WWW21 – Rachunek λ – zadania kwalifikacyjne

Michał Horodecki

April 2025

Wstęp

Forma rozwiązań dowolna, byle czytelna.
Nie trzeba robić wszystkiego, ale zachęcam żeby zrobić jak najwięcej, częściowe rozwiązania i pomysły też są OK ;)
W razie wątpliwości lub jakichkolwiek pytań śmiało piszcie na discordzie lub mailu (kontakt na stronie warsztatów).

Same zadania zaś, jak zwykle proszę wysyłać przez stronę warsztatów.

1 Programowanie [20pkt]

Ta sekcja ma przybliżyć Wam nieco sposób w jaki na warsztatach będziemy korzystać z rachunku lambda od strony programistycznej. W tym celu do napisania jest parę prostych programów, z pewnymi ograniczeniami.

Zasady:

- Język programowania dowolny, ale dobrze jakby miał sensowne wsparcie dla funkcji (np. Python, JavaScript, Haskell)
- Nie wolno korzystać z jakichkolwiek pętli (`while`, `for`, `foreach`, `loop`, `goto`, itp.) – w ramach alternatywy polecam rekurencję
- Nie wolno zmieniać raz przypisanych wartości (tj. każda zmienna powinna być `const` lub `final`)
- Nie wolno korzystać z funkcji wbudowanych w język/bibliotekę standardową, które w istotny sposób rozwiązują zadanie
- Korzystanie z wbudowanych operatorów i struktur danych jest ok jeśli nie są używane do modyfikacji

1.1 Liczby Fibbonaciego [2pkt]

Napisz funkcję `fib(n)`, która na wyjściu zwraca n-tą liczbę Fibbonaciego $F_0 = 0, F_1 = 1, \ldots$ modulo 1000

Za napisanie wersji która oblicza `fib(10)` można dostać 1pkt zaś za wersję obliczającą `fib(100)` można dostać 2pkt.

1.2 Liczby pierwsze [3pkt]

Napisz funkcję `nthprime(n)` która zwraca n-tą liczbę pierwszą $p_0 = 2, p_1 = 3, \ldots$

1.3 Iteracja funkcji [2pkt]

Napisz funkcję `iterate(f, x, n)` która oblicza $\underbrace{f(f \ldots f(f}_{n \text{ razy}}(x)) \ldots)$ lub zwraca x gdy $n = 0$.

Przykładowo, w Pythonie, przy definicji `def inc(x): return x + 1` wywołanie `iterate(inc, 2, 3)` powinno zwrócić 5.

1.4 Listy

Na potrzeby kolejnych zadań zamodelujemy listy za pomocą par/obiektów:

- Pustą listę modelujemy wyróżnioną wartością pokroju `null` lub `None`
- Singleton `[x]` będzie parą `(x, null)`, alternatywnie obiektem `{head: x, tail: null}`
- Mając listę L element dokładamy **na początek**, tworząc listę `[x, ...L]` poprzez stworzenie pary `(x, L)` (lub odpowiedniego obiektu).

  Lista n-elementowa $[x_1, \ldots, x_n]$ będzie zatem n-krotnie zagnieżdżoną parą:

  $$(x_1, (x_2, \ldots (x_n, \texttt{null}) \ldots))$$

  Czyli przykładowo w Pythonie listę $[1, 2, 3]$ można zapisać jako `(1, (2, (3, None)))`

1.4.1 Znajdywanie elementu [2pkt]

Napisz funkcję `elem(x, L)` która sprawdza czy lista L zawiera element x

1.4.2 Sumowanie listy [2pkt]

Napisz funkcję `sum(L)` która sumuje listę liczb L lub zwraca 0 jeśli lista jest pusta

1.4.3 Filtrowanie listy [3pkt]

Napisz funkcję `even(L)` która zwraca listę L' złożoną jedynie z parzystych liczb z L, w tej samej kolejności.

1.4.4 Uogólnienie sumowania [3pkt]

Napisz funkcję `fold(f, z, L)`, która:

- dla pustej listy zwraca z
- dla listy $[x_1, \ldots, x_n]$ zwraca $f(x_1, f(x_2, \ldots f(x_n, z) \ldots))$.

Bonusowe pytanie: Równie dobrze można by obliczać $f(\ldots f(f(z, x_1), x_2) \ldots, x_n)$.

1. Czy ma to znaczenie gdy chcemy policzyć sumę liczb w L?
2. Podaj przykład gdy te dwa podejścia zwrócą różne wyniki.

1.4.5 Uogólnienie filtrowania [3pkt]

Napisz funkcję `filter(f, L)` która zwraca listę L' złożoną jedynie z tych elementów L dla których funkcja f zwraca prawdę.

2 Matematyka [20pkt]

Z drugiej strony, rachunek lambda jest pewnym matematycznym tworem, do którego analizy przyda nam się zrozumienie dwóch konceptów.

2.1 Funkcje pierwotnie rekurencyjne [10pkt]

Powiemy, że funkcja $f : \mathbb{N}^k \to \mathbb{N}$ jest pierwotnie rekurencyjna jeśli jest skonstruowana w jeden z poniższych sposobów:

1. $f(x_1, \ldots, x_k) = C_k^c(x_1, \ldots, x_k) = c$ gdzie $c \in \mathbb{N}$ jest dowolną stałą
2. $f(x_1, \ldots, x_k) = P_k^i(x_1, \ldots, x_k) = x_i$
3. $f(x) = S(x) = x + 1$
4. Istnieją pierwotnie rekurencyjne funkcje $g, h_1, \ldots, h_n$, takie że
$$f(x_1, \ldots, x_k) = g(h_1(x_1, \ldots, x_k), \ldots, h_n(x_1, \ldots, x_k))$$
5. Istnieją pierwotnie rekurencyjne funkcje g, h takie że
$$\begin{cases} f(0, x_1, \ldots, x_{k-1}) = g(x_1, \ldots, x_{k-1}) \\ f(n, x_1, \ldots, x_{k-1}) = h(n, x_1, \ldots, x_n, f(n-1, x_1, \ldots, x_{k-1})) \end{cases}$$

2.1.1 Konstrukcja funkcji [8pkt]

Wykaż (przez konstrukcję), że poniższe funkcje są pierwotnie rekurencyjne:

1. $f_1(a) = a + c$, gdzie c jest stałą
2. $f_2(a, b) = a + b$
3. $f_3(a, b) = a \cdot b$
4. $f_4(a, b) = \max(a - b, 0)$
5. $f_5(a, b) = \begin{cases} 1 & \text{gdy } a = b \\ 0 & \text{wpp} \end{cases}$
6. $f_6(a, b) = \begin{cases} 1 & \text{gdy } a < b \\ 0 & \text{wpp} \end{cases}$
7. $f_7(a, b) = \begin{cases} 0 & \text{gdy } a = 0 \vee b = 0 \\ 1 & \text{wpp} \end{cases}$
8. $f_8(n) = \begin{cases} 0 & \text{gdy } n = 0 \\ 1 & \text{gdy } n = 1 \\ f_8(n-1) + f_8(n-2) & \text{wpp} \end{cases}$

   Hint: Załóż że istnieją pierwotnie rekurencyjne funkcje pary P, L, R, takie że $\begin{cases} L(P(a, b)) = a \\ R(P(a, b)) = b \end{cases}$

2.1.2 Funkcje które nie są pierwotnie rekurencyjne [2pkt]

Czy każda funkcja $f : \mathbb{N} \to \mathbb{N}$ jest pierwotnie rekurencyjna? Odpowiedź uzasadnij.

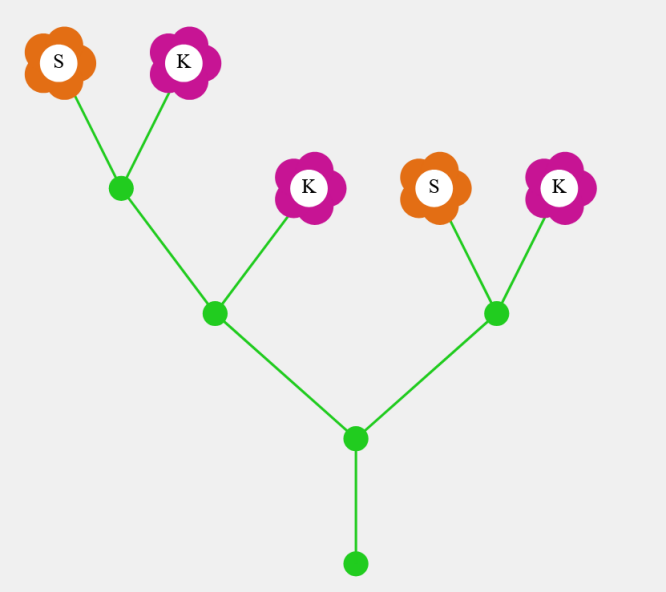
<hint>
</hint>

2.2 Rachunek SK – Bajtocjańskie krzewy [10pkt]

Ta sekcja może wydawać się trochę długa i abstrakcyjna, ale nie ma co się zrażać, jest tu dużo słów i obrazków na opisanie w miarę prostego systemu, a w razie czego służę pomocą.

Interaktywna rysowajka do tego zadania znajduje się pod linkiem `https://mhorod.github.io/sk-vis/`

W Bajtocji rośnie bardzo ciekawy gatunek krzewu. Każdy krzew jest drzewem binarnym (tj. z każdego miejsca wyrastają w górę dokładnie dwie gałęzie), na którego szczycie znajdują się dwa rodzaje kwiatów – będziemy nazywać je S i K.



Rysunek 1: Przykład Bajtocjańskiego krzewu

Ze względu na bardzo regularną strukturę, każdy krzew można zapisać za pomocą słów złożonych z liter S, K, oraz poprawnie użytych nawiasów:

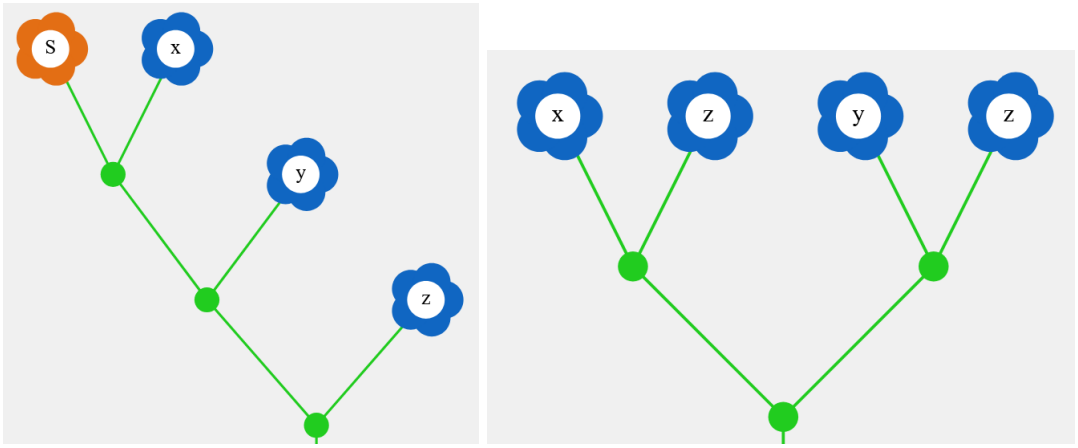
- Pojedyncza litera K lub S oznacza kwiat danego rodzaju
- Każdy nawias zawiera dokładnie dwa elementy, reprezentujące lewą i prawą gałąź, z których każdy może być K lub S lub kolejnym nawiasem

Krzew z powyższego rysunku można opisać napisem $(((SK)K)(SK))$

Krzew ten rośnie w dość specyficzny sposób – każdego roku wybiera dokładnie jedno swoje miejsce z którego wyrasta jeden z dwóch rodzajów fragmentów i zamienia je z ustalonym poniżej wzorem. W miejscu x, y, z może znajdować się dowolny kwiat, lub większy fragment krzewu. Jeśli takiego miejsca nie ma, to krzew już więcej się nie zmienia i mówimy że jest w postaci dojrzałej.

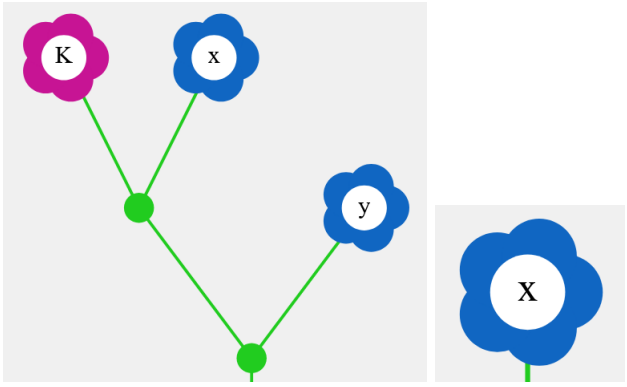
- Fragment S zmienia się jak poniżej

$$(((Sx)y)z) \to ((xz)(yz))$$



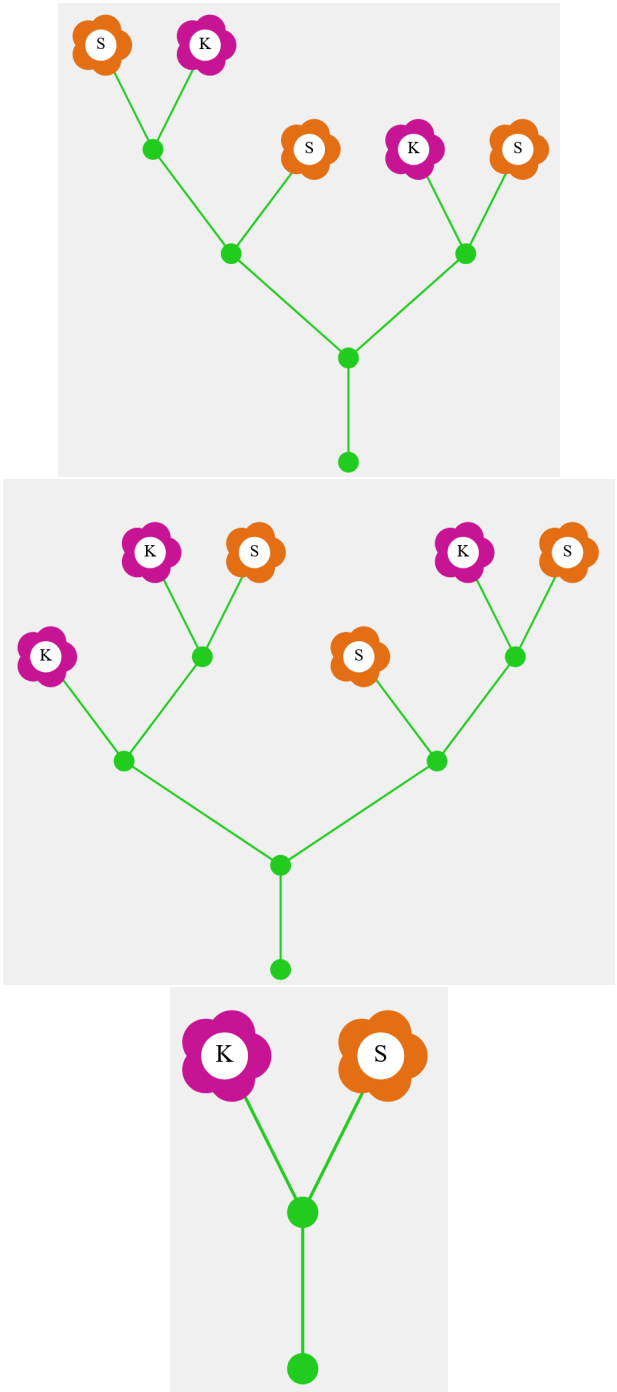
- Fragment K zmienia się jak poniżej

$$((Kx)y) \to x$$



Przykładowa ewolucja krzewu:

$$(((SK)S)(KS)) \to ((K(KS))(S(KS))) \to (KS)$$



2.2.1 Zabawa krzewami [2pkt]

Wejdź na stronę `https://mhorod.github.io/sk-vis/` i poeksperymentuj z różnymi konfiguracjami, tak żeby zapoznać się z tym systemem. W ramach rozwiązania prześlij jakiś nietrywialny krzew uzyskany w wyniku zabawy.

2.2.2 Identyczność [2pkt]

Znajdź taki fragment krzewu, nazwijmy go `I` jak identyczność, taki, że dla dowolnego fragmentu x zachodzi:

$$(Ix) \to x$$

2.2.3 Zwracanie drugiego argumentu [3pkt]

Krzew K zwraca nam pierwszy argument jaki do niego podamy:

$$((Kx)y) \to x$$

Skonstruuj taki fragment k, który zwraca drugi argument, tj.:

$$((kx)y) \to y$$

2.2.4 Dojrzałość [3pkt]

Czy każdy krzew ma postać dojrzałą? Uzasadnij lub wskaż krzew który rozrasta się w nieskończoność.

<hint> </hint>