INFORMATYKA OPARTA NA RACHUNKACH

Tomasz Drab
Wydział Matematyki i Informatyki, Uniwersytet Wrocławski
tdr@cs.uni.wroc.pl; ii.uni.wroc.pl/~tdr

Abstract. This paper discusses potential benefits of applying lambda calculus in computing science education, remarks how this could be done and summarises result of presenting this topic to over 2.700 students of various ages.

1. Wstęp

Coraz powszechniejsze jest zrozumienie, że dużo ważniejszym celem edukacji informatycznej jest nauka myślenia komputacyjnego niż nauka konkretnych technologii. Od lat nauczanie algorytmiki pozwala na osiąganie efektów kształcenia niezależnych od technologii. Jednak implementacja i testowanie rozwiązań odbywają się nadal w konkretnych środowiskach programistycznych, które, gdy pojawiają się problemy z ich obsługą, mogą odwodzić od istoty zagadnienia.

Kolejnym krokiem w kierunku nauczania myślenia komputacyjnego może być więc minimalizacja wykorzystania technologii na poziomie implementacji i testowania algorytmów. W tym celu można wykorzystać np. rachunek lambda — formalizm matematyczny będący funkcyjnym językiem programowania zupełnym w sensie Turinga. Rachunek ten pozwala na pisemne prowadzenie rachunków krok po kroku, czyli także testowanie programów, co uniezależnia cały proces programistyczny od użycia automatycznego komputera.

Mimo prostoty i ascetyczności rachunku lambda umożliwia on programowanie wysokopoziomowe konieczne przy tworzeniu programów o znaczeniu komercyjnym, w tym także programów opartych o metody sztucznej inteligencji. Struktury danych wyrażalne w tym języku umożliwiają także tworzenie modeli pojęciowych wspierających rozumowanie o takich algorytmach.

Celem niniejszej pracy jest prezentacja szans edukacyjnych wynikających z zastosowań rachunku lambda i tematów mu bliskich w procesie dydaktycznym. Większość propozycji oczekuje na wypróbowanie ich w większej skali. Na końcu za-

mieszczony został opis dotychczasowych osiągnięć związanych z nauczaniem rachunku lambda.

2. Uniwersalny język

Barendregt i Barendsen w swoim wprowadzeniu do rachunku lambda powołują się na marzenie Leibniza o stworzeniu uniwersalnego języka opisu problemów i procedury rozwiązywania problemów w nim wyrażonych [3]. Można powiedzieć, że w pewnym sensie, w stopniu, w jakim jest to możliwe, to **marzenie się spełniło**. Pojęcie algorytmu, czyli ściśle proceduralnego postępowania, zostało sformalizowane i przyjęte za sprawą maszyny Turinga, a rachunek lambda okazał się językiem pozwalającym wyrazić każde obliczenie, jakie ta maszyna może prowadzić.

W rachunku lambda występują jedynie trzy elementy składniowe, co **zawęża przestrzeń poszukiwań** możliwych rozwiązań podczas implementacji algorytmu. Jest to także punkt wyjścia do **zapanowania nad złożonością języka** zgodnie z programistyczną zasadą KISS – przez niewprowadzanie zbędnych komplikacji. Jest to ważne, ponieważ mimo prostoty języka, programowanie jest wystarczająco trudne.

Rachunek lambda udostępnia jeden, prostolinijny, mechanizm abstrakcji – definiowanie funkcji lambda. Dzięki temu wiele idei można **wyrazić wprost**, bez stosowania obejść w postaci dodatkowego kodowania itp. Pozwala to również na abstrakcję wspólnych części procedur zgodnie z zasadą DRY – „nie powtarzaj się".

Podstawowy rachunek lambda jest językiem nie zawierającym żadnych efektów obliczeniowych poza ukończeniem obliczenia. To znaczy, że w obliczeniach nie występują: wczytywanie i wypisywanie danych, zmiana stanu, przydział i zwalnianie pamięci, operacje na plikach, rzucanie wyjątków, losowość… To także zawęża przestrzeń poszukiwań rozwiązania problemu. Istnieje jednak możliwość **stopniowego rozszerzenia rachunku o operacje związane z efektami** [2], kiedy chce się je rozważać pozostając w ramach rachunków.

3. Śledzenie obliczeń

Uzasadnienie algorytmu Euklidesa opiera się na dwóch własnościach dotyczących największego wspólnego dzielnika: NWD(a, b) = NWD(b, a % b), gdy $b \neq 0$, oraz NWD(a, 0) = a. Dzięki tym równościom można również śledzić proces obliczania najmniejszego wspólnego dzielnika krok po kroku:

NWD(21, 35) = NWD(35, 21 % 35) = NWD(35, 21) = NWD(21, 35 % 21) =

= NWD(21, 14) = NWD(14, 21 % 14) = NWD(14, 7) = NWD(7, 14 % 7) =

= NWD(7, 0) = 7

Podobnie, mimo nieogonowej definicji rekurencyjnej, można śledzić obliczanie wartości funkcji Ackermanna-Péter:

A(1, 3) = A(0, A(1, 2)) = A(0, A(0, A(1, 1))) = A(0, A(0, A(0, A(1, 0)))) =

= A(0, A(0, A(0, A(0, 1)))) = A(0, A(0, A(0, A(0, 1)))) = A(0, A(0, A(0, 2))) =

= A(0, A(0, 3)) = A(0, 4) = 5

W oparciu o rachunek lambda można rozciągnąć tę **metodę śledzenia obliczeń na wszystkie algorytmy** operujące na dowolnych trwałych strukturach danych. Przykładem niech będzie, pomijająca wiele pośrednich kroków, prezentacja mechanizmu działania algorytmu sortowania przez scalanie dla list:

scalsort([1,3,4,5,2]) =

= podział([1,3,4,5,2])▷λ(*a*, *b*). scalenie(scalsort(*a*), scalsort(*b*)) =

= ([1,4,2], [3,5])▷λ(*a*, *b*). scalenie(scalsort(*a*), scalsort(*b*)) =

= scalenie(scalsort([1,4,2]), scalsort([3,5])) =

= scalenie(scalenie(scalsort([1,2]), scalsort([4])), scalsort([3,5])) =

= scalenie(scalenie([1,2], [4]), [3,5]) = scalenie([1,2,4], [3,5]) = [1,2,3,4,5]

Przykład działania funkcji „scalenie”:

scalenie([1,2,6,7,9], [3,4,5,8]) = 1::scalenie([2,6,7,9], [3,4,5,8]) =

= 1::2::scalenie([6,7,9], [3,4,5,8]) = 1::2::3::scalenie([6,7,9], [4,5,8]) =

= 1::2::3::4::scalenie([6,7,9], [5,8]) = 1::2::3::4::5::scalenie([6,7,9], [8]) =

= 1::2::3::4::5::6::scalenie([7,9], [8]) = 1::2::3::4::5::6::7::scalenie([9], [8]) =

= 1::2::3::4::5::6::7::8::scalenie([9], [ ]) = 1::2::3::4::5::6::7::8::9::scalenie([ ], [ ]) =

= 1::2::3::4::5::6::7::8::9::[ ] = [1,2,3,4,5,6,7,8,9]

Śledzenie obliczenia w języku imperatywnym w ogólności wymaga obserwacji stanu pamięci, stosu wywołań funkcji oraz bieżącej instrukcji w kodzie programu. Natomiast w podejściu funkcyjnym **stan obliczenia jest w pełni opisany jednym wyrażeniem**.

Ręczne wykonywanie obliczeń ułatwia również **zaangażowanie uwagi osoby** w śledzenie obliczenia krok po kroku, ponieważ wymaga od niej większej pracy niż przewijania następnych kroków w debuggerze. Podobną szansę w przypadku języków imperatywnych dawałoby symulowanie debuggera, jednak taka symulacja wydaje się zbyt trudna, by być praktyczną w edukacji szkolnej.

Śledzenie obliczenia krok po kroku pozwala **wniknąć w działanie algorytmu**, zaobserwować lub zaprezentować najbardziej istotne momenty lub tendencje obliczenia. Może to być np. zmniejszanie się w kolejnych krokach obliczenia argumentu funkcji rekurencyjnej. Prowadzenie uczniów do wykonywania takich obliczeń może być metodą dydaktyczną zgodą z ideami konstrukcjonizmu Paperta i *learning by doing*.

Obliczenia w rachunku lambda opierają się na prostych, mechanicznych zasadach. Odwoływanie się do wykonywania obliczeń w kontekście różnych tematów może prowadzić do **zidentyfikowania pracy czysto mechanicznej** i odróżnienia jej od pracy wymagającej twórczego wkładu, bez deprecjonowania tej pierwszej. W dobie upowszechniania się automatyzacji wielu procesów wydaje się być to cenną umiejętnością; także dla tych uczniów, których przyszła praca nie będzie wiązała się ściśle z informatyką, ale posiadanie tej umiejętności może przyczynić się do wdrożenia rozwiązań informatycznych w ich środowisku.

4. Model ontologiczny

Pojęcie liczby jest pojęciem wysoce abstrakcyjnym. Można przekonać się o tym zadając uczniom pytanie „Czym jest liczba trzy?”. Mimo tego wiele osób potrafi się nimi sprawnie posługiwać. Wynika to z tego, że działania odwołujące się do liczb nie wymagają rozważania ich natury. Właściwie **prowadzimy rachunki nie na liczbach**, a na liczebnikach, czyli ich reprezentacjach. To z kolei pozwala na mechanizację i uprzystępnienie obliczeń.

Podobnie abstrakcyjnym pojęciem jest pojęcie funkcji, które współcześnie sprawia takie trudności związane z ukazaniem jego istoty, że zostało wycofane z podstawy programowej dla szkół podstawowych [8]. Rachunek lambda może dać uczniom **silną i obliczeniową intuicję pojęcia funkcji** oraz **możliwość ich tekstowej reprezentacji**.

Pojęcie funkcji jest bardzo ważnym pojęciem matematycznym. Widać zatem, że w przypadku sukcesu wdrożenia rachunku lambda w dydaktyce rysuje się perspektywa współpracy między przedmiotami matematyka i informatyka na kolejnym polu, a tym samym postępu w nauczaniu obu dziedzin.

Udostępnienie uczniom tekstowej reprezentacji funkcji pozwala uchwycić ją jako jeden konkretny byt. To przygotowuje uczniów do swobodnego posługiwania się nimi np. przekazywania ich jako argumentów innych funkcji, czyli do wykorzystywania funkcji wyższego rzędu.

Rachunek lambda pozwala wyrażać trwałe struktury danych, czyli m.in. pary, warianty, listy, drzewa. Daje to podstawę do wykorzystywania ich tekstowej reprezentacji, gdyż znajdują one bezpośrednią interpretację jako wyrażenia rachunku lambda. Może być to szczególnie pomocne przy definiowaniu nowych abstrakcyjnych pojęć.

Z obserwacji autora, zdarza się, że przedstawianie uczniom liczb zespolonych natrafia na opór spowodowany brakiem interpretacji dla jednostki urojonej. Dzięki oswojeniu pojęcia pary, również operacyjnie, można przedstawić liczbę zespoloną jako parę liczb rzeczywistych bez konieczności wczesnego odwoływania się do układu współrzędnych. Po proceduralnym zdefiniowaniu działań na liczbach zespolonych można wykazać, że kwadrat jednostki urojonej jest, zgodnie z oczekiwaniem, zespoloną minus jedynką.

5. Systemy typów

Rachunek lambda pozwala na wyrażenie wielorakich danych: wartości logicznych, liczb, par, wariantów, list itp. a także funkcji przekształcających jedne dane w inne. Można się obawiać, że tak duża przestrzeń dostępnych obiektów może utrudnić zadanie łączenia ich w odpowiedni sposób. Jednak znacznym ułatwieniem może okazać się wprowadzenie typów porządkujących dane.

Dzięki niemu wprowadzając nowe pojęcie można zaczynać od określenia jego typu. Przykładowo binarne działania arytmetyczne można prezentować jako funkcje przyjmujące dwie liczby i zwracające jedną liczbę, funkcja wyliczająca wyrazy ciągu Fibonacciego jako przyjmującą liczbę naturalną i zwracającą liczbę naturalną, algorytmy sortowania jako funkcje przyjmujące listę liczb i zwracające (posortowaną) listę liczb, a konstruktor listy dla dowolnego typu $\mathbb{A}$ przyjmuje element typu $\mathbb{A}$ i listę elementów typu $\mathbb{A}$ i zwraca listę elementów typu $\mathbb{A}$ (oznaczenie strzałki „wiąże w prawo”):

$13 : \mathbb{N}$

$+ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$

$\mathrm{Fib} : \mathbb{N} \to \mathbb{N}$

[1, 3, 4, 5, 2] : Lista $\mathbb{N}$

scalsort : Lista $\mathbb{N} \rightarrow$ Lista $\mathbb{N}$

:: : $\mathbb{A} \rightarrow$ Lista $\mathbb{A} \rightarrow$ Lista $\mathbb{A}$

Określanie typów danych dotyczy także złożonych wyrażeń:

2 + 2 : $\mathbb{N}$

Fib(5) : $\mathbb{N}$

6::[1, 3, 4, 5, 2] : Lista $\mathbb{N}$

scalsort(6::[1, 3, 4, 5, 2]) : Lista $\mathbb{N}$

Fib(0) :: scalsort(6::[1, 3, 4, 5, 2]) : Lista $\mathbb{N}$

Rachunek lambda pozwala wyrażać konkretne listy i drzewa, system typów może zaś oferować formalizowanie pojęć listy i drzewa jako **rekurencyjnych typów danych**, co odpowiada definiowaniu własnych struktur danych. Oczywiście nie wymaga to mówienia o wskaźnikach, a nawet reprezentowaniu danych w pamięci komputera.

Posługiwanie się typami pozwala także na porcjowanie trudności zadań. Przykładowo rozważymy równanie z niewiadomą *x*:

scalsort([*x*, 1, 2]) = [1, 2, 3]

Pytanie „Czemu jest równy *x*?” można rozbić na pytania „Jakiego typu obiektem jest *x*?” oraz (po odpowiedzi) „Jaką liczbą naturalną jest *x*?”. Taka metoda uzupełniania luk może być wykorzystywana np. przy wspólnej implementacji algorytmów.

6. Programowanie wysokopoziomowe

Struktury opisane w poprzedniej sekcji pozwalają kodować także zbiory, macierze i bardziej złożone obiekty takie jak obrazy, dane dźwiękowe i multimedialne. Osadzenie ich w ramach rachunkowych i wystawienie na możliwość proceduralnego przetwarzania może być punktem wyjścia do analizowania ich metodami sztucznej inteligencji. Takie **odczarowanie złożonych danych** jest w pewnym stopniu konieczne, by postrzegać je jako nośniki informacji możliwej do wykorzystania za pomocą komputerów.

Operowanie na złożonych danych wspomaga programowanie wysokopoziomowe. Przykładowo możemy przypisać abstrakcyjne typy funkcjom składającym się na hipotetyczny system rekomendujący zakupy:

rozpoznawanie_kodu_kreskowego : obraz $\rightarrow$ identyfikator_produktu

odczyt_danych : baza_danych $\rightarrow$ identyfikator_produktu $\rightarrow$ dane_produktu

ocena_przydatności : preferencje_użytkownika $\rightarrow$ dane_produktu $\rightarrow$ ocena

komunikat_zwrotny : ocena $\rightarrow$ dokument_tekstowy

W projektowaniu rozwiązań biznesowych, również tych wykorzystujących sztuczną inteligencję, nie są istotne niskopoziomowe szczegóły implementacji. Dlatego możliwość programowania wysokopoziomowego sytuuje rachunek lambda w łączności z tym obszarem.

Dodatkowym potwierdzeniem związku rachunku lambda ze sztuczną inteligencją niech będzie także fakt, że John McCarthy, który ukuł termin „sztuczna inteligencja", jest także twórcą Lispu – wysokopoziomowego języka opartego na rachunku lambda, wykorzystywanego później w badaniach nad nią [4].

7. Podstawa dla logiki

Daleko posuniętym zastosowaniem rachunku lambda mogłoby być używanie go jako podstawy dla logiki matematycznej. Jest to możliwe dzięki związkowi znanemu jako izomorfizm Curry'ego-Howarda.

Izomorfizm ten wiąże dowody twierdzeń z programami. Przykładowo dowód twierdzenia o nieskończonej liczbie liczb pierwszych odpowiada programowi znajdującemu nową liczbę pierwszą. W związku z tym prowadzenie rozumowania można zinterpretować jako projektowanie konkretnego proceduralnego postępowania. Może być to szansa na zmniejszenie abstrakcyjności logiki i uprzystępnienie jej uczniom.

Rachunek lambda, poprzez izomorfizm, pozwala na tworzenie dowodów w logice konstruktywnej. Skoro dowodowi odpowiada pewne proceduralne działanie, które po przyswojeniu takiego przez ucznia staje się umiejętnością, mogłoby to pozwalać na wyeksponowanie zjawiska nabudowywania nowych umiejętności na starych. Byłby to przykład zgodności między konstruktywizmem logicznym a konstrukcjonizmem.

Na izomorfizmie Curry'ego-Howarda opierają się interaktywne programy asystujące przy dowodzeniu twierdzeń takie jak Coq [1]. Takie programy mogą na bieżąco

sprawdzać poprawność konstruowanego dowodu, co, dzięki odciążeniu nauczycieli, mogłoby pozwolić na poświęcenie większej ilości czasu na samą naukę rozumowania.

Zwiększenie dostępności treści, które łatwo można samodzielnie zweryfikować, mogłoby być dodatkową zachętą do samokształcenia.

8. Uniezależnienie od komputerów

Nauczanie oparte na rachunkach umożliwia ćwiczenie implementacji i testowania algorytmów bez potrzeby używania komputera. Uczniowie i nauczyciele są zapoznani z pracą tego rodzaju przez np. upraszczanie wyrażeń arytmetycznych.

Prowadzenie większej części zajęć informatycznych bez komputerów umożliwiałoby zwiększenie liczby godzin przeznaczonych na naukę informatyki bez konieczności zajmowania pracowni komputerowej.

Nieodróżnianie informatyki od technologii informacyjno-komunikacyjnych jest wymieniane jako jeden z powodów regresu nauczania informatyki [7]. Zaznajomienie się z ogromem praktycznych umiejętności informatycznych możliwych do wykształcenia bez użycia komputera powinno przyczynić się do zwiększenia świadomości na temat użyteczności i samodzielności informatyki jako dyscypliny.

Komputer bywa używany przez uczniów jako rozrywka przez co udostępnienie go może prowadzić do niepotrzebnych rozproszeń. Natomiast zadanie sformułowania algorytmu „na kartce” zachęca do przemyślenia idei algorytmu, a chroni przed pokusą modyfikowania kodu „na chybił trafił” w celu uzyskania lepszych wyników.

9. Praca przy komputerze

Niezależność od komputera nie oznacza zaprzestania posługiwania się nim. Potrzeba więc języka programowania odpowiedniego do wyrażania algorytmów sformułowanych w rachunku lambda. Naturalnym kandydatem wydawałyby się Lisp lub któryś z jego dialektów, jednak posiadają one dość trudną składnię. Język Haskell posiada składnię bardzo bliską językowi matematyki, ale programowanie imperatywne w tym języku wymaga zaawansowanej jego znajomości.

Proponowanym przez autora językiem do pracy przy komputerze, w tym kontekście, jest Python 3. Należy jednak zwrócić uwagę, że lista pythonowa nie jest tym samym, co lista lispowa, a operowanie na nich za pomocą pojęć głowy i ogona listy prowadziłoby do niewydajnych rozwiązań. Język Python nie wykonuje także optymalizacji rekurencji ogonowej, co w pewnych przypadkach może prowadzić do

przepełnień stosu wywołań. Niemniej jednak składnia dla par i pythonowych list (tablicolist) ułatwia implementację funkcji, o których można rozumować rachunkowo. Ponadto implementację można wzbogacić o statyczne deklaracje typów i sprawdzanie ich poprawności za pomocą systemu mypy [6].

Dla przykładu pokażemy wymagane własności funkcji wykorzystanych do implementacji sortowania przez scalanie, a następnie ich implementację w języku Python 3 wraz z deklaracjami typów.

podział([ ]) = ([ ], [ ])

podział([*x*]) = ([*x*], [ ])

podział(*y*::*z*::*xs*) = podział(*xs*)▷λ(*ys*, *zs*). (*y*::*ys*, *z*::*zs*)

scalenie([ ], *ys*) = *ys*

scalenie(*xs*, [ ]) = *xs*

scalenie(*x*::*xs*, *y*::*ys*) = jeśli *x* < *y* to *x*::scalenie(*xs*, *y*::*ys*)
inaczej y::scalenie(*x*::*xs*, *ys*)

scalsort([ ]) = [ ]

scalsort([*x*]) = [*x*]

scalsort(*xs*) = podział(*xs*)▷λ(*a*, *b*). scalenie(scalsort(*a*), scalsort(*b*)),
gdy *xs* ma co najmniej 2 elementy

```
from typing import Tuple, List

def podzial(xs: List[int]) -> Tuple[List[int], List[int]]:
    return (xs[::2], xs[1::2])

def scalenie(xs: List[int], ys: List[int]) -> List[int]:
    i: int = 0
    j: int = 0
    wynik: List[int] = []
    while i < len(xs) and j < len(ys):
        if xs[i] < ys[j]:
            wynik.append(xs[i])
            i += 1
        else:
            wynik.append(ys[j])
            j += 1
    wynik += xs[i:]
```

```
    wynik += ys[j:]
    return wynik

def scalsort(xs: List[int]) -> List[int]:
    if len(xs) < 2:
        return xs
    a, b = podzial(xs)
    return scalenie(scalsort(a), scalsort(b))
```

Przedstawiona implementacja w języku Python nie powinna powodować przepełnienia się stosu wywołań, ponieważ jego wysokość jest proporcjonalna do logarytmu liczby elementów sortowanej listy. Nie uchwyca ona jednak prostoty, z jaką zdefiniowana jest rachunkowo funkcja „scalenie".

10. Rachunek lambda na eliminacjach konkursu KOMA

Konkurs matematyczny KOMA organizowany jest przez Instytut Matematyczny Uniwersytetu Wrocławskiego od 2005 roku [5]. Eliminacje przeprowadzane są w szkołach, których uczniowie uczą się na co dzień. Nauczyciele w ciągu **jednej godziny lekcyjnej** wykładają uczniom temat wyznaczony przez organizatorów. Tego samego dnia lub niedługo po dniu wykładu uczniowie rozwiązują zadania z tego tematu.

W roku 2018 tematem eliminacji XIV edycji konkursu był rachunek lambda. Obawą organizatorów było to, że rachunek lambda może okazać się tematem zbyt abstrakcyjnym, by został dobrze przyswojony przez uczestników. Jednak temat okazał się dobrze opanowany zarówno przez nauczycieli matematyki, jak i uczniów, o czym świadczą wysokie progi kwalifikacji do finału, jakie musiały zostać ustalone na postawie wyników.

kategoria wiekowa	liczba uczestników eliminacji	procent uczestników zakwalifikowanych do finału	procentowy próg kwalifikacji do finału
klasy 4-6 szkół podstawowych	829	16,6%.	84%
klasy 7-8 szkół	1140	14,6%	91%

podstawowych i gimnazjum			
szkoły średnie	787	19,3%	78%

Tabela 1 Podsumowanie wyników eliminacji XIV edycji konkursu KOMA

Materiały przekazane nauczycielom dla każdej kategorii wiekowej oraz bardziej szczegółowe dane dostępne są na stronie konkursu.

11. Podsumowanie

Uzyskane na konkursie KOMA wyniki dają podstawy do przypuszczeń, że uczniowie i nauczyciele byliby gotowi, by rachunek lambda był nauczany w szkołach, a jego użyteczność była wykorzystana w procesie edukacyjnym.

Do tej pory nie wykonano jednak eksperymentów sprawdzających, czy rzeczywiście stosowanie tego narzędzia ułatwiłoby naukę informatyki. Otwarte również są kwestie, który moment w edukacji informatycznej byłby odpowiedni na wdrożenie takiego podejścia oraz jak bardzo miałoby się ono przejawiać w programie nauczania. Warto także rozważyć, jakie rozwiązania mogłyby zbliżyć sposób wyrażania algorytmów w ramach rachunkowych i w języku wykonywanym przez komputer.

12. Podziękowania

Chciałbym podziękować pani Małgorzacie Mikołajczyk za udostępnienie „infrastruktury” konkursu KOMA, które pozwoliło na wypróbowanie prezentacji rachunku lambda w większej skali, oraz za pomoc w przygotowaniu formy rachunku odpowiedniej dla uczniów.

Literatura

1. The Coq Development Team, *The Coq proof assistant*, https://coq.inria.fr, ostatni dostęp 14.05.2019 roku.
2. M. Pretnar, *An Introduction to Algebraic Effects and Handlers*, 2015.
3. H. Barendregt, E. Barendsen, *Introduction to Lambda Calculus*, 1994.
4. Kyoto Prize, https://www.kyotoprize.org/en/laureates/john_mccarthy, ostatni dostęp 14.05.2019 roku.

5. Konkurs Matematyczny KOMA, http://math.uni.wroc.pl/fmw/koma/konkurs-matematyczny-koma/, ostatni dostęp 14.05.2019 roku.
6. Mypy team, *mypy*, http://mypy-lang.org, ostatni dostęp 14.05.2019 roku.
7. Maciej M. Sysło, *Myślenie komputacyjne. Informatyka dla wszystkich uczniów*, 2011.