Programowanie funkcyjne

Wykład 12. Funkcje rekurencyjne i rachunek lambda

Zdzisław Spławski

- Formalizacje pojęcia obliczalności
- Pierwsze ważniejsze twierdzenia, dotyczące obliczalności
- Teza Churcha-Turinga
- Funkcje rekurencyjne
- Przykłady funkcji pierwotnie rekurencyjnych
- Funkcja Ackermanna
- Historia i niektóre zastosowania rachunku lambda
- Gramatyka i konwencje notacyjne
- Zmienne wolne i związane
- Reguły wnioskowania
- Semantyka operacyjna
- Strategie i grafy redukcji
- Twierdzenie Churcha-Rossera
- Twierdzenie o standardyzacji
- Programowanie funkcyjne
- Literatura
- Zadania kontrolne

Matematyczne modele algorytmu

- Funkcje pierwotnie rekurencyjne — T.Skolem (1923)
- Rachunek kombinatorów — Schönfinkel (1924), Curry (1930)
- Rachunek lambda (λ-rachunek) — A.Church (1932/33)
- Funkcje (częściowo) rekurencyjne — S.C.Kleene (1936)
- Rachunek równań rekurencyjnych — K.Gödel wykorzystując ideę J.Herbranda (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

- 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).

Operacja minimum

1. *Operacja minimum.* Niech $f : \mathbb{N}^{n+1} \to \mathbb{N}$. $(\mu m.f(x_1, \ldots, x_n, m) = 0) : \mathbb{N}^n \to \mathbb{N}$ oznacza najmniejszą liczbę m, dla której zachodzi $f(x_1, \ldots, x_n, m) = 0$ i $f(x_1, \ldots, x_n, k)$ jest zdefiniowana dla wszystkich $k \leq m$; w przeciwnym razie wynik jest niezdefiniowany.
2. *Operacja minimum ograniczonego.* Niech $f : \mathbb{N}^{n+1} \to \mathbb{N}$. $(\mu m < z.f(x_1, \ldots, x_n, m) = 0) : \mathbb{N}^n \to \mathbb{N}$ oznacza najmniejszą liczbę $m < z$, dla której zachodzi $f(x_1, \ldots, x_n, m) = 0$ i $f(x_1, \ldots, x_n, k)$ jest zdefiniowana dla wszystkich $k \leq m$; w przeciwnym razie wynik jest niezdefiniowany.
3. *Funkcją numeryczną* jest dowolna funkcja $f : \mathbb{N}^n \to \mathbb{N}$ dla pewnego n.
4. $\vec{x} \stackrel{\text{df}}{=} \langle x_1, \ldots, x_n \rangle$ dla $n \in \mathbb{N}$.

Funkcje bazowe

Definicja. *Funkcje bazowe* (*początkowe*):

1. $Z(x) = 0$, dla każdego x — funkcja zerująca
2. $S(x) = x + 1$, dla każdego x — funkcja następnika (ang. succesor)
3. $U_n^i(x_1, \ldots, x_n) = x_i$, dla $1 \leq i \leq n$ — funkcja projekcji

Schematy tworzenia nowych funkcji I

Definicja. Niech A będzie zbiorem funkcji numerycznych.

1. A jest zamknięty ze względu na operację *superpozycji* (*składania funkcji*), jeśli dla wszystkich f zdefiniowanych następująco:

$$f(\vec{x}) = g(h_1(\vec{x}), \ldots, h_r(\vec{x}))$$

gdzie $g, h_1, \ldots, h_r \in A$, zachodzi $f \in A$.

2. A jest zamknięty ze względu na operację *rekursji prostej*, jeśli dla wszystkich f zdefiniowanych następująco (druga kolumna odnosi się do przypadku $\vec{x} = \langle\rangle$):

$$\begin{cases} f(0, \vec{x}) = g(\vec{x}), \\ f(S(n), \vec{x}) = h(n, f(n, \vec{x}), \vec{x}), \end{cases} \qquad \begin{cases} f(0) = a, \quad \text{dla } a \in \mathbb{N} \\ f(S(n)) = h(n, f(n)), \end{cases}$$

gdzie $g, h \in A$, zachodzi $f \in A$.

Schematy tworzenia nowych funkcji II

3. A jest zamknięty ze względu na operację *minimum efektywnego*, jeśli dla wszystkich f zdefiniowanych następująco:

$$f(\vec{x}) = (\mu m.g(\vec{x}, m) = 0),$$

gdzie $g \in A$ i $\forall \vec{x}.\exists m.g(\vec{x}, m) = 0$, zachodzi $f \in A$.

4. A jest zamknięty ze względu na operację *minimum*, jeśli dla wszystkich f zdefiniowanych następująco:

$$f(\vec{x}) = (\mu m.g(\vec{x}, m) = 0),$$

gdzie $g \in A$, zachodzi $f \in A$.

Klasa $\mathcal{P}$ funkcji *pierwotnie rekurencyjnych*

Definicja. Klasa $\mathcal{P}$ funkcji *pierwotnie rekurencyjnych* jest najmniejszą klasą funkcji numerycznych zawierających wszystkie funkcje bazowe i zamkniętą ze względu na operacje

(i) superpozycji,

(ii) rekursji prostej.

Klasy $\mathcal{R}_0$ i $\mathcal{R}$ funkcji ogólnie i częściowo rekurencyjnych

Definicja. Klasa $\mathcal{R}_0$ funkcji (ogólnie) rekurencyjnych jest najmniejszą klasą funkcji numerycznych zawierających wszystkie funkcje bazowe i zamkniętą ze względu na operacje

(i) superpozycji,

(ii) rekursji prostej,

(iii) minimum efektywnego.

Definicja. Klasa $\mathcal{R}$ funkcji częściowo rekurencyjnych jest najmniejszą klasą funkcji numerycznych zawierających wszystkie funkcje bazowe i zamkniętą ze względu na operacje

(i) superpozycji,

(ii) rekursji prostej,

(iii) minimum.

Dodawanie

Nieformalnie:

$$\begin{cases} add(0, n) = n \\ add(m + 1, n) = add(m, n) + 1 \end{cases}$$

Formalnie:

$$\begin{cases} add(0, n) = U_1^1(n) \\ add(S(m), n) = h(m, add(m, n), n) \end{cases}$$

gdzie:

$$h(x_1, x_2, x_3) = S(U_3^2(x_1, x_2, x_3))$$

Poprzednik

Specyfikacja:

$$\delta(n) = \begin{cases} n-1, & \text{dla } n > 0 \\ 0, & \text{dla } n = 0 \end{cases}$$

Definicja (nieformalna):

$$\begin{cases} \delta(0) = 0 \\ \delta(S(n)) = n \end{cases}$$

Formalnie:

$$\begin{cases} \delta(0) = 0 \\ \delta(S(n)) = U_2^1(n, \delta(n)) \end{cases}$$

Został tu zastosowany schemat rekursji prostej, w którym $\vec{x} = \langle\rangle$.

Odejmowanie

Specyfikacja:

$$m \dot{-} n = \begin{cases} m - n, & \text{dla } m \geq n \\ 0, & \text{dla } m < n \end{cases}$$

Definicja (nieformalna):

$$\begin{cases} m \dot{-} 0 = m \\ m \dot{-} S(n) = \delta(m \dot{-} n) \end{cases}$$

W tym i następnych przykładach definicje funkcji będą podawane nieformalnie, ale należy zdawać sobie sprawę, że jest to jedynie pewien skrót notacyjny, który w razie potrzeby należy umieć zastąpić definicją w pełni formalną, zgodną z wprowdzonymi wyżej schematami definiowania funkcji rekurencyjnych.

Rozpoznawanie zera

Specyfikacje:

$$sg(n) = \begin{cases} 0, & \text{dla } n = 0 \\ 1, & \text{dla } n > 0 \end{cases} \qquad \overline{sg}(n) = \begin{cases} 1, & \text{dla } n = 0 \\ 0, & \text{dla } n > 0 \end{cases}$$

Definicje:

$$\begin{cases} sg(0) = 0 \\ sg(S(n)) = 1 \end{cases} \qquad \overline{sg}(n) = 1 \dot{-} sg(n)$$

Funkcja Ackermanna

Funkcja $A(m, n)$, zdefiniowana za pomocą następującego układu równań, jest obliczalna:

$$A(0, n) = n + 1$$
$$A(m + 1, 0) = A(m, 1)$$
$$A(m + 1, n + 1) = A(m, A(m + 1, n))$$

Powyższa definicja zawiera podwójną rekursję, która jest nieco silniejsza, niż schemat rekursji prostej. Zauważmy jednak, że przy każdym wywołaniu rekurencyjnym jeden z argumentów maleje, więc po *skończonej* ilości wywołań obliczenia muszą się zakończyć.

Twierdzenie. Jeśli $f : \mathbb{N} \to \mathbb{N}$ jest pierwotnie rekurencyjna, to istnieje takie m, że dla wszystkich n, $f(n) < A(m, n)$, tzn. funkcja Ackermanna asymptotycznie rośnie szybciej, niż jakakolwiek funkcja pierwotnie rekurencyjna.

Mamy więc następujące

Twierdzenie. $\mathcal{P} \subsetneq \mathcal{R}_0 \subsetneq \mathcal{R}$.

- 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. W latach trzydziestych 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. Opracowany przez niego prototypowy język ISWIM wywarł wpływ na projektantów zarówno języków funkcjonalnych, jak i imperatywnych.
- Wykorzystując te rezultaty Christopher Strachey położył podstawy semantyki denotacyjnej języków programowania. Techniczne problemy rozwiązał amerykański logik Dana Scott, opracowując teorię dziedzin, która stanowi ważny rozdział informatyki teoretycznej.
- Curry i niezależnie Howard zauważyli odpowiedniość między rachunkem 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 funkcjonalnych.
- 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

Term bez zmiennych wolnych nazywamy *termem stałym*, *termem zamkniętym* lub *kombinatorem* (ang. ground term, combinator).
Najbardziej znane i najczęściej używane są kombinatory I, K, S.

$$\begin{aligned} \mathsf{I} &= \lambda x.x \\ \mathsf{K} &= \lambda xy.x \\ \mathsf{S} &= \lambda xyz.xz(yz) \end{aligned}$$

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 przez:

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

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

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

Przykład.

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

Reguły wnioskowania dla rachunku lambda I

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

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

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

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

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

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. Dowód równości termów $(\lambda xy.x)\ (\lambda z.z)$ i $(\lambda x.x)\ (\lambda yz.z)$.

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

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

My również przyjmiemy taką konwencję.

Redukcje

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

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

Relacja β-*redukcji* $\twoheadrightarrow_\beta$ jest zwrotnym i przechodnim domknięciem relacji $\rightarrow_\beta$.
Relacja β-*konwersji* $=_\beta$ jest relacją równoważności generowaną przez $\rightarrow_\beta$.

Twierdzenie 1. $\lambda \vdash M = N \Leftrightarrow M =_\beta N$.
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

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

Lemat 2. Niech M będzie w β-NF. Wówczas

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

Dowód. Oczywisty, jeśli $\twoheadrightarrow_\beta$ jest $\rightarrow_\beta$. Rezultat wynika z przechodniości.

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*, 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.
- **Redukcja aplikatywna**: wartościuj każdy argument dokładnie raz.

Strategie redukcji — przykłady I

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

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

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

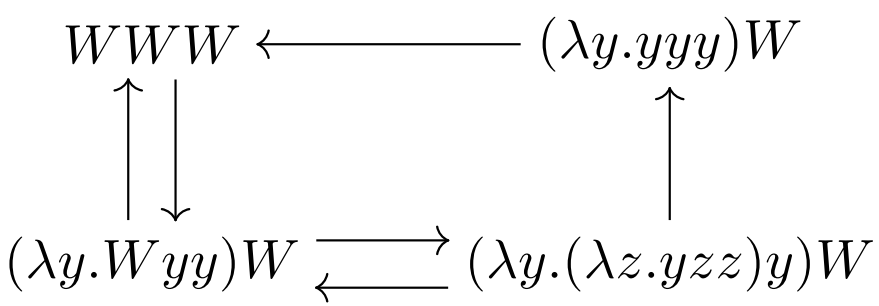
$$\{N \in \Lambda \mid M \twoheadrightarrow_R N\}$$

ukierunkowanym relacją redukcji $\rightarrow_R$. 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.

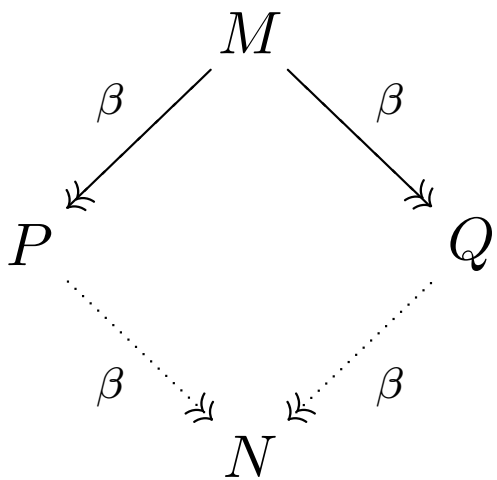
$G_\beta(\Omega)$, dla $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$

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



Twierdzenie Churcha-Rossera

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



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 2, 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 2.

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. $\lambda \not\vdash \mathbf{true} = \mathbf{false}$, gdzie $\mathbf{true} \equiv \lambda xy.x$ i $\mathbf{false} \equiv \lambda xy.y$. W przeciwnym razie $\mathbf{true} =_\beta \mathbf{false}$ na podstawie Tw.1, 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 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.

$$\lambda x.\underline{(\lambda yx.+xy)x} \rightarrow_\beta \lambda x.\lambda z.+zx$$

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, np.

$$\underline{(\lambda x.(\lambda yx.+xy)x)5} \rightarrow_\beta \underline{(\lambda yx.+xy)5} \rightarrow_\beta \lambda x.+x5$$

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

Literatura (wybrane pozycje) I

- H.P.Barendregt, *The Lambda Calculus. Its Syntax and Semantics*, Elsevier, Amsterdam 1984 (revised edition).
- 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

- M.Felleisen, M.Flatt, *Programming Languages and Lambda Calculi*, draft 2006, http://www.cs.utah.edu/plt/publications/pllc.pdf
- J.R.Hindley, J.P.Seldin, *Lambda-calculus and Combinators . An Introduction* , Cambridge University Press, Cambridge 2008
- Ch.Hankin, *Lambda Calculi. A Guide for Computer Scientists*, Oxford University Press, Oxford 1994
- J.R.Hindley, *Basic Simple Type Theory*, Cambridge University Press, Cambridge 1997
- J.L. Krivine, *Lambda-Calculus, Types and Models*, Masson, Paris 1993
- P. Urzyczyn, *Rachunek lambda, wykład monograficzny*, http://www.mimuw.edu.pl/~urzy/Lambda/

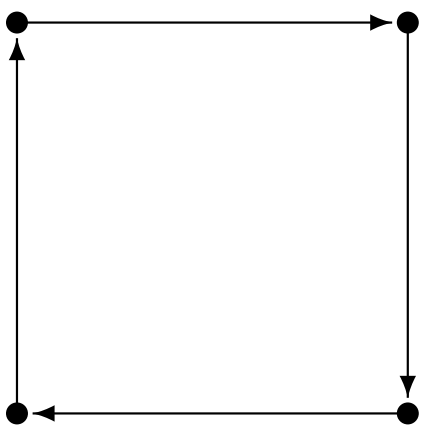
1. W poniższym termie wskaż wszystkie beta-redeksy.
   $(\lambda x.x)((\lambda x.x)(\lambda z.(\lambda x.x)z))$
2. Przeprowadź normalizację poniższego termu. Zwróć uwagę na konieczność zmiany nazwy zmiennej związanej.
   $(\lambda x.xx)(\lambda yz.yz)$
3. Przeprowadź normalizację poniższych termów, jeśli 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

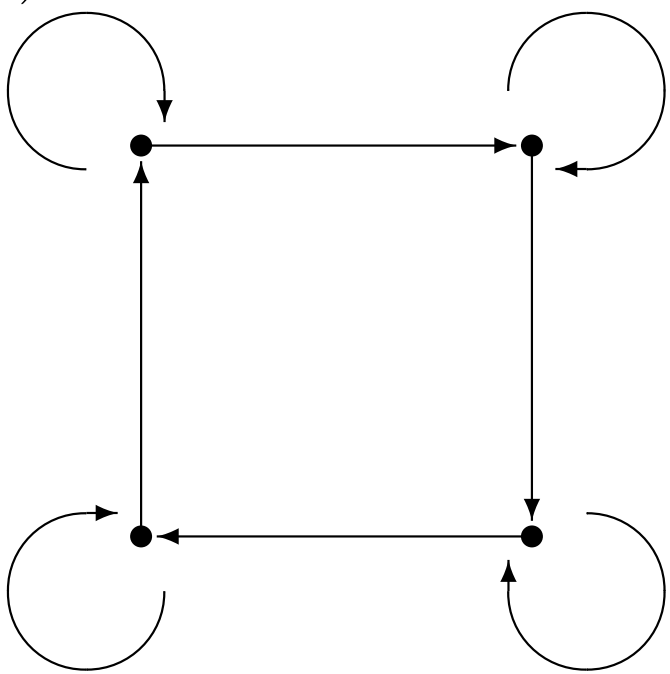
$$M \equiv \lambda x.(\lambda y.yy)x$$

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).

7. Pokaż, że poniższe funkcje są pierwotnie rekurencyjne.
   (a) $m \cdot n$ = iloczyn argumentów
   (b) $m^n = m$ do potęgi n, gdzie $0^0 = 1$
   (c) $\min(m, n)$ = mniejszy z argumentów
   (d) $\max(m, n)$ = większy z argumentów
   (e) $\min_k(n_1, \ldots, n_k)$
   (f) $\max_k(n_1, \ldots, n_k)$
   (g) $|m - n|$ = wartość bezwzględna różnicy
   (h) $n!$
   (i) $m \,\%\, n$ = reszta z dzielenia m przez n (gdzie $m\%0 = m$)
   (j) $\lfloor m/n \rfloor$ = część całkowita ilorazu (gdzie $\lfloor m/0 \rfloor = 0$)

   Dla wszystkich wartości argumentów zachodzi zwykły związek $m = \lfloor m/n \rfloor + m\%n$.

8. Udowodnij, że funkcja

$$\sum_{n<k} f(n,\vec{x}) = \begin{cases} 0, & \text{gdy } k = 0 \\ f(0,\vec{x}) + \cdots + f(k-1,\vec{x}), & \text{gdy } k > 0 \end{cases}$$

jest pierwotnie rekurencyjna (przy założeniu, że f jest pierwotnie rekurencyjna).

9. Udowodnij, że klasa funkcji pierwotnie rekurencyjnych jest zamknięta ze względu na definicje warunkowe

$$f(\vec{x}) \quad = \quad \begin{cases} h_1(\vec{x}), & \text{gdy } R_1(\vec{x}) \\ h_2(\vec{x}), & \text{gdy } R_2(\vec{x}) \\ \ldots\ldots\ldots\ldots & \\ h_m(\vec{x}), & \text{gdy } R_m(\vec{x}) \end{cases}$$

gdzie h_i i funkcje charakterystyczne relacji R_i $(1 \leq i \leq m)$ są pierwotnie rekurencyjne, a relacje $R_i(n,\vec{x})$ wykluczają się wzajemnie oraz wyczerpują wszelkie możliwości.