda x.y(\lambda z.z))(\underline{\omega\omega}) \to \ldots$ AOR

Grafy redukcji

Definicja. *Graf R-redukcji* termu M (notacja $G_R(M)$) jest zbiorem

$$\{N \in \Lambda \mid M \twoheadrightarrow_R N\}$$

ukierunkowanym relacją redukcji $\to_R$. Jeśli kilka redeksów powoduje przekształcenie $M_0 \twoheadrightarrow_R M_1$, to tyle samo ukierunkowanych krawędzi prowadzi od M_0 do M_1 w $G_R(M)$).

Przykład.

$G_\beta(\Omega)$, dla $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$

$$\begin{tikzcd} \Omega \arrow[loop left] \end{tikzcd}$$

$G_\beta(WWW)$, dla $W \equiv \lambda xy.xyy$

$$\begin{tikzcd}
WWW \arrow[d, shift left] & (\lambda y.yyy)W \arrow[l] \\
(\lambda y.Wyy)W \arrow[u, shift left] \arrow[r, shift left] & (\lambda y.(\lambda z.yzz)y)W \arrow[l, shift left] \arrow[u]
\end{tikzcd}$$

Twierdzenie Churcha-Rossera

Jeśli $M \twoheadrightarrow_\beta P$ i $M \twoheadrightarrow_\beta Q$, to dla pewnego N zachodzi $P \twoheadrightarrow_\beta N$ i $Q \twoheadrightarrow_\beta N$.

$$
\begin{tikzcd}
 & M \arrow[dl, two heads, "\beta"'] \arrow[dr, two heads, "\beta"] & \\
P \arrow[dr, two heads, dotted, "\beta"'] & & Q \arrow[dl, two heads, dotted, "\beta"] \\
 & N &
\end{tikzcd}
$$

Definicja.

(i) *Równanie* jest formułą postaci $M = N$, gdzie $M, N \in \Lambda$; równanie jest *zamknięte*, jeśli $M, N \in \Lambda^0$.

(ii) Niech $\mathcal{T}$ będzie formalną teorią równościową (z równaniami jako formułami). Teoria $\mathcal{T}$ jest *niesprzeczna* (ang. consistent), co zapisujemy $\mathrm{Con}(\mathcal{T})$, jeśli nie każde zamknięte równanie jest jej twierdzeniem. W przeciwnym razie teoria $\mathcal{T}$ jest *sprzeczna* (ang. inconsistent).

(iii) Jeđli $\mathcal{T}$ jest zbiorem równań, to teoria $\lambda + \mathcal{T}$ jest tworzona przez dodanie równań $\mathcal{T}$ jako aksjomatów do λ. $\mathcal{T}$ jest *niesprzeczna*, co zapisujemy $\mathrm{Con}(\mathcal{T})$, jeśli $\mathrm{Con}(\lambda + \mathcal{T})$.

Wnioski z twierdzenia Churcha-Rossera I

Wniosek 1. Jeśli $M =_\beta N$, to istnieje taki term L, że $M \twoheadrightarrow L$ i $N \twoheadrightarrow L$.

Dowód. Przez indukcję po sposobie generowania relacji $=_\beta$.

Wniosek 2.
(i) Jeśli M ma N jako β-NF, to $M \twoheadrightarrow_\beta N$.
(ii) λ-term ma co najwyżej jedną β-NF.
Dowód.
(i) Niech $M =_\beta N$, gdzie N jest β-NF. Na podstawie Wniosku 1 $M \twoheadrightarrow_\beta L$ i $N \twoheadrightarrow_\beta L$ dla pewnego L. Wówczas $N \equiv L$ na podstawie Lematu 3, a więc $M \twoheadrightarrow_\beta N$.
(ii) Niech M ma N_1 i N_2 jako β-NF. Wówczas $N_1 =_\beta M =_\beta N_2$. Na podstawie Wniosku 1 $N_1 \twoheadrightarrow_\beta L$ i $N_2 \twoheadrightarrow_\beta L$ dla pewnego L więc $N_1 \equiv L \equiv N_2$ na podstawie Lematu 3.

Wnioski z twierdzenia Churcha-Rossera II

Dalsze wnioski:
(1) λ-rachunek jest niesprzeczny jako teoria równościowa, tzn. nie można w niej wyprowadzić wszystkich równości, np. $\lambda \nvdash$ **true** $=$ **false**, gdzie **true** $\equiv \lambda xy.x$ i **false** $\equiv \lambda xy.y$. W przeciwnym razie **true** $=_\beta$ **false** na podstawie Tw.2, co jest niemożliwe na podstawie Wniosku 2(ii), ponieważ **true** i **false** są różnymi β-NF.
(2) W celu znalezienia β-NF termu M (jeśli istnieje), różne podtermy termu M mogą być redukowane w dowolnej kolejności. Jeśli redukcja doprowadzi do β-NF, to na podstawie Wniosku 2(ii) β-NF jest jedyna.

Twierdzenie 4 (Zupełność). Załóżmy, że M i N mają postać nornalną. Wówczas albo $\lambda\eta \vdash M = N$ albo $\lambda\eta + M = N$ jest sprzeczna.

Twierdzenie o standardyzacji

Jeśli term M ma postać normalną N to istnieje normalna redukcja z M do N.

Programowanie funkcyjne I

Jak widzieliśmy, beta redukcja wymaga zmiany zmiennych związanych (stosując α-konwersję) w przypadku konfliktu nazw zmiennych, powodującego związanie zmiennej wolnej w wyniku redukcji, np.

$$\lambda x.\underline{(\lambda yx.fxy)x} \to_\beta \lambda x.\lambda z.fzx$$

Taka operacja jest jednak kosztowna i w językach funkcyjnych unika się jej, redukując wyrażenia do słabej czołowej postaci normalnej (WHNF). Powyższy term jest już w WHNF. Wartościowanie zostanie przeprowadzone po zaaplikowaniu do argumentu, co spowoduje zniknięcie lewej zmiennej związanej x. Problematyczne wolne wystąpienie zmiennej x w $(\lambda yx.fxy)x$ zostanie zastapione przez wartość argumentu i konflikt nazw zmiennych znika, np.

$$\underline{(\lambda x.(\lambda yx.fxy)x)s} \to_\beta \underline{(\lambda yx.fxy)s} \to_\beta \lambda x.fxs$$

Programowanie funkcyjne II

W językach funkcyjnych mają zastosowanie dwie strategie wartościowania: wartościowanie gorliwe (ang. eager evaluation) i wartościowanie leniwe (ang. lazy evaluation) , będące sposobami implementacji strategii AOR i NOR.

wartościowanie gorliwe = AOR do WHNF

wartościowanie leniwe = NOR do WHNF + współdzielenie + leniwe konstruktory

Przy wartościowaniu leniwym każdy argument funkcji jest wartościowany co najwyżej raz. Argumenty leniwych konstruktorów nie są wartościowane.

Czasem stosuje się strategie mieszane, np. wartościowanie gorliwe + leniwe konstruktory. W OCamlu konstruktory są gorliwe. W Haskellu konstruktory są leniwe.

Język funkcyjny można potraktować jak rachunek lambda (beztypowy lub z typami) z dodanymi stałymi (z odpowiednimi regułami redukcji) i dużą ilością „lukru syntaktycznego”.

Literatura (wybrane pozycje) I

- H.P.Barendregt, *The Lambda Calculus. Its Syntax and Semantics*, Elsevier, Amsterdam 1984 (revised edition).
- H.Barendregt, W.Dekkers, R.Statman, *Lambda Calculus with Types*, Cambridge University Press, Cambridge 2013
- H.P.Barendregt, *Functional Programming and Lambda Calculus*, w: J.van Leeuwen (ed.), Handbook of Theoretical Computer Science, vol. B, North Holland 1990, Ch.7, pp. 321-363
- H.P.Barendregt, *Lambda Calculi with Types*, w: S.Abramsky, Dov M. Gabbay, T.S.E. Maibaum, Handbook of Logic in Computer Science, vol. 2, Clarendon Press, Oxford 1992, pp. 117-309, http://www.cs.ru.nl/˜henk/papers.html

Literatura (wybrane pozycje) II

- H.Barendregt, E.Barendsen, *Introduction to Lambda Calculus*, revised edition 1998, 2000, http://www.cse.chalmers.se/research/group/logic/TypesSS05/Extra/geuvers.pdf
- F.Cardone, R.Hindley, *History of Lambda-Calculus and Combinatory Logic*, w: Handbook of the History of Logic 2006
- Ch.Hankin, *Lambda Calculi. A Guide for Computer Scientists*, Oxford University Press, Oxford 1994
- J.R.Hindley, J.P.Seldin, *Lambda-calculus and Combinators . An Introduction*, Cambridge University Press, Cambridge 2008

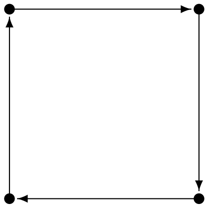
1. W poniższym termie wskaż wszystkie beta-redeksy.
   $(\lambda x.x)((\lambda x.x)(\lambda z.(\lambda x.x)z))$
2. Przeprowadź normalizację poniższego termu. Zwróć uwagę na konieczność zmiany nazwy zmiennej związanej.
   $(\lambda x.xx)(\lambda yz.yz)$
3. Przeprowadź normalizację poniższych termów, jeśli to możliwe. Pokaż wszystkie możliwe ścieżki redukcji (w postaci grafu redukcji).
   $(\lambda x.x)(\lambda z.z)$
   $(\lambda x.y)(\lambda z.z)$
   $(\lambda x.xx)(\lambda z.z)$
   $(\lambda x.(\lambda y.yx)z)(zw)$
   $(\lambda x.x)((\lambda x.x)(\lambda z.(\lambda x.x)z))$
   $(\lambda uv.v)((\lambda x.xx)(\lambda x.xx))$
   $(\lambda x.xx)(\lambda x.xx)$
   $(\lambda x.xxy)(\lambda x.xxy)$

4. Narysuj graf β-redukcji dla termu MM, gdzie

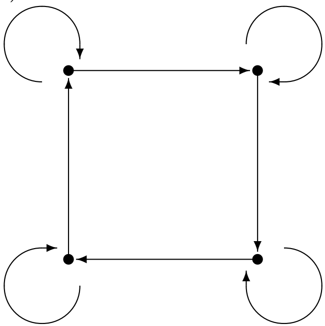
$$M \equiv \lambda x.(\lambda y.yy)x$$

5. Znajdź lambda termy, posiadające poniższe grafy β-redukcji.

a)



b)



6. Udowodnij, że w rachunku lambda regułę (η) można zastąpić poniższą regułą ekstensjonalości:

$$\frac{Mx \;=\; Nx}{M \;=\; N}\,(ext) \quad x \notin FV(M) \cup FV(N)$$

Pokaż, że w $\lambda\beta\eta$ można wywieść regułę (ext) jako regułę wtórną i odwrotnie, w $\lambda\beta + (ext)$ można wywieść (η).