Wykład 11

Elementy rachunku lambda

Plan wykładu

- Wprowadzenie
- Operator abstrakcji λ
- Notacja λ
- Funkcje wielu zmiennych w rachunku λ
- Liczby w rachunku λ
- Operacje arytmetyczne w rachunku λ
- Funkcje nieobliczalne w rachunku λ
- Podsumowanie

Pojęcia pomocnicze

Operatorem nazywamy symbol (funkcyjny), który w połączeniu z pewnym wyrażeniem (napisem) zawierającym zmienną (wyróżnioną jako związaną tym operatorem) nabiera konkretnego znaczenia.

Np.: kwantyfikatory, całki, nieskończone sumy i produkty itp.

$$\boxed{\sum_{x=1}^{\infty} F(x,y) = F(1,y) + F(2,y) + \ldots}$$

Pojęcia pomocnicze

Operatorem abstrakcji (znakiem abstrakcji) nazywa się symbol operacji, która przekształca funkcję zdaniową w nazwę zbioru przedmiotów spełniających tę funkcję.

$\{x \mid \underbrace{x \text{ jest małpą}}_{\text{funkcja zdaniowa}}\}$

Pojęcia pomocnicze

Operator lambda (λ-operator) stanowi uogólnienie operatora abstrakcji, gdyż stosuje się go nie tylko do funkcji zdaniowych lecz również do wyrażeń zbudowanych z symboli funkcyjnych (termów).

Obiekty w rachunku λ

a, b, c, ..., z, a', b', ..., z', a'', b'', ..., z'', a''', ...

Funkcja b działa na funkcję c

a = bc

Wynikiem jest funkcja a

Syntaktyka rachunku λ

λ x. [fx]

Syntaktyka rachunku λ

| (λx.[fx]) | λy.[gzy] |

LM

Syntaktyka rachunku λ

| λx. | [λy.[gxy]] |

λx.[L]

Język rachunku lambda

- Alfabet A rachunku λ (zbiór identyfikatorów)

A = {a,b,c,...,z,a',b',...,z',a'',b'',...,z'',a''',....} ∪ {λ, (,)}

- ⟨wyrażenie λ⟩::= ⟨identyfikator⟩| (⟨wyrażenie λ⟩) ⟨wyrażenie λ⟩| λ ⟨wektor zmiennych związanych⟩.[⟨wyrażenie λ⟩]
- ⟨wektor zmiennych związanych⟩::= ⟨identyfikator⟩| ⟨identyfikator⟩{, ⟨identyfikator⟩}

Podwyrażenie wyrażenia G

(1) G jest podwyrażeniem G

(2) jeśli G ≡ LM, to każde podwyrażenie wyrażeń L i M jest także podwyrażeniem G,

(3) jeśli G ≡ λx.[L], to każde podwyrażenie L jest także podwyrażeniem G.

Przykłady:

λy.[x] i x są podwyrażeniami λy.[x]

λz.[xyz] i xyz są podwyrażeniami λy.[λz.[xyz]]

Zmienne wolne w wyrażeniach

(1) x jest jedyną zmienną wolną występującą w wyrażeniu x

(2) jeśli P' i S' są zbiorami zmiennych wolnych występujących odpowiednio w wyrażeniach P i S, to zbiór zmiennych wolnych występujących w aplikacji PS jest równy P' ∪ S',

(3) jeśli P' jest zbiorem zmiennych wolnych występujących w wyrażeniu P, to P' \ {x} jest zbiorem zmiennych wolnych występujących w wyrażeniu λx.[P].

Zmienne wolne w wyrażeniu λ

Przykłady:

x jest zmienną wolną w:	y jest zmienną związaną w:
λy.[x]	λy.[x]
λyz.[xyz]	λy.[λz.[xyz]]
ab(λz.[zb])λu.[x]	
	ale wolną w podwyrażeniu:
	λz.[xyz]

Wyrażenie wolne ze względu na podstawienie

M ≡ λxy.[xy] εu = λv.[u] λv.[λxy.[xy]] Zmienne wolne w M pozostają wolne.	M ≡ xz εu = λx.[u] λx.[xz] Zmienna x wolna w M zostaje związana.

Wyrażenie wolne ze względu na podstawienie

niech M i εx będą λ-wyrażeniami;

εx oznacza, że zmienna x jest zmienną wolną w εx lub że nie występuje w εx;

zapis ε(M/x) oznacza rezultat zastąpienia wszystkich wolnych wystąpień zmiennej x przez wyrażenie M;

wyrażenie M jest wolne po zastąpieniu nim zmiennej x, jeśli podczas zastępowania nie nastąpią przypadkowe związania zmiennych wolnych występujących w M.

Redukcje w rachunku λ

$$(\underbrace{\lambda\ x.\ [\ fx\ ]}_{f})a = f\ a$$

$$\sin = \lambda x.\left[x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \ldots\right]$$

Redukcje w rachunku λ

λf. [f(fx)]

λf. [λx. [f(fx)]]

Elementarne reguły redukcji

Reguła przemianowywania zmiennych $\xrightarrow{\alpha}$

Niech A ≡ λx. [*F*x] oraz niech y będzie nową zmienną, która nie występuje w *F*x jako zmienna wolna i pozostaje zmienna wolną po zastąpieniu nią zmiennej x w *F*x, wtedy:

$$\lambda x.[Fx] \rightarrow \lambda y.[Fy]$$

Na przykład: $\lambda x\underbrace{\lambda z.[xz]}_{F[x]} \xrightarrow{\alpha} \lambda y\lambda z.[yz]$

Elementarne reguły redukcji

Podstawowa reguła redukcji $\xrightarrow{\beta}$

Niech $A \equiv (\lambda x.[\varepsilon x])M$ i wyrażenie M jest wolne ze względu na zastąpienie nim zmiennej x w wyrażeniu εx, wtedy:

$$(\lambda x.[\varepsilon x])M \xrightarrow{\beta} \varepsilon[M/x]$$

Na przykład: $(\lambda x.[\underbrace{xz]}_{\varepsilon x})\underbrace{a}_{M} \xrightarrow{\beta} (az)$

Elementarne reguły redukcji

Reguła pomocnicza $\xrightarrow{\eta}$

Niech $A \equiv \lambda x.[Fx]$ gdzie F jest wyrażeniem, w którym zmienna x nie występuje jako zmienna wolna, wtedy:

$$\lambda x.[Fx] \xrightarrow{\eta} F$$

Na przykład: $\lambda x.[\underbrace{(\lambda y.[y])}_{F}x] \xrightarrow{\beta} \lambda x.[x] \xrightarrow{\alpha} \underbrace{\lambda y.[y]}_{F}$

$$\lambda x.[(\lambda y.[y])x) \xrightarrow{\eta} \lambda y.[y]$$

Redukcja wyrażeń

niech L i M będą wyrażeniami, wtedy przez $L \to M$ oznaczymy elementarną redukcję L do M;

wyrażenie jest redukowalne do M, co zapisujemy $L \Rightarrow M$, jeśli istnieje ciąg $L = L_0, L_1, ..., L_{n-1}, L_n = M$ taki, że $\forall_{(0 \le i \le n-1)}\ L_i \to L_{i+1}$;

Funkcje wielu zmiennych

$\boxed{\text{funkcja dwóch zmiennych}}$ $\boxed{f(a_1)a_2}$

$f_2: A_1 \times A_2 \to B$ $f_{c2}: A_1 \to (A_2 \to B)$

$f_2(a_1, a_2) = b$ $f_{c2}(a_1, a_2) = f_{a1}(a_2) = f(a_1)(a_2)$

$\boxed{\text{funkcja trzech zmiennych}}$ $\boxed{((fa_1)a_2)a_3}$

$f_3: A_1 \times A_2 \times A_3 \to B$ $f_{c3}: A_1 \to (A_2 \to (A_3 \to B))$

$f_3(a_1, a_2, a_3) = b$ $f_{c3}(a_1, a_2, a_3) = f(a_1)(a_2, a_3) = f(a_1)(a_2)(a_3)$

Technika Curry'ego - currying (Schönfinkel)

$\boxed{\text{funkcja n zmiennych}}$

$f_n: A_1 \times A_2 \times ... \times A_n \to B$

$f_n(a_1, a_2, ..., a_n) = b$

$\boxed{(...((fa_1)a_2)a_3)...a_n)}$

$f_{cn}: A_1 \to (A_2 \to ... \to (A_n \to B)...)$

$f_{cn}(a_1, a_2, a_3, ..., a_n) = f_{c,n}(a_1)(a_2)...(a_n)$

Liczby w rachunku λ

$0 = \lambda fx.\ [\ x\ ]$
$1 = \lambda fx.\ [\ fx\ ]$
$2 = \lambda fx.\ [\ f(fx)\ ]$
$3 = \lambda fx.\ [\ f(f(fx))\ ]$
$4 = \lambda fx.\ [\ f(f(f(fx)))\ ]$

....

f(x) = x + 1

S = λabc.[b((ab)c)]

S**3** = (λabc.[b((ab)c)])**3**

= λbc.[b((**3**b)c)]

(3b)c = b(b(bc))

= λbc.[b(b(b(bc)))]

λbc.[b(b(b(bc)))] = 4

= 4

Teoretyczne podstawy informatyki

f(x)= 2x

D = λabc.[(ab)((ab)c)]

D**3** = (λabc.[(ab)((ab)c)])**3**

= λbc.[(**3**b)((**3**b)c)]

(3b)c = b(b(bc))

= λbc.[(**3**b)(b(b(bc)))] = λbc.[b(b(b(b(b(bc)))))]

= 6   1 2 3 4 5 6

Teoretyczne podstawy informatyki

f(x, y) = x^y

P = λfg.[fg]

(P**2**)**3** = ((λfg.[fg])**2**)**3**

2 = λfx.[f(fx)]

= (λg.[**2**g])**3** = (λg.[λfx.[f(fx)] g])**3**

= (λg.[λx.[g(gx)]])**3** = (λgx. [g(gx)])**3**

3 = λfx.[f(f(fx))]

= λx. [3(3x)]= λx. [λfy.[f(f(fy))](3x)]

= λx. [λy.[(3x)((3x)((3x)y))]] = λxy.[(3x)((3x)(3xy))]

= λxy.[3x(3x(x(x(xy))))] = λxy.[x(x(x(x(x(x(x(x(xy))))))))]

Teoretyczne podstawy informatyki

Definicja operacji arytmetycznych w rachunku λ

Dodawanie

A = λfgxy.[((fx)(gx))y]

Mnożenie

M = λfgx.[f(gx)]

Potęgowanie

P = λfg.[fg]

Teoretyczne podstawy informatyki

Przykłady

function f(x: integer): integer;
begin f := x end;

$P_1^1(x) = x$;
definicja funkcji f w rachunku lambda: λx.[x];
przykładowe obliczenia dla wywołania: ... y:=f(5);...

$(\lambda\ \underbrace{x.[x]}_{\varepsilon x})\ \overbrace{5}^{M} \xrightarrow{\beta}$

Teoretyczne podstawy informatyki

Przykłady

function f_1(x: integer): integer;
begin f_1 := x + 1 end;

$f_1 \equiv S_1$; (funkcja następnika)
definicja funkcji f_1 w rachunku lambda: λabc.[(ab)(bc)];
przykładowe obliczenia dla wywołania: ... y:=f_1(5);...

$(\lambda abc.[\underbrace{(ab)(ab)c}_{\varepsilon a}]\overbrace{)5}^{M} \xrightarrow{\beta}$

Teoretyczne podstawy informatyki

Przykłady

function f_2(x, y: integer): integer;
begin f_2 := x end;

$f_2 \equiv P_1^2$; (projekcja)
definicja funkcji f_2 w rachunku lambda: (λxy.[y]);
przykładowe obliczenia dla wywołania: y:=f_3(3,4);

$$((\underbrace{\lambda xy.[y]}_{\varepsilon x})\overbrace{4}^{M})3 \xrightarrow{\beta} \quad \xrightarrow{\beta}$$

Teoretyczne podstawy informatyki

Przykłady

function f_3(function f; x: integer): integer;
begin f_3 := f(x) end;

definicja funkcji f_3 w rachunku lambda: λfx.[fx];
przykładowe obliczenia dla wywołania: y:=f_3(f_1,5);

((λfx.[fx])λabc.[(ab)(bc)])5 → (λx.[(λabc.[(ab)(bc)])x]])5
→ (λabc.[(ab)(bc)])5

Teoretyczne podstawy informatyki

Wyrażenie G, w którym nie można zastosować redukcji typu β lub η jest wyrażeniem <u>zredukowanym</u> (lub wyrażeniem w postaci normalnej);

Przykład wyrażenia nieredukowalnego

λx.[xx]λx.[xx] $\xrightarrow{\beta}$ λx.[xx]λx.[xx] $\xrightarrow{\beta}$ ....

Teoretyczne podstawy informatyki

Przykład wyrażenia, dla którego istnieją dwa ciągi redukcji: skończony i nieskończony

(λxy.[y])(λx.[xx])λx.[xx] λy.[y]

(λxy.[y])(λx.[xx])λx.[xx] ((λxy.[y])(λx.[xx]))λx.[xx]

Teoretyczne podstawy informatyki

Twierdzenie

Jeśli dla danego wyrażenia istnieją dwa ciągi redukcji, z których jeden jest nieskończony, drugi zaś daje określoną wartość, to wyrażenie to zawiera podwyrażenie redukowalne do postaci LM, przy czym M jest nieredukowalne, L ma postać λx.[P], gdzie P ma postać zredukowaną i nie zawiera x.

$$\boxed{\lambda x.[P]M}$$

(λxy.[y])(λx.[xx])λx.[xx]

Teoretyczne podstawy informatyki

Twierdzenie

Jeśli wyrażenie oblicza się w drodze kolejnych skrajnie lewostronnych redukcji operator-argument, to uzyskany ciąg redukcji kończy się, jeśli wyrażenie jest redukowalne.

Teoretyczne podstawy informatyki

Twierdzenie (Churcha-Rossera o zbieżności)

Jeśli dane wyrażenie ma dwa ciągi redukcji prowadzące do postaci zredukowanych, to postaci te są sobie równe z dokładnością do przemianowania zmiennych (znajdują się w tej samej klasie wartości).

Model rachunku lambda

$\boxed{(\lambda u.[u])u}$

val{(λu.[u])u} = *selfapplic*

Dana Scott (1970) - model rachunku λ

$$D \cong (D \rightarrow D)$$

gdzie $(D \rightarrow D)$ jest przestrzenią funkcji obliczalnych z D w D.

Izomorfizm z D w $(D \rightarrow D)$ oznacza, że D jest takim zbiorem wartości, który jest potrzebny do modelu rachunku λ.

D jest kratą zupełną, a funkcje obliczalne są definiowane jako funkcje ciągłe w rozumieniu teorii krat. Każda funkcja ciągła z D w D ma punkt stały (Tarski, Knaster).

Język LISP (1960)

Lista składa się z:

- nazwy funkcji (**car**)
- i listy argumentów (**cdr**).

Język LISP

define (ROZ (**lambda**(X)

cond ((**null** X) **nil**)

((**atom** X) X)

((**atom** (**car** X))(**cons** (**car** X) (ROZ (**cdr** X))))

(T(**append** (ROZ (**car** X)) (ROZ (**cdr** X))))))))