Programowanie dowodami

Radosław Rowicki

30 kwietnia 2024

1 Intro

Zasady:

1. Nie szukamy odpowiedzi w internecie, bo one tam są i dość łatwo na nie wpaść choćby przypadkiem.
2. Można śmiało pisać pytania na gmaila `radrowicki+www20@`. Szczególnie jak coś jest niejasne, potrzebuję feedbacku.
3. Jeśli chcesz rozwiązywać zadania z lambd w jakimś języku programowania, nie mam z tym problemu.

2 Indukcja

Zapomnijmy na razie o całej arytmetyce, jaką znamy. Wyobraźmy sobie, że żyjemy w świecie, gdzie w matematyce znane są jedynie funkcje, zbiory i logika.

Zdecydowanie brakuje nam tu liczb. Żeby spełnić ludzkie minimum socjalne, potrzebujemy chociażby tych "naturalnych", takich jak dwadzieścia, jeden, trzydzieści, osiem. Jako herosi naszej wymyślonej krainy postanawiamy coś z tym zrobić i uratować ludzkość od wiecznego nieliczenia.

Od czegoś trzeba zacząć — zatem zacznijmy od niczego. Zdefiniujmy sobie *zero*, lub 0, jako *„nic"*. Niestety *nic* to zbyt mało — na świecie bywają różne obiekty, które bynajmniej nie są *niczym*. Potrzebujemy więc jakiegoś narzędzia do ich zliczania. Utwórzmy zatem do tego pewien zbiór, który nazwiemy majestatycznie zbiorem *liczb naturalnych* i będziemy oznaczać go $\mathbb{N}$. Określimy teraz 5 *aksjomatów*, czyli prawd przyjętych w ramach definicji naszego zbioru:

1. 0 jest liczbą naturalną ($\in \mathbb{N}$)
2. Dla każdej liczby naturalnej n istnieje inna liczba naturalna, która jest jej *następnikiem* i oznaczymy ją $S(n)$
3. 0 nie jest następnikiem żadnej liczby naturalnej
4. **Aksjomat indukcji:** Jeśli
   - 0 ma cechę P
   - Z faktu że n ma cechę P wynika że $S(n)$ ma cechę P

   to każda liczba naturalna ma cechę P

Super! Mamy liczby! Możemy teraz się nimi bawić: niech $1 = S(0)$, $2 = S(1) = S(S(0))$ itd. Jednak wciąż brakuje nam wielu innych ważnych rzeczy. Nie znamy dodawania, mnożenia, odejmowania i masy prawd, które w realnym świecie są oczywiste. Ze zdefiniowaniem dwóch pierwszych chętnie Ci pomogę:

Dodawanie:

$$a + 0 = a$$
$$a + S(b) = S(a + b)$$

Mnożenie:

$$a \times 0 = a$$
$$a \times S(b) = a \times b + a$$

Resztę jednak zostawię tobie.

Zadania:
Zad. 1 *(1 pkt.)* Udowodnij, że $a + 1 = S(a)$
Zad. 2 *(1 pkt.)* Udowodnij, że $0 + a = a$ (to nie jest część definicji dodawania!)
Zad. 3 *(1 pkt.)* Udowodnij, że $2 + 2 = 4$. Przyjmijmy $4 = S(S(S(S(0))))$
Zad. 4 *(1 pkt.)* Udowodnij, że $2 \times 2 = 4$
Zad. 5 *(1 pkt.)* Udowodnij, że $a \times 1 = a$
Zad. 6 *(1 pkt.)* Określ, kiedy jedna liczba jest mniejsza (bądź mniejsza-równa) od drugiej
Zad. 7 *(2 pkt.)* Zdefiniuj moduł (wartość bezwzględną) z różnicy dwóch liczb (nie mamy liczb całkowitych, stąd moduł a nie zwykłe odejmowanie)
Zad. 8 *(2 pkt.)* Udowodnij, że dodawanie jest łączne, tj $(a + b) + c = a + (b + c)$
Zad. 9 *(2 pkt.)* Udowodnij, że dodawanie jest przemienne, tj $a + b = b + a$
Zad. 10 *(2 pkt.)* Udowodnij, że mnożenie jest łączne
Zad. 11 *(2 pkt.)* Udowodnij, że mnożenie jest przemienne
Zad. 12 *(3 pkt.)* Udowodnij, że $(a + b) \times c = a \times c + b \times c$

Czasami trzeba dobrze wybrać zmienną do indukcji. Jak nie idzie, to zazwyczaj warto spróbować innej. W razie kompletnej porażki mogę dać hinta.

Rozkmina: Jak wyobrażasz sobie definicję liczb całkowitych i wymiernych w naszym systemie? Czy możliwe jest określenie liczb rzeczywistych? Jeśli tak, to w jaki sposób? Jeśli nie, to dlaczego? Czy potrzeba do tego więcej aksjomatów? Dokąd nocą tupta jeż?

3 Lambada

Poznajcie lambdę. Say hi

$$\lambda$$

Ale brzydka. Po co ona tu przyszła? A po to, żeby wam uprzykrzyć życie. Zagrajmy więc w grę.

3.1 Zasady zabawy

Będziemy robić operacje na napisach. Interesują nas tylko i wyłącznie napisy które opisują tak zwane *lambda termy*. Napis jest lambda termem wtw pasuje do jednego z schematów:

1. **Zmienna:** x (słowo zaczynające się małą literą)
2. **Abstrakcja:** $\lambda\ x\ .\ T$ (lambda oraz kropka są elementami składni, x to zmienna, a T to dowolny lambda term)
3. $(T_1)\ T_2$ — aplikacja (dwa dowolne lambda termy obok siebie, lewy w nawiasie)
4. (T) — nawias (a konkretnie lambda term w nawiasie)
5. $(T_1)[x/T_2]$ — podstawienie (x to zmienna, T_1T_2 to lambda termy, reszta to składnia)

Przykład lambda termów

$$lol$$

$$(lo)\ l$$

$$\lambda\ x\ .\ x$$

$$(alama)\ kota$$

$$(\lambda\ x\ .\ (x)\ lol)\ (\lambda\ kek\ .\ kok)\ misio$$

$$((\lambda\ x\ .\ (x)\ lol)\ \lambda\ kek\ .\ kok)\ misio$$

$$(\lambda\ x\ .\ y)[x/(y)\ y]$$

Przepisywanie Definiujemy następujące zasady przepisywania napisów:

Nawiasy:

1. $((T)) \to (T)$
2. $(x) \to x$

Zaglądanie:

1. $(T_0) \to (T_1)$ jeśli $T_0 \to T_1$
2. $(T_0)\ T_1 \to \lambda\ T_0\ .\ T_1'$ jeśli $T_1 \to T_1'$
3. $(T_0)\ T_1 \to \lambda\ T_0'\ .\ T_1$ jeśli $T_0 \to T_0'$
4. $\lambda\ x\ .\ T_0 \to \lambda\ x\ .\ T_1$ jeśli $T_0 \to T_1$
5. $(T_1)[x/T_3] \to (T_2)[x/T_3]$ jeśli $T_1 \to T_2$

Podstawianie:

1. $(x)[x/T] \to T$
2. $(x)[y/T] \to x$ jeśli $x \neq y$

3. $((T_1)\ T_2)[x/T_3] \to ((T_1)[x/T_3])\ (T_2)[x/T_3]$
4. $(\lambda\ x\ .\ T_0)[x/T_1] \to \lambda\ x\ .\ T_0$
5. $(\lambda\ x\ .\ T_0)[y/T_1] \to \lambda\ x\ .\ (T_0)[y/T_1]$ jeśli $x \neq y$ oraz x nie występuje w T_1

Redukcje, konwersje:

1. α-konwersja: $\lambda\ x\ .\ T \to \lambda\ y\ .\ (T)[x/y]$ jeśli y nie występuje w T
2. β-redukcja: $(\lambda\ x\ .\ T_1)\ T_2 \to (T_1)[x/T_2]$
3. η-redukcja: $\lambda\ x\ .\ (f)\ x \to f$

Wiele kroków:

1. $T \xrightarrow{*} T$
2. $T_1 \xrightarrow{*} T_3$ jeśli istnieje taki T_2 że $T_1 \to T_2$ oraz $T_2 \xrightarrow{*} T_3$

Umawiamy się że napisanie $T_1\ T_2$ (z wyraźną spacją) implicite oznacza $(T_1)\ T_2$. Tak samo wielokrotne aplikacje jak $T_1\ T_2\ T_3$ oznaczają $((T_1)\ T_2)\ T_3$. Wszystko pod warunkiem że nie wpływa to na przepisania i składnia jest jasna. Przykładowo w przypadku $(\lambda\ x\ .\ x)\ y$ nawias należy pisać, natomiast powinno być oczywiste że $f\ x$ oznacza $(f)\ x$
Dodatkowo, niech $\lambda\ x\ y\ z\ .\ T$ oznacza $\lambda\ x\ .\ \lambda\ y\ .\ \lambda\ z\ .\ T$.

Intuicja (czyli jak o tym myśleć)

- Term $\lambda\ x\ .\ T$ określa **anonimową funkcję** której argumentem jest x, a ciałem T. Przykładowo, nieco odbiegając od założeń naszego języka, stwierdzenie $\lambda\ x\ .\ x + 2$ to ekwiwalent znanej z liceum funkcji liniowej, po prostu bez nazwy (λ to nie jest nazwa funkcji, tylko konstrukcja składniowa!). Odpowiednikiem $f(x) = x + 2$ byłoby coś w stylu $f = \lambda\ x\ .\ x + 2$.
- Term $(f)\ x$ to aplikacja funkcji. W matematyce nawiasujemy to na odwrót: $f(x)$. Przykładowo, podążając za poprzednim przykładem, $f(3) = (\lambda\ x\ .\ x + 2)\ 3 \xrightarrow{*} 3 + 2 = 5$. Reguła β-redukcji odpowiada za logikę aplikacji funkcji (podstawienie argumentu pod zmienną).
- α-konwersja to nic innego jak zmiana nazwy argumentu funkcji. Wszak $f(x) = x + 2$ oraz $f(y) = y + 2$ to to samo. Jedyne czego nie można robić to podmieniać na zmienną która używana jest w ciele funkcji, na przykład $f(x) = x + y$ oraz $f(y) = y + y$ to nie do końca to samo.
- Nasze funkcje są zawsze jednoargumentowe. Jeśli chcemy mieć funkcję dwuargumentową, definiujemy funkcję która bierze pierwszy argument i zwraca funkcję która bierze drugi argument i dopiero zwraca wynik. Przykładowo, znowu naginając język, funkcja która dodaje dwie liczby by wyglądała jakoś tak: $\lambda\ x\ .\ \lambda\ y\ .\ x + y$. Nasza umowna notacja powinna robić większy sens: $\lambda\ x\ y\ .\ x + y$.

Oczywiście w naszym języku nie ma takich rzeczy jak liczby czy dodawanie, użyłem ich tu tylko do zobrazowania jakiegoś schematu myślenia. Ale nie martw się, dojdziemy do arytmetyki niebawem!

Przykłady

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

$$(\lambda\ x\ .\ x)\ y \xrightarrow{*} y$$

$$\begin{aligned}
(\lambda\ x\ .\ \lambda\ f\ .\ (f)\ x)\ f \to & \qquad \textit{(beta redukcja)} \\
(\lambda\ f\ .\ (f)\ x)[x/f] \to & \qquad \textit{(alfa konwersja, bo konflikt na f)} \\
(\lambda\ g\ .\ (g)\ x)[x/f] \xrightarrow{*} & \qquad \textit{(podstawienia)} \\
(\lambda\ g\ .\ (g)\ x) & \qquad \textit{(koniec)}
\end{aligned}$$

3.2 LaTeX

Jak chcesz wysyłać rozwiązania w latexu, to może ci się przydać moja brzydka preambuła:

```
\usepackage{suffix}
\usepackage{xcolor}
\usepackage[utf8]{inputenc}
\usepackage{polski}
\usepackage{amsthm}
\usepackage{amsfonts}
\usepackage{listings}
\usepackage{scalerel}
\usepackage{syntax}
\usepackage{amsmath}
\usepackage[inline]{enumitem}
\usepackage{chngcntr}

\newcommand{\lsub}[3]{(#1)[#2 / #3]}        % Podstawienie
\newcommand{\labs}[2]{\lambda~{#1}~.~{#2}}  % Abstrakcja
\newcommand{\lapp}[2]{({#1})~{#2}}          % Aplikacja
% \to                                       % Przepisanie
\WithSuffix\newcommand\to*{\xrightarrow{*}} % Przepisanie (wiele kroków)
```

3.3 Zadanka

W rozwiązaniach musicie zawrzeć wszystkie kroki przepisywania z reguł „Redukcje i konwersje". Całą resztę możecie zostawić „w pamięci". Nie musicie uzasadniać warunków używania reguł (są dość oczywiste).

3.3.1 Przepisujemy

Przepisz następujące termy tak żeby się już nie dało ich przepisywać (oprócz reguły α-konwersji, którą można odpalić prawie zawsze):

Zad. 13 *(1 pkt.)* $((\lambda\ x\ .\ \lambda\ y\ .\ (y)\ x)\ w)\ (\lambda\ z\ .\ z)$
Zad. 14 *(1 pkt.)* $((\lambda\ x\ .\ \lambda\ y\ .\ y\ x)\ y)\ (\lambda\ x\ .\ x)$

Zad. 15 *(1 pkt.)* $((\lambda\ y\ .\ y)\ \lambda\ x\ .\ x\ x)\ (\lambda\ z\ .\ (z)\ q)$
Zad. 16 *(2 pkt.)* $((\lambda\ z\ .\ z)\ \lambda\ z\ .\ (z)\ z)\ (\lambda\ z\ .\ (z)\ y)$
Zad. 17 *(2 pkt.)* $(\lambda\ a\ b\ c\ \ .\ c\ b\ a)\ z\ z\ (\lambda\ w\ .\ v\ w)$
Zad. 18 *(2 pkt.)* $(\lambda\ x\ y\ z\ \ .\ x\ z\ (y\ z))\ (\lambda\ x\ .\ z)\ (\lambda\ x\ .\ a)$

Nieskończoność Tutaj uzasadnij poprawność swojego rozwiązania:

Zad. 19 *(3 pkt.)* Wymyśl term który można przepisywać bez końca w taki sposób, żeby dało się użyć zasady β-redukcji nieskończenie wiele razy. Ponadto, ma się go nie dać przepisać (nawet wieloma krokami) tak żeby się go nie dało go dalej nieskończenie przepisywać w sposób opisany powyżej. Innymi słowy, zawsze kiedyś ma się dać zrobić β.
Zad. 20 *(3 pkt.)* Wymyśl term który można przepisywać bez końca w taki sposób, żeby on generalnie ciągle rósł. Nie musi być dłuższy po każdym jednym przepisaniu, może nawet czasem zmaleć, ważne by dało się go przepisywać tak, by mógł osiągnąć dowolny rozmiar. Długość termu mierzymy w ilości zmiennych jakie w nim występują.
Ponadto, ma się go nie dać przepisać (nawet wieloma krokami) tak żeby się go nie dało go przepisywać w sposób opisany powyżej. Innymi słowy, dla dowolnego n, zawsze kiedyś ma się dać go przepisać tak, by był dłuższy niż n.

3.3.2 Programowanie

Nadamy teraz sens niektórym termom. Behold, oto definiuję czym jest **PRAWDA**:

$$\lambda\ x\ y\ .\ x$$

Przekonujące, co? No to analogicznie **FAŁSZ**:

$$\lambda\ x\ y\ .\ y$$

Dobra, to teraz wyjaśniam o co chodzi. Jak wam napiszę

`if` alamakota `then` kotmaale `else` alaunikapodatków

To każdy zrozumie. To teraz wyobrażmy sobie że alamakota to **prawda**.

`if` $(\lambda\ x\ y\ .\ x)$ `then` kotmaale `else` alaunikapodatków

Teraz poprawmy kolorki...

`if` $(\lambda\ x\ y\ .\ x)$ `then` kotmaale `else` alaunikapodatków

Jeszcze trochę...

`if` $(\lambda\ x\ y\ .\ x)$ `then` kotmaale `else` alaunikapodatków

Mała β-redukcja...

kotmaale

Wyszło! Analogicznie, jak zamienimy alamakota na **fałsz**, to nasze wyrażenie aktywuje `else`:

`if` $(\lambda\ x\ y\ .\ y)$ `then` kotmaale `else` alaunikapodatków

$\rightarrow$

alaunikapodatków

Zatem nasz wspaniały `bool` (prawda/fałsz) to po prostu wyrażenie `if` (z obowiązkowym `else`) które pozwala nam podjąć decyzję w zależności od tego czy coś jest prawdą czy nie. W pewnym sensie to kwintesencja wartości logicznej — bo co innego można z nią zrobić tak u fundamentu? No to twoja kolej na zabawę:

Zad. 21 *(1 pkt.)* Napisz lambda term który będzie się zachowywał jak logiczna negacja. To znaczy takie T, że $T\ (\lambda\ x\ y\ .\ x) \xrightarrow{*} \lambda\ x\ y\ .\ y$ oraz $T\ (\lambda\ x\ y\ \ .\ y) \xrightarrow{*} \lambda\ x\ y\ .\ x$.
Zad. 22 *(2 pkt.)* Napisz term który zachowuje się jak koniunkcja.
Zad. 23 *(2 pkt.)* Napisz term który zachowuje się jak alternatywa.
Zad. 24 *(2 pkt.)* Napisz term który zachowuje się jak równoważność.

Swoją drogą, dawno nie było liczb naturalnych, nie? Takich z zerem i następnikiem (moje ulubione). Okazuje się że zero to wcale nie jest takie *nic*. Zero to też lambda term, wygląda tak:

$$\lambda\ f\ x\ .\ x$$

Następnik też nie jest wzięty z czapy. Oto on:

$$\lambda\ n\ .\ \lambda\ f\ x\ .\ f\ (n\ f\ x)$$

Ten to jest paskudny dopiero. Ale nie martw się, wyjaśniam:

Intuicja Liczba naturalna n to abstrakcja która zaaplikowana do innej abstrakcji *składa* ją z samą sobą n razy. Przykładowo, $2 = \lambda\ f\ x\ .\ f\ (f\ x)$. W szczególności zero, jak opisałem wyżej, zupełnie olewa ową abstrakcję — $0\ f\ x = x$ (czyli f jest zaaplikowane zero razy do x). Następnik natomiast to abstrakcja która, jeśli zaaplikowana do termu reprezentującego liczbę n, redukuje się do termu który reprezentuje liczbę o jeden większą (tj. składa swój argument n razy i jeszcze jeden raz).

A co ja będę się rozpisywał, zadanko:

Zad. 25 *(0.5 pkt.)* Zdefiniujmy sobie 3 jako $\lambda\ f\ x\ .\ f\ (f\ (f\ x))$. Zredukuj ile się da term $S\ 3$, gdzie S to nasz następnik. To co wyjdzie nazwiemy 4.

Powinno wyjść $\lambda\ f\ x\ .\ f\ (f\ (f\ (f\ x)))$. A skoro o zadankach mowa...

Zad. 26 *(3 pkt.)* Zdefiniuj term zachowujący się jak dodawnie. Powinien wyglądać tak: $\lambda\ n\ m\ .\ T$ i redukować się poprawnie gdy zaaplikowany do dowolnych termów reprezentujących liczby. Udowodnij że $2 + 2 \xrightarrow{*} 4$
Zad. 27 *(3 pkt.)* Zdefiniuj term zachowujący się jak mnożenie. Pokaż poprawność na jakimś śmiesznym przykładzie.
Zad. 28 *(3 pkt.)* Zdefiniuj term zachowujący się jak potęga (wykładnik może być przed lub po podstawie, bez znaczenia). Spróbuj zrobić go jak najkrótszego. On serio może być bardzo krótki.
Zad. 29 *(4 pkt.)* Zdefiniuj term który sprawdza czy jedna liczba jest większa od drugiej. Konkretnie, term G postaci $\lambda\ n\ m\ .\ T$ taki że $G\ x\ y$ redukuje się do prawdy gdy $x > y$, a w przeciwnym razie

do fałszu
Zad. 30 *(6 pkt.)* Zdefiniuj term który zachowuje się jak silnia. Good luck.

Nie ma znaczenia co twoje termy będą robić zaaplikowane do nie-liczb.

3.4 Nudzi ci się?

Nie musisz robić niczego z tej sekcji. Jedyne co chcę tu zaproponować to więcej lambd. No to go:

- Niech *pusta lista* to $E = \lambda\ c\ e\ .\ e$
- Niech *dodanie elementu* to $C = \lambda\ x\ l\ .\ \lambda\ c\ e\ .\ c\ x\ (l\ c\ e)$

Nieobowiązkowe zadanka / kminy:

Zad. 1 Zrozum co tu się dzieje
Zad. 2 Zdefiniuj term który zaaplikowany do termu reprezentującego listę redukuje się do termu który reprezentuje długość tej listy.
Zad. 3 Zdefiniuj sklejanie list (append, extend, concatenate; różnie zwą)
Zad. 4 Zdefiniuj odwracanie listy.
Zad. 5 Zdefiniuj odwracanie listy tak, żeby twój term miał postać $\lambda\ l\ .\ l\ T_0\ T_1\ T_2$.
Zad. 6 Zdefiniuj mapowanie listy (np $M\ f\ [a, b, c] \xrightarrow{*} [f\ a, f\ b, f\ c]$)
Zad. 7 Zdefiniuj filtrowanie listy — Ffl powinien zredukować się do listy o elementach z l, ale tylko takich (i wszystkich) że fe redukuje się do prawdy.
Zad. 8 Zdefiniuj zipowanie dwóch list (np $Z\ f\ [a, b, c]\ [x, y, z] \xrightarrow{*} [f\ a\ x, f\ b\ y, f\ c\ z]$)

Jak zareprezentować drzewa binarne? Jak zrobić DFSa? Albo BFSa? Albo w ogóle grafy? Czy da się opisywać liczby wymierne i rzeczywiste? Tyle pytań, a jeszcze więcej odpowiedzi[1]

[1] Czasem jedno pytanie ma wiele odpowiedzi, przy czym każde pytanie ma odpowiedź „nie wiem", więc odpowiedzi musi być więcej QED