Spis treści

1 Ćwiczenia 1 (K1) - 24.02.2020 - logika i indukcja matematyczna **5**
1.1 Informacje o przedmiocie . . . 5
1.2 Logika zdań . . . 5
1.2.1 Logika rzędu 0 . . . 5
1.2.2 Logika rzędu 1 . . . 7
1.2.3 Zadania . . . 8
1.3 Indukcja matematyczna . . . 9
1.3.1 Przykład . . . 9
1.4 Zadania do domu . . . 9

2 Ćwiczenia 2 (K1) - 02.03.2020 - podstawy funkcji prymitywnie rekurencyjnych **11**
2.1 Definiowanie zbiorów liczbowych przez rekurencje . . . 11
2.2 Definiowanie języka przez rekurencje . . . 12
2.3 Definiowanie funkcji przez rekurencje . . . 12
2.4 Definiowanie obliczalności przez rekurencje . . . 12
2.5 Zadania do domu . . . 16

3 Ćwiczenia 3 (K1) - 09.03.2020 - więcej funkcji p.rek. + relacje **25**
3.1 Poszerzamy klasę funkcji prymitywnie rekurencyjnych . . . 25
3.2 Przykład . . . 26
3.3 Relacje i funkcje charakterystyczne . . . 26
3.3.1 Własności relacji prymitywnie rekurencyjnych . . . 27
3.4 Funkcje zadane w przedziałach - jak pokazać, że są p. rek. ? . . . 29
3.5 Relacje z kwantyfikatorami . . . 29
3.6 Zadania do domu . . . 30

4 Ćwiczenia odwołane 16.03.2020 **33**

5 Ćwiczenia 5 (K1) - 23.03.2020 - funkcje częściowo rekurencyjne **34**
5.1 Funkcja Akermana . . . 34
5.2 Funkcja całkowita vs częściowa . . . 34
5.3 Operator μ - minimalizacja (przeszukiwanie) . . . 34
5.3.1 Działanie operatora μ . . . 34
5.4 Prosty przykład . . . 35
5.5 Bardziej praktyczne przykłady + c.d. zadań domowych . . . 36
5.5.1 Dzielenie całkowite . . . 36
5.5.2 Pierwiastek z sufitem lub podłogą . . . 37
5.5.3 Najmniejsza wspólna wielokrotność NWW . . . 37
5.5.4 Największy wspólny dzielnik NWD . . . 38
5.6 Zadania do domu TODO . . . 38

6 Ćwiczenia 6 (K2) - 30.03.2020 - rachunek lambda, składnia, zmienne wolne i związane, konwencje, redukcje **39**
6.1 Składnia rachunku lambda . . . 39
6.2 Konwencje stosowane w rachunku lambda . . . 39
6.2.1 Konwencja 1 . . . 39
6.2.2 Konwencja 2 . . . 40
6.2.3 Konwencja 3 . . . 40

6.3 Ewaluacja aplikacji funkcji . . . 40
6.3.1 Ewaluacja jako drzewo syntaktyczne . . . 40
6.3.2 Problem z ewaluacją przez naiwne podstawianie . . . 41
6.4 Zmienne wolne i związane . . . 42
6.4.1 Zmienne wolne . . . 43
6.4.2 Zmienna związane . . . 43
6.4.3 Wolna i związana ?! . . . 43
6.5 α-konwersja . . . 44
6.6 β-redukcja . . . 44
6.7 η-redukcja . . . 45
6.8 Rozwiązanie problemu *capture* . . . 45
6.9 Zadania do domu . . . 45

7 Ćwiczenia 7 (K2) - 06.04.2020 - r. lambda, redukcje c.d., funkcja if-then-else 47
7.1 Postacie termów . . . 47
7.2 Zastosowanie rachunku lambda do zdań logicznych . . . 47
7.2.1 if-then-else . . . 48
7.3 Zadania do domu . . . 48

8 Ćwiczenia 8 (K2) - 20.04.2020 - r. lambda, logika, liczby, wyrażenia arytmetyczne 49
8.1 Definiowanie logiki przez rachunek lambda c.d. . . . 49
8.1.1 NOT . . . 49
8.1.2 AND . . . 49
8.1.3 OR . . . 50
8.2 Zadania do domu - Część I . . . 50
8.3 Definiowanie liczb przy pomocy rachunku lambda . . . 51
8.4 Następnik i przykłady . . . 51
8.5 Operacje arytmetyczne . . . 52
8.5.1 Dodawanie . . . 52
8.6 Zadania do domu - Część II . . . 53
8.6.1 Mnożenie . . . 53
8.6.2 Potęgowanie . . . 53

9 Ćwiczenia 9 (K2) - 27.04.2020 54
9.1 Kolokwium I . . . 54
9.2 c.d. odejmowanie w wyrażeniach lambda TODO . . . 54
9.3 Schematy blokowe . . . 54
9.4 Schematy blokowe jako model obliczalności . . . 56
9.5 Składanie funkcji . . . 56
9.6 Prymitywna rekurencja . . . 57
9.7 Programowanie przy użyciu WHILE TODO . . . 58
9.8 Zadania do domu cz. I TODO . . . 58

10 Ćwiczenia 10 (K2/K3) - 04.05.2020 - WHILE c.d. + wstęp do maszyn Turinga 59
10.1 Programowanie WHILE c.d. (K2) . . . 59
10.1.1 Przykład . . . 59
10.2 Zadania do domu cz. I TODO (K2) . . . 60
10.3 Maszyna Turinga (K3) . . . 61
10.3.1 Formalna definicja . . . 62
10.3.2 Przykłady . . . 62
10.4 Zadania do domu cz. II (K3) . . . 66

11 Ćwiczenia 11 (K3) - 11.05.2020 - Rozwiązywanie zadań z Turinga **67**
11.1 Rozwiązywanie zadań 67
11.2 Zadania do domu 73

12 Ćwiczenia 12 (K3) - 18.05.2020 - Rozstrzygalność, decydowalność, języki rekurencyjne i rekurencyjnie przeliczalne, wstęp do problemu stopu **75**
12.1 Problemy i rozstrzygalność 75
12.1.1 Problem 75
12.1.2 Rozstrzygalność 75
12.2 Problemy decyzyjne i decydowalność 76
12.3 Języki i decydowalność 76
12.3.1 Język rekurencyjny 76
12.3.2 Język rekurencyjnie przeliczalny 77
12.3.3 Język niedecydowalny 77
12.4 Problem stopu - wstęp 79
12.5 Zadania do domu TODO 79

13 Ćwiczenia 13 (K3) - 25.05.2020 - problem stopu i uniwersalna maszyna Turinga **80**
13.1 Problem stopu - c.d. 80
13.1.1 Symbole wejściowe dla uniwersalnej maszyny Turinga 80
13.2 Problem stopu i dowód 81
13.2.1 Dowód 81
13.3 Zadanie domowe 81

14 Ćwiczenia 14 (K3) - 01.06.2020 - Redukowalność, maszyny ATM i ETM **82**
14.1 Redukowalność 82

15 Ćwiczenia 15 (K3) - 08.06.2020 - Twierdzenie Rice'a **84**
15.1 Twierdzenie Rice'a 84
15.2 Własności trywialne i nietrywialne 84
15.3 Przykłady 85

16 Ściągawka z maszyn Turinga **86**

Na podstawie:

- Ćwiczenia audytoryjne - teoria obliczeń
- `https://www.youtube.com/watch?v=bFkU-qV2Ioo` - filmiki 1 do 5

1 Ćwiczenia 1 (K1) - 24.02.2020 - logika i indukcja matematyczna

1.1. Informacje o przedmiocie

- max. do zdobycia 100 punktów
- zaliczenie od 50 punktów
- 100 punktów = 3 kolokwia po 25 + 25 punktów za aktywność na zajęciach
- dopuszczalne dwie nieobecności nieusprawiedliwione

1.2. Logika zdań

1.2.1. Logika rzędu 0

Podstawowe operatory:

symbol logiczny	spójnik	nazwa zdania złożonego
$\wedge$	i	koniunkcja
$\vee$	lub	alternatywa
$\neg$, ~	nieprawda, że...	negacja (zaprzeczenie)
$\Rightarrow$	jeżeli..., to...	implikacja
$\Leftrightarrow$	wtedy i tylko wtedy, gdy...	równoważność

Działanie operatorów:

p	q	p ∧ q
0	0	0
0	1	0
1	0	0
1	1	1

p	q	p ∨ q
0	0	0
0	1	1
1	0	1
1	1	1

p	¬ p
0	1
1	0

p	q	p ⇒ q
0	0	1
0	1	1
1	0	0
1	1	1

p	q	p ⇔ q
0	0	1
0	1	0
1	0	0
1	1	1

Pytanie

Jakie znasz podstawowe operatory logikii zerowego rzędu, jak działają ?

Podstawowe prawa logiki zerowego rzędu:

Przemienność alternatywy:
$p \vee q \Leftrightarrow q \vee p$

Łączność alternatywy:
$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$

Przemienność koniunkcji:
$p \wedge q \Leftrightarrow q \wedge p$

Łączność koniunkcji:
$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$

Rozdzielność koniunkcji względem alternatywy:
$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$

Rozdzielność alternatywy względem koniunkcji:
$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$

Pierwsze prawo De Morgana:
$\sim(p \vee q) \Leftrightarrow \sim p \wedge \sim q$ - zaprzeczeniem alternatywy jest koniunkcja zaprzeczeń

Drugie prawo De Morgana:
$\sim(p \wedge q) \Leftrightarrow \sim p \vee \sim q$ - zaprzeczeniem koniunkcji jest alternatywa zaprzeczeń

Zaprzeczenie implikacji:
$\sim(p \Rightarrow q) \Leftrightarrow p \wedge \sim q$

Zastąpienie równoważności implikacją:
$(p \Leftrightarrow q) \Leftrightarrow [(p \Rightarrow q) \wedge (q \Rightarrow p)]$

Pytanie

Podaj pierwsze i drugie prawo De Morgana (dotyczą negacji koniunkcji i alternatywy).

Pytanie

Jak możemy inaczej zapisać $p \implies q$?

1.2.2. Logika rzędu 1

W logice pierwszego rzędu do rozważań dodajemy kwantyfikatory. Nasze formuły są postaci:

$$\forall f.logiczna$$
$$\exists f.logiczna$$

Dwa ważne prawa logiczne w tej logice to:

$$\neg\forall x p(x) \equiv \exists x \neg p(x)$$
$$\neg\exists x p(x) \equiv \forall x \neg p(x)$$

Pytanie

Jak inaczej możemy zapisać $\neg\forall x p(x)$? A jak możemy zapisać $\neg\exists x p(x)$?

Przypominamy znaczenie symboli oznaczających zbiory liczb:

- N - naturalne (także 0)
- Z - całkowite
- Q - wymierne
- R - rzeczywiste
- C - zespolone

1.2.3. Zadania

Pytanie

Określić czy poniższe zdania są prawdziwe w zbiorach N, Z, Q, R:

$$1)\ \forall x\ x \geq 0$$
$$2)\ \forall x\ x^2 = 2 \implies x = 5$$
$$3)\ \forall x \exists y\ (x + y = 0)$$
$$4)\ \forall x \exists y\ x > y$$
$$5)\ \exists x \forall y\ x > y$$

1.
   - N - Tak, bo każda naturalna jest większa lub równa 0
   - Z - Nie
   - Q - Nie
   - R - Nie
2.
   - N, Z, Q - Tak, bo nie istnieje taki x, że $x^2 = 2$
   - R - Nie
3.
   - N - nie
   - Z, Q, R - tak
4.
   - N - nie
   - Z, Q, R - tak
5. Aby ułatwić sobie sprawę negujemy formułę $\neg(\exists x \forall y\ x > y) \equiv \forall x \exists y\ x \leq y$, jeśli negacja jest prawdziwa, to formuła wyjściowa jest fałszywa, jeśli negacja jest fałszywa, to formuła wyjściowa jest prawdziwa
   - Negacja dla N, Z, Q, R jest prawdziwa więc formuła wyjściowa jest fałszywa dla N, Z, Q, R

1.3. Indukcja matematyczna

To metoda stosowana w dowodzeniu twierdzeń, zazwyczaj w zbiorze liczb naturalnych. Dowód składa się z trzech etapów:

1. $S \subset N$
2. Sprawdzenie poprawności twierdzenia dla najmniejszej liczby ze zbioru S
3. Założenie tezy dla pewnego n: $T(n)$, postawienie tezy dla $T(n+1)$ i pokazanie, że $T(n) \implies T(n+1)$

1.3.1. Przykład

Pytanie

$$p(n) = 1 + 3 + 5 + ... + 2n + 1 = (n+1)^2$$
$$n \in N$$

1. Czy S należy do N ? Tak
2. Dla $n = 0$: $L = 1$, $P = 1$, $L = P$
3. Założenie: $1 + 3 + 5 + ... + 2n + 1 = (n+1)^2$
   Teza: $1 + 3 + 5 + ... + 2n + 1 + 2(n+1) + 1 = (n+1+1)^2$

   Założenie podstawiamy do tezy:

   $(n+1)^2 + 2(n+1) + 1 = (n+1+1)^2$
   $L = n^2 + 2n + 1 + 2n + 2 + 1 = n^2 + 4n + 4$
   Ze wzoru skroconego mnozenia: $(n+2)^2 = (n+2)^2$

1.4. Zadania do domu

Pytanie

1)$\forall n \in N \setminus \{0\}\ 1 + 4 + 7 + ... + (3n-2) = \frac{1}{2}n(3n-1)$
2)$\forall n \in N \setminus \{0\}\ 4^n - 1$ jest podzielne przez 3
3)Wieza Hanoi - pokazac, ze $\forall n \in N \setminus \{0\}$ liczba ruchow zeby przeniesc cala wieze wynosi $2^n - 1$

1. ✓

1) $S \in N$

2) $n = 1 \implies 1 = \frac{1}{2}(3-1) = 1$

3) Zalozenie: $1 + 4 + 7 + ... + (3n-2) = \frac{1}{2}n(3n-1)$

Teza: $1 + 4 + 7 + ... + (3n-2) + (3(n+1)-2) = \frac{1}{2}(n+1)(3(n+1)-1)$

$L = \frac{1}{2}n(3n-1) + 3n + 1 = \frac{3}{2}n^2 + \frac{5}{2}n + 1 = \frac{1}{2}(3n^2+5n+2) = \frac{1}{2}(3n+2)(n+1)$

$P = \frac{1}{2}(n+1)(3n+2)$

$L = P$

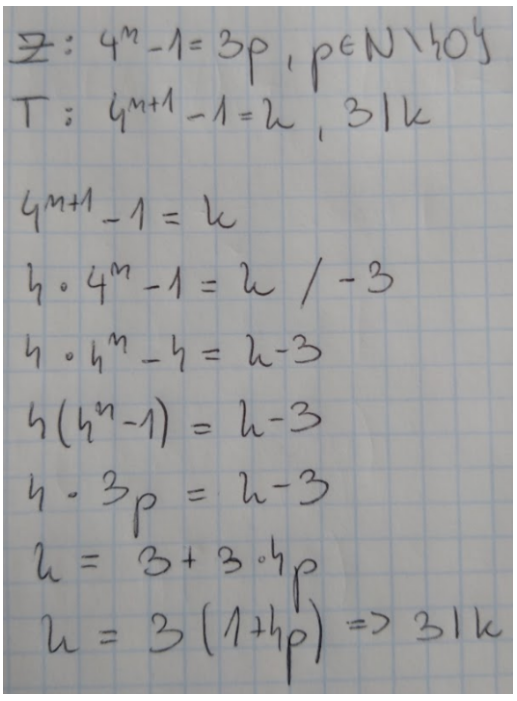
2. ✓

1) $S \in N$

2) $n = 1 : \ 4^n - 1 = 3$

3) Zalozenie: $4^n - 1 = 3p, p \in N \setminus \{0\}$

Teza: $4^{(n+1)} - 1 = k, k$ - liczba podzielna przez 3



3. ✗

2 Ćwiczenia 2 (K1) - 02.03.2020 - podstawy funkcji prymitywnie rekurencyjnych

2.1. Definiowanie zbiorów liczbowych przez rekurencje

Dany zbiór liczbowy możemy definiować przy pomocy rekurencji, przykładowo:

- zbiór liczb naturalnych N

$$0 \in N$$
$$n \in N \implies n+1 \in N$$

- całkowite

$$0 \in L$$
$$n \in L \implies n+1 \in L$$
$$n \in L \implies n-1 \in L$$

  Lub alternatywnie:

$$0 \in L$$
$$n \in L \implies -n \in L$$

- wymierne

$$0 \in L$$
$$n \in L \implies -n \in L$$
$$n \in L,\ x \neq 0 \implies \frac{n}{x} \in L$$

Takie definicje możemy stosować dla dowolnie wybranych przez nas zbiorów liczb.

Przykładowo możemy w ten sposób opisać wszystkie liczby podzielne przez 5:

$$5 \in L$$
$$x \in L \implies x+5 \in L$$

lub alternatywnie

$$5 \in L$$
$$n \in L,\ k \in N^{+} \implies n \cdot k \in L$$

2.2. Definiowanie języka przez rekurencje

Przyjmujemy pewną bazę i pewne reguły, które stosujemy do definiowania języka. Przykładowo dla języka logiki zerowego rzędu:

$$
\begin{aligned}
&p, q, r, \ldots \text{ - baza} \in L \\
&p \wedge q \in L \\
&p \vee q \in L \\
&p \implies q \in L \\
&p \Leftrightarrow q \in L \\
&\neg p \in L
\end{aligned}
$$

2.3. Definiowanie funkcji przez rekurencje

Również funkcje możemy zdefiniować rekurencyjnie. Przykładowo mając dodawanie:

$$f(x, y) = x + y$$

Możemy tą funkcję zdefiniować rekurencyjnie jako:

$$
\begin{aligned}
&f(0, 0) = 0 \\
&f(n, m + 1) = f(n, m) + 1 \\
&f(n + 1, m) = f(n, m) + 1
\end{aligned}
$$

Zadanie

Zdefiniować rekurencyjnie funkcje: $2^n \cdot 3^{m+1}$.

$$
\begin{aligned}
&f(0, 0) = 3 \\
&f(n, m + 1) = f(n, m) \cdot 3 \\
&f(n + 1, m) = f(n, m) \cdot 2
\end{aligned}
$$

2.4. Definiowanie obliczalności przez rekurencje

W informatyce chcielibyśmy wiedzieć czy dana funkcja jest obliczalna. Pierwsza próba zdefiniowania zbioru funkcji obliczalnych została podjęta przy użyciu tzw. funkcji prymitywnie rekurencyjnych. Ta definicja nie obejmuje wszystkich funkcji obliczalnych, ale ich znaczną większość dlatego ją sobie omówimy. Funkcja prymitywnie rekurencyjna to funkcja postaci $f : N^n \to N$. Zbiór wszystkich funkcji prymitywnie rekurencyjnych oznaczamy symbolem C.

Aby funkcja była prymitywnie rekurencyjna musi mieć jedną z poniższych form:

1. Jest funkcją *stałą*, tzn. $f(x_1, x_2, \ldots, x_n) = m$, taką funkcję oznaczamy w skrócie M_m^n
2. Jest funkcją, która dodaje jeden do pewnego argumentu, tzn. $f(x_1, x_2, \ldots, x_n) = x_i + 1$, taką funkcję nazywamy *następnikiem* i oznaczamy w skrócie $S(x)$

3. Jest funkcją, która przyjmuje wartość równą jednemu z jej argumentów, tzn. $f(x_1, x_2, \ldots, x_n) = x_i$, taką funkcję nazywamy *projekcją* i oznaczamy p_i^n

Powyższe funkcje to podstawowe składowe funkcji prymitywnie rekurencyjnych, inne funkcje mogą być tworzone tylko na ich podstawie.

Funkcje prymitywnie rekurencyjne możemy tworzyć z funkcji podstawowych na dwa sposoby:

1. jako złożenie funkcji prymitywnie rekurencyjnych, tzn. jeśli $g_1, g_2, g_3, \ldots g_n, h \in C$, to oznacza, że również funkcja $f(x_1, x_2, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), g_2(x_1, \ldots, x_n), \ldots, g_n(x_1, \ldots, x_n)) \in C$

   Przykładowo funkcja $f(x) = x+5$ może być zapisana jako $f(x) = S(S(S(S(S(x)))))$, prawa strona jest złożeniem funkcji prymitywnie rekurencyjnych, a więc f jest również prymitywnie rekurencyjna.

2. jeśli zdefiniujemy f za pomocą rekurencji dwóch funkcji prymitywnie rekurencyjnych g oraz h, to f jest prymitywnie rekurencyjna.

Ten drugi punkt omówmy sobie trochę dokładniej zanim podamy formalny wzór. Drugi punkt mówi o tym, że jeśli mamy daną funkcję $f(x_1, x_2, \ldots, x_n)$, to możemy pokazać, że jest ona prymitywnie rekurencyjna jeśli pokażemy, że spełnione są dwa poniższe warunki

- wstawiając za ostatni argument, czyli za x_n wartość 0 otrzymamy funkcję prymitywnie rekurencyjną postaci $g(x_1, x_2, \ldots, x_{n-1})$
- następnie chcemy zdefiniować co się dzieje jeśli ostatni argument zwiększymy o 1, czyli dla $x_n = m + 1$, a więc musimy zapisać, że $f(x_1, x_2, \ldots, x_{n-1}, m + 1) = \ldots$

  Prawą stronę uzupełniamy na podstawie postaci funkcji, tak jak definiowaliśmy funkcje rekurencyjnie.

  Czyli przykładowo dla funkcji $f(x, y) = x + y$ wiemy, że będzie $f(x, m + 1) = f(x, m) + 1$.

  Cała sztuka teraz polega na znalezieniu takiej funkcji prymitywnie rekurencyjnej h, która przyjmując jako argumenty $x_1, \ldots, x_{n-1}$, m oraz $f(x_1, \ldots, x_{n-1}, m)$, przekształci lewą stronę w prawą. Szukamy więc takiej funkcji h, że

$$f(x_1, x_2, \ldots, x_{n-1}, m + 1) = h(x_1, \ldots, x_{n-1}, m, f(x_1, \ldots, x_{n-1}, m))$$

Podsumowując, nasze zadanie polega na znalezieniu dwóch funkcji prymitywnie rekurencyjnych g, h takich, że:

$$f(x_1, x_2, \ldots, x_{n-1}, 0) = g(x_1, x_2, \ldots, x_{n-1})$$
$$f(x_1, x_2, \ldots, x_{n-1}, m + 1) = h(x_1, \ldots, x_{n-1}, m, f(x_1, \ldots, x_{n-1}, m))$$

Pokażmy to na przykładzie:

Zadanie

Pokazać, że funkcja $f(x, y) = x + y$ jest jest prymitywnie rekurencyjna (czyli, że dodawanie jest funkcją prymitywnie rekurencyjną).

Spróbujemy pokazać, że funkcję tą możemy zapisać przy użyciu rekurencji funkcji prymitywnie rekurencyjnych, będziemy więc chcieli znaleźć takie funkcje g, h, że spełnione będzie:

$$f(x, 0) = g(x)$$
$$f(x, m + 1) = h(x, m, f(x, m))$$

Funkcję g łatwo uzyskać podstawiając jako ostatni argument 0:

$$f(x, 0) = x + 0 = x$$
$$g(x) = x = p_1^1(x) \text{ - f. prymitywnie rekurencyjna}$$

Teraz szukamy funkcji h, najpierw policzmy lewą stronę równania:

$$f(x, m + 1) = x + m + 1 = f(x, m) + 1$$

Dlaczego zapisaliśmy wynik przy użyciu $f(x, m)$? Ponieważ taka funkcja jest argumentem funkcji h i to może nam pomóc w poszukiwaniu funkcji h.

Szukamy więc takiej funkcji prymitywnie rekurencyjnej h, która spełnia poniższe równanie:

$$h(x, m, f(x, m)) = f(x, m) + 1$$

Łatwo zauważyć, że jest to projekcja trzeciego z argumentów powiększona o 1, a więc nasza funkcja h jest postaci:

$$h(x, m, f(x, m)) = S(p_3^3(x, m, f(x, m))$$

Funkcja ta jest prymitywnie rekurencyjna jako złożenie funkcji prymitywnie rekurencyjnych.

Zadanie

Udowodnić, że mnożenie jest prymitywnie rekurencyjne, czyli, że $f(x, y) = x \cdot y$ jest funkcją prymitywnie rekurencyjną. Pamiętaj, że pokazaliśmy już, że dodawanie jest funkcją prymitywnie rekurencyjną.

Ponownie zaczynamy od pierwszego z równań i obliczamy lewą stronę:

$$f(x, 0) = x \cdot 0 = 0$$

Szukamy takiej funkcji prymitywnej rekurencyjnie g, która dla x będzie miała wartość 0:

$$g(x) = 0 = M_0^1$$

W kolejnym kroku poszukujemy funkcji h. Rozpiszmy lewą stronę równania:

$$f(x, m + 1) = x \cdot (m + 1) = x \cdot m + x = f(x, m) + x$$

Szukamy więc funkcji rekurencyjnie prymitywnej h, która spełniać będzie równanie:

$$h(x, m, f(x, m)) = f(x, m) + x$$

Mamy tutaj sumę projekcji pierwszego i trzeciego argumentu:

$$h(x, m, f(x, m)) = f(x, m) + x = p_1^3(x, m, f(x, m)) + p_3^3(x, m, f(x, m))$$

Dodatkowo wiemy, że dodawanie jest funkcją rekurencyjnie prymitywną, a więc mamy złożenie funkcji prymitywnie rekurencyjnych, które jest funkcją prymitywnie rekurencyjną. Ostatecznie więc:

$$g(x) = 0 = M_0^1$$
$$h(x, m, f(x, m)) = f(x, m) + x = p_1^3(x, m, f(x, m)) + p_3^3(x, m, f(x, m))$$

2.5. Zadania do domu

Zadanie Zadania do domu

Pokazać, że poniższe funkcję są prymitywnie rekurencyjne (w dziedzinie liczb naturalnych):

1. $min(x, y)$
2. $max(x, y)$
3. x^y (* - dwie zmienne)
4. $a \dot{-} b = \begin{cases} a - b, & \text{gdy } a > b \\ 0 & \text{w p.p} \end{cases}$
5. $|a - b|$
6. $a!$
7. $a|b = \begin{cases} 0, & \text{gdy } b = k \cdot a \\ 1, & w\ p.p \end{cases}$

Generalnie należy zacząć rozwiązywanie od przykładu 4, bo pozostałe są w większości złożeniem tej funkcji...

1. ✓
   Pamiętajmy, że jesteśmy cały czas w dziedzinie naturalnych. Ponadto najpierw trzeba wykazać max, bo min jest jego złożeniem...

   $$min(x, y) = x + y - max(x, y)$$

   $f(x, 0) = 0 \Rightarrow g(x) = M_1^1$
   $f(x, m + 1) = x + (m + 1) - max(x, (m + 1)) = x + S(m) - max(x, S(m)) \Rightarrow$
   $h(x, m, f(x, m)) = p_1^2(x, m, f(x, m)) + S(p_2^3(x, m, f(x, m))) - max(p_1^3(x, m, f(x, m)), S(p_2^3(x, m, f(x, m))))$

2. ✓
   Pamiętajmy, że jesteśmy cały czas w dziedzinie naturalnych...
   Zauważmy, że

   $$max(n, m) = \begin{cases} n > m \Rightarrow (n \dot{-} m) + m = n - m + m = n \\ n < m \Rightarrow (n \dot{-} m) + m = 0 + m = m \\ n = m \Rightarrow (n \dot{-} m) + m = 0 + m = mm = n \end{cases}$$

   Stąd

   $$max(x, y) = (x \dot{-} y) + y$$

   $max(x, 0) = x \Rightarrow g(x) = p_1^1(x)$
   $max(x, m + 1) = (x \dot{-} (m + 1)) + m + 1 = (x \dot{-} S(m)) + S(m) \Rightarrow$
   $h(x, m, f(x, m)) = (p_1^3(x, m, f(x, m)) - S(p_2^3(x, m, f(x, m))) + S(p_2^3(x, m, f(x, m))$

3. ✗
Zaczynamy tak jak zawsze:

$$f(x,0) = x^0 = 1$$
$$f(x,m+1) = f(x,m) \cdot x$$

$$g(x) = M_1^1$$
$$h(x,m,f(x,m)) = p_3^3 \cdot p_1^3$$

Tu coś jeszcze trzeba dodać dla y^x, ale nie wiem dokładnie o co chodzi...

4. ✓
Zakładając, że rozważamy tylko liczby naturalne (dla wszystkich liczb musielibyśmy rozbijać na dwa warunki):

$$f(a,0) = a$$
$$f(a,m+1) = a - (m+1) = a - m - 1 = f(a,m) - 1 = pop(f(a,m))$$

$$g(a) = p_1^1(a)$$
$$h(a,m,f(a,m)) = pop(p_1^3(a,m,f(a,m)))$$

Pokażemy teraz, że funkcja

$$f(x) = \begin{cases} x-1, \ x \neq 0 \\ 0, \ x = 0 \end{cases}$$

jest prymitywnie rekurencyjna, wtedy nasza funkcja h będzie złożeniem funkcji prymitywnie rekurencyjnych.

$$f(0) = 0$$
$$f(m+1) = m$$

$$g(x) = M_0^1$$
$$h(m,f(m)) = p_1^2(m,f(m))$$

5. ✓

$$|x-y| = \begin{cases} x-y, x \geq y \\ y-x, x < y \end{cases}$$

Zauważmy,że

$$|x-y| = (x \dot{-} y) + (y \dot{-} x)$$

$f(x,0) = x \Rightarrow g(x) = x = p_1^1(x)$
$f(x,m+1) = (x \dot{-} (m+1)) + ((m+1) \dot{-} x) = (x \dot{-} S(m)) + (S(m) \dot{-} x) \Rightarrow$
$h(x,m,f(x,m)) = (p_1^3(x,m,f(x,m)) \dot{-} S(p_2^3(x,m,f(x,m)))) + (S(p_2^3(x,m,f(x,m))) \dot{-} p_1^3(x,m,f(x,m)))$

6. ✓
   Pamiętajmy, że $0! = 1$

$$f(0) = 1$$
$$f(m+1) = (m+1)! = (m+1) \cdot f(m)$$

$$g(x) = M_1^1$$
$$h(m, f(m)) = S(p_1^2(m, f(m))) \cdot p_2^2(m, f(m))$$

7. ✓
   Okazało się, że tego się nie da pokazać przy pomocy tej metody i zostało to pokazane dopiero na koniec następnych zajęć...

Dorzucam jeszcze kilka rozwiązań z książek

- An Introduction to Recursive Function Theory (Cutland)

Rozwiązania te dowodzą, że funkcja jest obliczalna:

4.5. *Theorem*
The following functions are computable. (Proofs are given as the functions are listed.)

(*a*) $x+y$ *Proof.* Example 4.3(*a*) gives a definition by recursion from the computable functions x and $z+1$.

(*b*) xy *Proof.* $x0=0$,
$x(y+1)=xy+x$,
is a definition by recursion from the computable functions $\mathbf{0}(x)$ and $z+x$.

(*c*) x^y *Proof.* $x^0=1$,
$x^{y+1}=x^y x$; by recursion and (*b*).

(*d*) $x\dot{-}1$ *Proof.* $0\dot{-}1=0$,
$(x+1)\dot{-}1=x$; by recursion.

(*e*) $x\dot{-}y=\begin{cases}x-y & \text{if } x\geq y,\\ 0 & \text{otherwise.}\end{cases}$ (*cut-off subtraction*)
Proof. $x\dot{-}0=x$,
$x\dot{-}(y+1)=(x\dot{-}y)\dot{-}1$; by recursion and (*d*).

(*f*) $\mathrm{sg}(x)=\begin{cases}0 & \text{if } x=0,\\ 1 & \text{if } x\neq 0.\end{cases}$ (cf. exercises 1-3.3(1*a*))
Proof. $\mathrm{sg}(0)=0$,
$\mathrm{sg}(x+1)=1$; by recursion.

(*g*) $\overline{\mathrm{sg}}(x)=\begin{cases}1 & \text{if } x=0,\\ 0 & \text{if } x\neq 0.\end{cases}$ (cf. example 1-4.2(*b*))
Proof. $\overline{\mathrm{sg}}(x)=1\dot{-}\mathrm{sg}(x)$; by substitution, (*e*) and (*f*).

(*h*) $|x-y|$ *Proof.* $|x-y|=(x\dot{-}y)+(y\dot{-}x)$; by substitution, (*a*) and (*e*).

(*i*) $x!$ *Proof.* Example 4.3(*b*) gives a definition by recursion from computable functions.

(*j*) $\min(x,y)=$ *minimum of* x *and* y.
Proof. $\min(x,y)=x\dot{-}(x\dot{-}y)$; by substitution.

(*k*) $\max(x,y)=$ *maximum of* x *and* y.
Proof. $\max(x,y)=x+(y\dot{-}x)$; by substitution.

(*l*) $\mathrm{rm}(x, y)$ = *remainder when* y *is divided by* x (*to obtain a total function, we adopt the convention* $\mathrm{rm}(0, y) = y$).

Proof. We have

$$\mathrm{rm}(x, y+1) = \begin{cases} \mathrm{rm}(x, y)+1 & \text{if } \mathrm{rm}(x, y)+1 \neq x, \\ 0 & \text{if } \mathrm{rm}(x, y)+1 = x. \end{cases}$$

This gives the following definition by recursion:

$$\mathrm{rm}(x, 0) = 0,$$
$$\mathrm{rm}(x, y+1) = (\mathrm{rm}(x, y)+1)\,\mathrm{sg}(|x - (\mathrm{rm}(x, y)+1)|).$$

The second equation can be written

$$\mathrm{rm}(x, y+1) = g(x, \mathrm{rm}(x, y))$$

where $g(x, z) = (z+1)\,\mathrm{sg}(|x - (z+1)|)$; and g is computable by several applications of substitution. Hence $\mathrm{rm}(x, y)$ is computable.

(*m*) $\mathrm{qt}(x, y)$ = *quotient when* y *is divided by* x (*to obtain a total function we define* $\mathrm{qt}(0, y) = 0$).

Proof. Since

$$\mathrm{qt}(x, y+1) = \begin{cases} \mathrm{qt}(x, y)+1 & \text{if } \mathrm{rm}(x, y)+1 = x, \\ \mathrm{qt}(x, y) & \text{if } \mathrm{rm}(x, y)+1 \neq x, \end{cases}$$

we have the following definition by recursion from computable functions:

$$\mathrm{qt}(x, 0) = 0,$$
$$\mathrm{qt}(x, y+1) = \mathrm{qt}(x, y) + \overline{\mathrm{sg}}(|x - (\mathrm{rm}(x, y)+1)|).$$

(*n*) $\mathrm{div}(x, y) = \begin{cases} 1 & \text{if } x \mid y \ (x \textit{ divides } y), \\ 0 & \text{if } x \nmid y. \end{cases}$

(*We adopt the convention that* $0 \mid 0$ *but* $0 \nmid y$ *if* $y \neq 0$.) *Hence* $x \mid y$ *is decidable* (recall definition 1-4.1).

Proof. $\mathrm{div}(x, y) = \overline{\mathrm{sg}}(\mathrm{rm}(x, y))$, computable by substitution. □

- Incompleteness and Computability

Proposition 2.7. *The exponentiation function* $\exp(x, y) = x^y$ *is primitive recursive.*

Proof. We can define exp primitive recursively as

$$\exp(x, 0) = 1$$

$$\exp(x, y + 1) = \text{mult}(x, \exp(x, y)).$$

Strictly speaking, this is not a recursive definition from primitive recursive functions. Officially, though, we have:

$$\exp(x, 0) = f(x)$$
$$\exp(x, y + 1) = g(x, y, \exp(x, y)).$$

where

$$f(x) = \text{succ}(\text{zero}(x)) = 1$$
$$g(x, y, z) = \text{mult}(P_0^3(x, y, z), P_2^3(x, y, z)) = x \cdot z$$

and so f and g are defined from primitive recursive functions by composition. □

Proposition 2.8. *The predecessor function* $\mathrm{pred}(y)$ *defined by*

$$\mathrm{pred}(y) = \begin{cases} 0 & \text{if } y = 0 \\ y - 1 & \text{otherwise} \end{cases}$$

is primitive recursive.

Proof. Note that

$$\mathrm{pred}(0) = 0 \text{ and}$$
$$\mathrm{pred}(y + 1) = y.$$

This is almost a primitive recursive definition. It does not, strictly speaking, fit into the pattern of definition by primitive recursion, since that pattern requires at least one extra argument x. It is also odd in that it does not actually use $\mathrm{pred}(y)$ in the definition of $\mathrm{pred}(y + 1)$. But we can first define $\mathrm{pred}'(x, y)$ by

$$\mathrm{pred}'(x, 0) = \mathrm{zero}(x) = 0,$$
$$\mathrm{pred}'(x, y + 1) = P_1^3(x, y, \mathrm{pred}'(x, y)) = y.$$

and then define pred from it by composition, e.g., as $\mathrm{pred}(x) = \mathrm{pred}'(\mathrm{zero}(x), P_0^1(x))$. □

Proposition 2.9. *The factorial function* $\mathrm{fac}(x) = x! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot x$ *is primitive recursive.*

Proof. The obvious primitive recursive definition is

$$\begin{aligned} \mathrm{fac}(0) &= 1 \\ \mathrm{fac}(y+1) &= \mathrm{fac}(y) \cdot (y+1). \end{aligned}$$

Officially, we have to first define a two-place function h

$$\begin{aligned} h(x,0) &= \mathrm{const}_1(x) \\ h(x,y) &= g(x,y,h(x,y)) \end{aligned}$$

where $g(x,y,z) = \mathrm{mult}(P_2^3(x,y,z), \mathrm{succ}(P_1^3(x,y,z)$ and then let

$$\mathrm{fac}(y) = h(P_0^1(y), P_0^1(y))$$

From now on we'll be a bit more laissez-faire and not give the official definitions by composition and primitive recursion. □

Proposition 2.10. *Truncated subtraction,* $x \dot{-} y$*, defined by*

$$x \dot{-} y = \begin{cases} 0 & \text{if } x > y \\ x - y & \text{otherwise} \end{cases}$$

is primitive recursive.

Proof. We have:

$$\begin{aligned} x \dot{-} 0 &= x \\ x \dot{-} (y+1) &= \mathrm{pred}(x \dot{-} y) \end{aligned}$$ □

Proposition 2.11. *The distance between x and y, $|x - y|$, is primitive recursive.*

Proof. We have $|x - y| = (x \dot{-} y) + (y \dot{-} x)$, so the distance can be defined by composition from $+$ and $\dot{-}$, which are primitive recursive. □

Proposition 2.12. *The maximum of x and y, $\max(x, y)$, is primitive recursive.*

Proof. We can define $\max(x, y)$ by composition from $+$ and $\dot{-}$ by

$$\max(x, y) = x + (y \dot{-} x).$$

If x is the maximum, i.e., $x \geq y$, then $y \dot{-} x = 0$, so $x + (y \dot{-} x) = x + 0 = x$. If y is the maximum, then $y \dot{-} x = y - x$, and so $x + (y \dot{-} x) = x + (y - x) = y$. □

3 Ćwiczenia 3 (K1) - 09.03.2020 - więcej funkcji p.rek. + relacje

Kolokwium 1:

- Teoretycznie za dwa tygodnie ma być kolokwium (tj. 23.03.2020), ale do 25.03.2020 nie ma zajęć więc nie wiadomo...
- Do kolokwium wchodzi zakres wszystkiego co zrobiliśmy od początku semestru
- Na kolokwium można korzystać z wszystkich dowodów, które pokazaliśmy na ćwiczeniach, np. z tego, że dodawanie, mnożenie, max, min itd. są prymitywnie rekurencyjne

3.1. Poszerzamy klasę funkcji prymitywnie rekurencyjnych

Na ostatnich zajęciach zdefiniowaliśmy sobie pewne podstawowe postacie funkcji, które mówią nam o tym, że dana funkcja jest prymitywnie rekurencyjna. Teraz pokażemy kilka przykładów bardziej skomplikowanych funkcji dla których wykażemy, że są prymitywnie rekurencyjne.
Funkcje, które będziemy rozważać to:

- suma ograniczona
- iloczyn ograniczony
- jeden w zerze i zero dla pozostałych, oznaczać będziemy $z(x)$
- zero w zerze i jeden dla pozostałych, oznaczać będziemy $pos(x)$

Na początek zdefiniujmy sobie funkcje:

$$\sum_{t<y} f(x_1, x_2, \ldots, x_n, t) = \begin{cases} 0 \text{ ,gdy } y = 0 \\ f(x_1, \ldots, x_n, 0) + f(x_1, \ldots, x_n, 1) + ... + f(x_1, \ldots, x_n, y-1) \text{ ,gdy y > 0} \end{cases}$$

$$\prod_{t<y} f(x_1, \ldots, x_n, t) = \begin{cases} 1 \text{ ,gdy y = 0} \\ f(x_1, \ldots, x_n, 0) \cdot ... \cdot f(x_1, \ldots, x_n, y-1) \text{ ,gdy y>0} \end{cases}$$

Pierwsza z tych funkcji nazywana jest ograniczoną sumą funkcji f (bounded sum), a druga ograniczonym iloczynem funkcji f.

Jeżeli funkcja f jest funkcją prymitywnie rekurencyjną, to funkcjami prymitywnie rekurencyjnymi są również ograniczone dodawanie i ograniczone mnożenie dla tej funkcji:

$$f(x_1, \ldots, x_n, y) \text{ - f. prymitywnie rekurencyjna} \Rightarrow$$
$$s = \sum_{t<y} f(x_1, \ldots, x_n, t) \text{ - f. prymitywnie rekurenyjna}$$
$$p = \prod_{t<y} f(x_1, \ldots, x_n, t) \text{ - f. prymitywnie rekurenyjna}$$

Pokażemy dowód dla sumy ograniczonej funkcji f:

$$\sum_{t<y} f(x_1, \ldots, x_n, 0) = s(x_1, \ldots, x_n, 0) = 0 \Rightarrow g(x_1, \ldots, x_n) = M_0^n$$
$$\sum_{t<y} f(x_1, \ldots, x_n, m+1) = s(x_1, \ldots, x_n, m+1) = f(x_1, \ldots, x_n, 0) + f(x_1, \ldots, x_n, 1) + \cdots + f(x_1, \ldots, x_n, m) \Rightarrow$$
$$h(x_1, \ldots, x_n, m, s(x_1, \ldots, x_n, m)) = p_{n+2}^{n+2}(x_1, \ldots, x_n, m, s(x_1, \ldots, x_n, m)) + f(p_1^{n+2}, p_2^{n+2}, \ldots, p_{n+1}^{n+2})$$

Kolejną funkcją dla której chcemy pokazać, że jest prymitywnie rekurencyjna jest funkcja:

$$z(x) = \begin{cases} 1, & x = 0 \\ 0, & \text{w p.p.} \end{cases}$$

Dowód:

$z(0) = 1 \Rightarrow g(x) = M_1^1$
$z(m+1) = \ldots$
$h(m, f(m)) = pop(p_2^2(m, f(m)))$ - ??? nie wiem, tak było na ćwiczeniach...

Następną funkcją dla której pokażemy, że jest prymitywnie rekurencyjna jest funkcja:

$$pos(x) = \begin{cases} 0, & x = 0 \\ 1, & \text{w p.p.} \end{cases}$$

Dowód:

$pos(0) = 0 \Rightarrow g(x) = M_0^1$
$pos(m+1) = M_1^2$

3.2. Przykład

Spróbujemy teraz wykorzystać zdobytą wiedzę o nowych funkcjach do dowiedzenia, że poniższa funkcja jest prymitywnie rekurencyjna

$$f(x, y) = \begin{cases} 1, & x \leq y \\ 0, & \text{w p.p.} \end{cases}$$

Dowód:

$f(x, 0) = z(x)$ - gdzie z to zdefiniowana przez nas powyżej funkcja
$g(x) = z(p_1^1(x))$
$f(x, m+1) = z(x \dot{-} S(m))$
$h(x, m, f(x, m)) = z(p_1^3(x, m, f(x, m)) \dot{-} S(p_2^3(x, m, f(x, m))))$

Zauważmy, że wpadnięcie na coś takiego, nie jest w takich sytuacjach oczywiste. Warto zapamiętać, że przy tego rodzaju funkcjach (gdzie w pewnym przypadku mamy 1, a w innym 0) warto zawsze pomyśleć o funkcjach $z\ pos$ oraz $\dot{-}$.

3.3. Relacje i funkcje charakterystyczne

Ostatnia funkcja dla której wykazaliśmy, że jest prymitywnie rekurencyjna, była również tzw. funkcją charakterystyczną. Funkcje charakterystyczne, to funkcje postaci:

$$f(x) = \begin{cases} 1, x \in B \\ 0, x \notin B \end{cases}$$

Funkcje takie mogą opisywać zbiory czy też relacje.

Mówimy, że zbiór/relacja opisana na zbiorze liczb naturalnych $R \subset N$ jest prymitywnie rekurencyjna wtedy gdy jej funkcja charakterystyczna jest prymitywnie rekurencyjna. Funkcję charakterystyczną relacji R zapisujemy jako:

$$\chi(x_1, x_2) = \begin{cases} 1, & \text{gdy } (x_1, x_2) \in R \\ 0, & \text{w p.p.} \end{cases}$$

3.3.1. Własności relacji prymitywnie rekurencyjnych

- $Q \subset N^n$ - relacja
  $g_1, g_2, \ldots, g_n$ - p. rek.

  to

  Tu byl jakis dziwny wzor...

- $\overline{Q}$ - dokmnięcie relacji, czy jest p. rekurencyjne gdy Q jest ?

  Czym jest domknięcie ?

  $A \subset U$
  $A' = U - A$ - domknięcie

  Tak, domknięcie relacji prymitywnie rekurencyjnej jest prymitywnie rekurencyjne, bo wtedy po prostu f. charakterystyczna ma zamienione przedziały:

  $$\chi(x_1, x_2) = \begin{cases} 0 \\ 1, \text{ w p.p.} \end{cases} \Rightarrow \begin{cases} 1 \\ 0, \text{ w p.p.} \end{cases}$$

  Czyli zachodzi:

  $z(\chi_Q) = \chi_{\overline{Q}}$ - gdzie z to funkcja zdefiniowana na poprzedniej stronie

Zadanie

Relacje Q, R są prymitywnie rekurencyjne, pokazać, że:

- ich iloczyn jest prymitywnie rekurencyjny
- ich suma jest prymitywnie rekurencyjna

Wiemy, że relacja jest opisana funkcją charakterystyczną, która przyjmuje w pewnych przedziałach 1, a w pewnych 0. Jeśli zatem mamy iloczyn dwóch takich relacji, to będzie mieć 1 tylko tam gdzie

obydwie mają 1. Czyli możemy pomnożyć przez siebie te dwie funkcje charakterystyczne, a mnożenie jest prymitywnie rekurencyjne.

$$\chi_{Q,R} = \chi_Q \cdot \chi_R$$

a dla sumy podobnie, ale tutaj mamy 1, jeśli przynajmniej jedna relacja ma 1, a więc skorzystamy z wcześniej zdefiniowanej funkcji pos:

$$\chi_{Q \cup R} = pos(\chi_Q + \chi_R)$$

Zadanie

Wykazać, że relacje są prymitywnie rekurencyjne:

1. $\{(x, y) : x > y\}$
2. $\{(x, y) : x = y\}$

Dla tego typu zadań gdzie mamy do czynienia z relacjami najlepiej zacząć od zdefiniowania f. charakterystycznej albo rozwiązać zadanie pokazując, że dana relacja jest dopełnieniem jakiejś innej, która jest p. rekurencyjna.

1. Funkcja charakterystyczna tej relacji to:

$$\chi_R(x, y) = \begin{cases} 1, x > y \\ 0, \text{ w p.p.} \end{cases}$$

Teraz przekształćmy ją do zapisu z innymi f. prymitywnymi:

$$\chi_R(x, y) = pos(x \dot{-} y)$$

A zatem:

$$\chi_R(x, 0) = 0 \Rightarrow g(x) = M_0^1$$
$$\chi_R(x, m + 1) = pos(x \dot{-} (m + 1)) = pos(x \dot{-} S(m))$$
$$h(x, m, f(x, m)) = pos(p_1^3(x, m, f(x, m)) - S(p_1^2(x, m, f(x, m))))$$

Stosując drugie z podejść, moglibyśmy pokazać, że relacja jest po prostu domknięciem relacji $\{(x, y) : x \leq y\}$

2. Tutaj f. charakterystyczna jest postaci:

$$\chi_R(x, y) = \begin{cases} 1, x = y \\ 0. \text{ w p.p.} \end{cases}$$

Przekształćmy ją sobie do zapisu z f. prymitywnymi:

$$\chi_R(x, y) = z(|x - y|)$$

I dalej po prostu przekształcamy na projekcję analogicznie jak w pierwszym przykładzie.

3.4. Funkcje zadane w przedziałach - jak pokazać, że są p. rek. ?

Czasami może się zdarzyć, że mamy funkcję zadaną na przedziałach, a jej wartościami nie są same zera i jedynki tzn. funkcja jest dana w postaci:

$$f(x,y) = \begin{cases} g(x,y), \text{ gdy } R(x,y) \text{ czyli gdy f.char. ma wartość 1} \\ h(x,y), \text{ gdy } \neg R(x,y) \text{ czyli gdy f.char. ma wartość 0} \end{cases}$$

Przykładem takiej funkcji może być:

$$f(x,y) = \begin{cases} x \cdot y, \text{ gdy } x < y \\ x + y, \text{ w p.p.} \end{cases}$$

Jak pokazać, że taka funkcja jest prymitywnie rekurencyjna ?

Wystarczy, że pokażemy, że $R(x)$ jest prymitywnie rekurencyjna oraz, że g i h są prymitywnie rekurencyjne. Dzieje się tak dlatego, że funkcję f możemy zapisać inaczej jako:

$$f(x) = g(x) \cdot \chi_R + h(x) \cdot \chi_{\overline{R}}$$

A wiemy że dodawania i mnożenie jest prymitywnie rekurencyjne.

Podobnie mając np. funkcje w postaci:

$$f(x) = \begin{cases} g(x), R(x) \wedge Q(x) \\ h(x), R(x) \wedge \neg Q(x) \\ k(x), \neg R(x) \wedge Q(x) \\ i(x), \neg R(x) \wedge \neg Q(x) \end{cases}$$

możemy zapisać taką funkcję jako:

$$f(x) = g(x)\chi_{R \bigcap Q} + h(x)\chi_{R \bigcap \overline{Q}} + k(x)\chi_{\overline{R} \bigcap Q} + i(x)\chi_{\overline{R} \bigcap Q}$$

3.5. Relacje z kwantyfikatorami

Jeżeli mamy relację prymitywnie rekurencyjną $R(x_1, ..., x_n, t)$, to relacjami rekurencyjnie prymitywnymi są również relacje:

$$\forall_{t<y} R(x_1, ..., x_n, t) = \prod_{t<y} \chi_R(x_1, ..., x_n, t) = \chi_R(x_1, ..., x_n, 0) \cdot \chi_R(x_1, ..., x_n, 1) \cdot ... \cdot \chi_R(x_1, ..., x_n, y-1)$$

oraz

$$\exists_{t<y} R(x_1, x_2, ..., x_n, t) = pos(\sum_{t<y} \chi_R(x_1, ..., x_n, t)) = pos(\chi_R(x_1, ..., x_n, 0) + \chi_R(x_1, ..., x_n, 1) +$$

$$... + \chi_R(x_1, ..., x_n, y-1))$$

Teraz możemy wrócić do przykładu numer 7 z poprzednich zadań do domu, ponieważ możemy go rozwiązać przy użyciu relacji:

$$a|b = f(a,b) = \begin{cases} 1, \text{ gdy } b = a \cdot k \\ 0, \text{ w p.p.} \end{cases}$$

To jest równoważne:

$$\exists_{k<b+1} b = k \cdot a$$

Relacja $b = k \cdot a$ jest prymitywnie rekurencyjna, bo składa się z mnożenia i równości, co możemy rozpisać:

$$f(a,b) = (b = k \cdot a) = RownaSie(b, Mnozenie(k,a))$$

jest to niewątpliwie złożenie funkcji prymitywnie rekurencyjnych. Pokazaliśmy bowiem, że mnożenie jest prymitywnie rekurencyjne jeszcze na zajęciach 2. Pokazaliśmy także na tych laboratoriach, że funkcja $f(x,y) = \begin{cases} 1, & x = y \\ 0, & w\ p.p. \end{cases}$ jest prymitywnie rekurencyjna. A więc też cały kwantyfikator jest prymitywnie rekurencyjny na podstawie powyższego twierdzenia.

Wykazując prymitywną rekurencyjność funkcji, które zadane są na przedziałach zawsze warto rozważyć, które z podejść będzie dla nas najlepsze:

- Możemy próbować wykazać przy pomocy najprostszych funkcji prymitywnie rekurencyjnych jak to robiliśmy dla funkcji $z(x)$ oraz $pos(x)$, ale takie podejście jest zazwyczaj dobre tylko dla prostych funkcji
- Możemy wykazać to korzystając z funkcji pos oraz z. Np. dla

  $$f(x,y) = \begin{cases} 1, & x \leq y \\ 0, & \text{w p.p.} \end{cases}$$

  zapisywaliśmy jako $f(x, m+1) = z(x \dot{-} S(m))$
- Możemy w końcu najpierw zapisać naszą funkcję jako relacje prymitywnie rekurencyjną, a następnie skorzystać z 'przeszukiwania' na podstawie kwantyfikatorów i wówczas to, że nasze wyrażenie pod kwantyfikatorem jest złożeniem funkcji prymitywnie rekurencyjnych daje nam prymitywnie rekurencyjną funkcje dla całego kwantyfikatora

3.6. Zadania do domu

Na wstępnie sobie powiedzmy, że te zadanka pochodzą prosto z książki Cutlanda. Fragmenty rozwiązań można znaleźć pod `https://github.com/blargoner/math-computability-cutland/blob/master/exercises.pdf`, Rozdział 4.16

Zadanie

1. ✓Wykazać, że zbiór liczb parzystych jest prymitywnie rekurencyjny.
2. ✓Wykazać, że zbiór liczb pierwszych jest prymitywnie rekurencyjny.
3. ✓Pokazać, że dowolny wielomian jest prymitywnie rekurencyjny
4. Pokazać, że $\lceil \sqrt{x} \rceil$ jest prymitywnie rekurencyjna (sufit).
5. Pokazać, że $NWW(x,y)$ jest prymitywnie rekurencyjna.
6. Pokazać, że $NWD(x,y)$ jest prymitywnie rekurencyjna.
7. Pokazać, że funkcja Eulera jest prymitywnie rekurencyjna.

1. ✓Zbiór liczb parzystych zapiszemy jako: $\{x : x = 2a, a \in N\}$

   Zapiszemy funkcję charakterystyczną takiej relacji:

   $$f(x,a) = \begin{cases} 1, \text{ gdy } x = 2a \\ 0, \text{ w p.p.} \end{cases}$$

   Dla ustalonej liczby x, możemy przeszukiwać wszystkie liczby mniejsze od $x+1$ takie aby znaleźć taką liczbę a dla której $x = 2a$, co zapiszemy jako:

   $$\exists_{a<x+1} f(x,a) = \exists_{a<x+1}(x = 2a) = pos(\sum_{a<x+1} f(x,a)) = pos(f(x,0) + f(x,1) + ... + f(x,x))$$

   A nasza funkcja f jest prymitywnie rekurencyjna, bo $x = 2a$ można zapisać jako $RownaSie(x, Mnozenie(2,a))$.

2. ✓Liczby pierwsze dzielą się tylko przez samą siebie i przez 1, mają więc dokładnie dwa dzielniki. My pokazaliśmy już na tych zajęciach, że funkcja $a|b$ jest prymitywnie rekurencyjna i zapisaliśmy ją za pomocą funkcji charakterystycznej. Funkcja ta zwraca 1 jest gdy a dzieli b i 0 w przeciwnym przypadku.

   Liczbę dzielników danej liczby x możemy więc znaleźć korzystając z funkcji:

   $$g(a,b) = \sum_{a<b+1} f(a,b) = \sum_{a<b+1} a|b$$

   Pokazaliśmy już, że funkcja charakterystyczna dla $a|b$ jest prymitywnie rekurencyjna, a także na początku tych zajęć pokazaliśmy, że skończona suma funkcji prymitywnie rekurencyjnych jest prymitywnie rekurencyjna. Zatem $g(a,b)$ musi być prymitywnie rekurencyjna.

   Liczby pierwsza mają dokładnie dwa dzielniki, a więc przyrównujemy liczbę dzielników do 2, co daje nam funkcję charakterystyczną dla liczb pierwszych:

   $$\chi(x) = \begin{cases} 1, \text{ gdy } g(x,x) = 2 \\ 0, \text{ w p.p.} \end{cases}$$

   A wiemy, że funkcja równa się jest prymitywnie rekurencyjna więc ostatecznie mamy funkcję prymitywnie rekurencyjną.

3. ✓Dla wielomianów można spróbować skorzystać z ograniczonej sumy, wielomian stopnia n zapiszemy jako:

   $$f(n) = \sum_{t<n+1} a_t x^t$$

   Fragment $a_t x^t$ jest prymitywnie rekurencyjny, bo to $Mnozenie(a_t, Potegowanie(x,t))$. Suma ograniczona dla f.p.rek. jest funkcją prymitywnie rekurencyjną.

   UWAGA: Kolejne zadania wymagają znajomości operatora μ, są rozwiązane w omówieniu następnych ćwiczeń:

4. To się okazało, że pokazać należy przy pomocy operatora μ, który jest dopiero na następnych zajęciach...

5. To również przy pomocy μ...
6. A to przy pomocy NWD, które z kolei jest przy pomocy μ...
7. A to przy pomocy NWW, które jest przy pomocy NWD, które jest przy pomocy μ... :)

4 Ćwiczenia odwołane 16.03.2020

5 Ćwiczenia 5 (K1) - 23.03.2020 - funkcje częściowo rekurencyjne

Do tej pory zdefiniowaliśmy sobie funkcję prymitywnie rekurencyjne. Wiemy, że takie funkcje możemy uzyskać z kilku podstawowych funkcji, które są prymitywnie rekurencyjne oraz dodatkowo stosując operatory złożenia oraz prymitywnej rekurencji, pokazywaliśmy to na ćwiczeniach numer 2.

Okazuje się, że funkcje prymitywnie rekurencyjne nie wyrażają wszystkich funkcji obliczalnych. Aby móc zdefiniować wszystkie funkcje obliczalne musimy też wziąć pod uwagę dodatkowo tzw. funkcje częściowo rekurencyjne. Funkcje częściowo rekurencyjne zawierają w sobie funkcje prymitywnie rekurencyjne i stanowią zbiór wszystkich funkcji obliczalnych w zbiorze liczb naturalnych.

Dobra wiadomość jest taka, że aby uzyskać funkcje częściowo rekurencyjne wystarczy nam jedna dodatkowa operacja nazywana minimalizacją. Zanim jednak poznamy ten operator, powiedzmy sobie czym są funkcje częściowe.

5.1. Funkcja Akermana

Skąd tutaj jakaś funkcja Akermana ? M.A. na zajęciach ją rozpisywał i obliczał jakąś jej wartość. Chodzi o to, że na przykładzie tej funkcji można pokazać, że same funkcje prymitywnie rekurencyjne to za mało i musimy wprowadzić operator μ. Ale sama funkcja nie jest dla nas ważna i potrzebna do rozważań.

5.2. Funkcja całkowita vs częściowa

- Funkcja całkowita (total funciton) to taka dla której możemy obliczyć wartość dla wszystkich liczb naturalnych. Np. mnożenie - jakiekolwiek dwie liczby naturalne podamy na wejściu to zawsze jesteśmy w stanie obliczyć wynik. Takie funkcje oznaczamy zazwyczaj w taki sposób:

$$f : X^n \to X$$

- Funkcja częściowa (partial function) to taka funkcja, dla której dla pewnych argumentów nie możemy obliczyć jej wartości. Np. dzielenie - jeśli dzielimy jakąkolwiek liczbę x przez 0 to wynik funkcji nie jest określony. Funkcje częściowe zazwyczaj oznaczamy w taki sposób:

$$f : X^n \rightharpoonup X$$

5.3. Operator μ - minimalizacja (przeszukiwanie)

W poprzednich rozdziałach przy pomocy operatorów złożenia i prymitywnej rekurencji mogliśmy definiować tylko funkcje całkowite. Dodanie operatora minimalizacji do tych operatorów powoduje, że możemy definiować również funkcje częściowo rekurencyjne, a zatem mamy już narzędzia do definiowania wszystkich funkcji obliczalnych. Funkcje częściowo rekurencyjne to po prostu funkcje do których definiowania używamy dodatkowo operatora minimalizacji μ. Operator ten na zajęciach był nazywany też czasem przeszukiwaniem.

5.3.1. Działanie operatora μ

Operator ten stosujemy do danej funkcji f takiej, że:

$$f : N^{n+1} \rightharpoonup N$$

Okej, mamy zwykłą funkcję wektora liczb naturalnych w liczbę naturalną. Operator μ zastosowany do takiej funkcji tworzy zaś funkcję postaci:

$$\mu f : N^n \rightharpoonup N$$

Jak widzimy wiele się nie różnią, poza tym, że ta ma o jeden argument mniej. Co się stało z ostatnim argumentem ? Operator μ zakłada, że pierwsze n argumentów jest ustalone, a ostatni z nich (czyli n+1-szy) jest przeszukiwany od 0 w górę w poszukiwaniu pierwszej (najniższej) takiej wartości tego argumentu aby funkcja f miała wartość 0:

$$f(\underbrace{x_1, x_2, \ldots, x_n}_{ustalone}, x)$$

μ szuka w następujący sposób:

$f(x_1, x_2, \ldots, x_n, 0) = y_0$
$f(x_1, x_2, \ldots, x_n, 1) = y_1$
$f(x_1, x_2, \ldots, x_n, 2) = y_2$
...
$f(x_1, x_2, \ldots, x_n, x) = y = 0$
znaleziono najnizsza wartosc dla ktorej jest 0 ! zwroc x

Bardziej formalnie możemy zapisać operator μ jako:

$$\mu f(x_1, x_2, \ldots x_n, x) = \mu f(\bar{\boldsymbol{x}}, x) = \mu(x)[f(\bar{\boldsymbol{x}}, x) = 0] = \begin{cases} \text{najmniejsza wartosc x dla ktorej } f(\bar{\boldsymbol{x}}, x) = 0 \text{ o ile istnieje takie x} \\ \text{niezdefiniowana jesli taki x nie istnieje} \end{cases}$$

Czyli mając funkcję jak na rysunku poniżej, to nasz operator μ znajdzie wartość zaznaczonego x, czyli pierwszego (najmniejszego) dla którego wartość funkcji wynosi 0:

Gdyby nie było takiego argumentu dla którego funkcja ma wartość 0, to wtedy funkcja operatora μ jest nieokreślona.

5.4. Prosty przykład

Zacznijmy od bardzo prostego przykładu aby złapać ogólną ideę tego okrutnego operatora. Jako funkcję f mamy daną funkcję projekcji, która z dwóch argumentów wybiera drugi:

$$f(x_1, x_2) = p_2^2(x_1, x_2)$$

Stosujemy do tej funkcji operator μ:

$$\mu f(x_1, x_2) = \mu(x) f(x_1, x)$$

Bierzemy więc jakieś ustalone x_1, np. 9 i przeszukujemy różne wartości dla x idąc od zera:

$$p_2^2(8,0) = 0$$
$$p_2^2(8,1) = 1$$
$$p_2^2(8,2) = 2$$

W tym przypadku odpowiedź mamy od razu na początku i nawet nie musimy szukać dalej, naszą odpowiedzią jest wartość $x = 0$.

5.5. Bardziej praktyczne przykłady + c.d. zadań domowych

5.5.1. Dzielenie całkowite

Załóżmy, że chcemy pokazać, że dzielenie całkowite jest częściowo rekurencyjne. Dzielenie całkowite to funkcja taka, że, np.:

$$10 \; div \; 4 = 2r.2$$
$$15 \; div \; 4 = 3r.3$$

Przy czym reszta r nas nie interesuje, istotny jest sam wynik. Jak zapisać taką funkcję przy pomocy operatora μ ?

Na początek rozpisujemy sobie, że funkcja ta działa tak naprawdę w następujący sposób:

$$x \; div \; y = q \cdot y + r$$

gdzie $r < y$. Aby znaleźć wynik szukamy więc największego q dla którego $q \cdot y \leq x$. Nie jest to jednak minimalizacja (szukamy największego q) więc trudno zapisać to przy pomocy operatora μ. Możemy przeformułować problem na taki:

Aby znaleźć wynik szukamy najmniejszego q takiego, że $(q+1)y > x$. Jak to dopasować do naszego μ, które szuka zerowej wartości funkcji ? Możemy stworzyć funkcję charakterystyczną:

$$f(x,y,q) = \begin{cases} 1, \; (q+1)y \leq x \\ 0, \; (q+1)t > x \end{cases}$$

teraz szukamy najmniejszego q takiego aby funkcja f była równa 0, co zapiszemy:

$$\mu f(x,y,q)$$

Taka funkcja da nam najmniejsze q dla którego funkcja f jest równa 0.

5.5.2. Pierwiastek z sufitem lub podłogą

Funkcja $\lceil \sqrt{x} \rceil$ ma dawać najmniejszą liczbę całkowitą większą od $\sqrt{x}$. Czyli dla:

$$\sqrt{16} = 4 \to 4$$
$$\sqrt{15} = 3.xxx \to 4$$
$$\sqrt{14} = 3.xxx \to 4$$
$$...$$
$$\sqrt{9} = 3 \to 3$$
$$\sqrt{8} = 2.xxx \to 3$$

Jeśli mamy więc liczbę x, to chcemy znaleźć taką liczbę a, że $a \cdot a$ da nam najmniejszą liczbę większą od x. Możemy więc zapisać funkcję charakterystyczną:

$$f(x,a) = \begin{cases} 1, \ a \cdot a < x \\ 0, \ a \cdot a \geq x \end{cases}$$

Teraz wystarczy, że znajdziemy najmniejsze a, czyli nasza funkcja jest ostatecznie postaci:

$$\mu f(x,a)$$

Np. dla $x = 10$

$$f(10,0) = 1$$
$$f(10,1) = 1$$
$$f(10,2) = 1$$
$$f(10,3) = 1$$
$$f(10,4) = 0$$

Stąd wynikiem jest 4.

UWAGA: Gdybyśmy zamiast funkcji sufitu mieli funkcję podłogi, to wtedy robimy tylko dodatkowo w naszej końcowej funkcji odejmowanie jedynki i mamy:

$$\mu f(x,a) \dot{-} 1$$

5.5.3. Najmniejsza wspólna wielokrotność NWW

Dla liczb a oraz b to liczba, która dzieli zarówno a jak i b i, która jest mniejsza od $a \cdot b$. Zapiszmy więc funkcję charakterystyczną:

$$f(a,b,z) = \begin{cases} 0, \ z < ab \ \wedge \ a|z \ \wedge \ b|z \\ 1, \ w \ p.p. \end{cases}$$

Możemy się zastanawiać czy jest ona prymitywnie rekurencyjna. Oczywiście jest, bo pokazaliśmy, że $a|b$ jest p.rek. oraz, że $z < y$ jest p.rek., pokazaliśmy także, że iloczyn relacji prymitywnie rekurencyjnych jest prymitywnie rekurencyjny. A więc teraz wystarczy, że znajdziemy najmniejsze z, które spełniać będzie naszą funkcję f:

$$\mu f(a,b,z)$$

5.5.4. Największy wspólny dzielnik NWD

W tym przypadku korzystamy z zależności:

$$NWD(a,b) = \frac{ab}{NWW(a,b)}$$

Iloraz jest prymitywnie rekurencyjny (czego nie pokazaliśmy, ale tak jest). Więc w zasadzie mamy złożenie funkcji prymitywnie rekurencyjnych, bo pokazaliśmy, że NWW jest p.rek.

5.6. Zadania do domu TODO

- Podać przykład funkcji $f(\boldsymbol{x})$ częściowej (zdefiniowanej nie na całej dziedzinie liczb naturalnych) takiej, że $h(x)$ jest całkowita, gdzie $h(x) = \mu(x)f(\boldsymbol{x})$
- Funkcja f jest k-zmiennych, pokazać, że jest rekurencyjna z użyciem operatora μ - nie wiem o co chodzi w tym poleceniu...
- Funkcja $g(\boldsymbol{x})$ jest funkcją niemalejąca, nieograniczoną i prymitywnie rekurencyjną. Pokazać, że obraz tej funkcji jest zwyczajnie rekurencyjny, tzn. że funkcję charakterystyczną tego obrazu można znaleźć przy użyciu tych reguł - też nie wiem o co chodzi...

6 Ćwiczenia 6 (K2) - 30.03.2020 - rachunek lambda, składnia, zmienne wolne i związane, konwencje, redukcje

Na tych ćwiczeniach rozważamy rachunek lambda. Rachunek lambda stanowi model obliczalności, pomimo, że oryginalnie nie został zaprojektowany do tego celu. Jest on jednak równoważny innym modelom obliczalności takim jak funkcje prymitywnie rekurencyjne czy też maszyna Turinga.

Każda funkcja, która jest obliczalna może zostać wyrażona i obliczona przy pomocy tego rachunku.

6.1. Składnia rachunku lambda

Rachunek lambda stosuje tylko dwa 'słowa kluczowe' są to:

- λ
- .

W rachunku lambda operujemy pojęciem zmiennej oraz termu.

- Zmienne oznaczamy małymi literami np. x, y, z, możemy o nich myśleć jak o zmiennych w funkcjach, np. $f(x, y, z) = ...$.
- Każda zmienna jest termem, a więc np. samo x jest termem, czy też samo y jest termem (tzw. nazwa)
- Jeżeli M jest termem, to termem jest też każde wyrażenie postaci $\lambda x.M$ (tzw. funkcja lub abstrakcja)
- Jeżeli M oraz N są termami, to również MN jest termem (tzw. aplikacja)

Gramatyka dla rachunku lambda (przez expression rozumiemy term, term może być nazwą, funkcją lub aplikacją):

$$\begin{aligned} \text{<expression>} &:= \text{<name>} \mid \text{<function>} \mid \text{<application>} \\ \text{<function>} &:= \lambda\ \text{<name>}.\text{<expression>} \\ \text{<application>} &:= \text{<expression><expression>} \end{aligned}$$

Rachunek lambda pozwala na wygodne zapisywanie funkcji bez podawania jest nazwy. Dla przykładu:

- $f(x) = x + 3$ zapiszemy jako $\lambda x.(x + 3)$
- Aplikacja w rachunku lambda jest odpowiednikiem podstawienia za zmienną pewnej wartości, $f(2)$ zapiszemy jako $(\lambda x.(x + 3))\ (2)$, jest to wyrażenie postaci MN gdzie M oraz N są termami

Dla ścisłej kompletności tego opisu powinniśmy jeszcze dodać do rachunku lambda stałe, takie jak np. 2,3,4... oczywiście każda stała jest również termem.

6.2. Konwencje stosowane w rachunku lambda

6.2.1. Konwencja 1

W rachunku lambda możliwe jest wstawianie nawiasów, które obejmują termy. Dla przykładu możemy zapisać, że $(\lambda x.x)y$.

Chcemy jednak unikać wprowadzania do wyrażeń ogromnej liczby nawiasów dlatego przyjmuje się, że jeśli w wyrażeniu nie ma żadnych nawiasów to aplikacja funkcji wiąże od strony lewej do prawej. Dla przykładu:

$MNQR$ jest równoważne $(((MN)Q)R)$

A zatem przykładowo

$$(\lambda t.t)(\lambda c.cx)(\lambda g.g) = (\ ((\lambda t.t)(\lambda c.cx))\ (\lambda g.g))$$

6.2.2. Konwencja 2

Jeżeli po kropce nie wstawiono nawiasu, to przyjmuje się, że wyrażenie lambda ma największy możliwy zasięg aż do wystąpienia kolejnego znaku λ. Dla przykładu:

$\lambda x.MNP$ jest równoważne $\lambda x.(MNP)$ - o ile MNP nie zawierają znaku lambda

Dla przykładu

$$\lambda x.xyz\lambda tc.ct = (\lambda x.xyz)(\lambda t.\lambda c.ct)$$

6.2.3. Konwencja 3

Zagnieżdżone wyrażenia lambda zapisujemy poprzedzając kropkę kilkoma zmiennymi. Dla przykładu

$\lambda x.\lambda y.\lambda z.M$ jest równoważne $\lambda xyz.M$

6.3. Ewaluacja aplikacji funkcji

Funkcje mogą być aplikowane do wyrażeń, jak już wiemy taką aplikację zapisujemy w postaci termu MN. Jako przykład weźmy

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

W tym przykładzie aplikujemy term y do termu (funkcji) $(\lambda x.x)$, wynik zapisujemy w ten sposób:

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

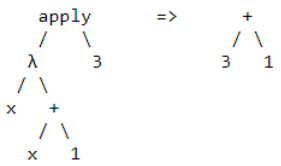
Wynikiem tego wyrażenia jest więc term y. W zapisie $[y/x]$ (czyt. *y zastępuje x*) chodzi o to że wszystkie wystąpienia x zastępujemy przez y.

6.3.1. Ewaluacja jako drzewo syntaktyczne

Tego typu podstawiania (aplikacje) często wygodnie jest zapisywać w postaci drzew syntaktycznych. Dla przykładu weźmy takie wyrażenie:

$$(\lambda x.x + 1)3$$

Mamy tu aplikacje termu 3 do funkcji. Możemy to przedstawić w postaci drzewa:

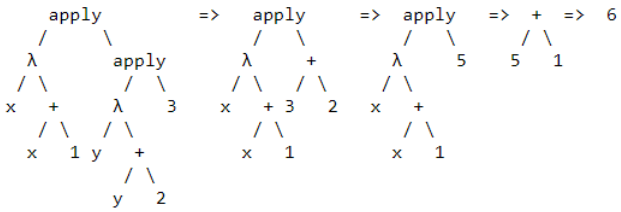


O co w tym chodzi ? apply określa aplikacje. Aplikujemy 3 do wyrażenia lambda. Lewa gałąź reprezentuje wyrażenie lambda. Lewe poddrzewo od znaku λ reprezentuje argumenty wyrażenia lambda (w tym przypadku x), a prawe poddrzewo to ciało tego wyrażenia. Do ciała podstawiamy część, która jest w prawej części drzewa, czyli trzy i przekształcamy drzewo zgodnie z podstawieniem.

Kolejny przykład:

$$(\lambda x.x + 1)((\lambda y.y + 2)3)$$

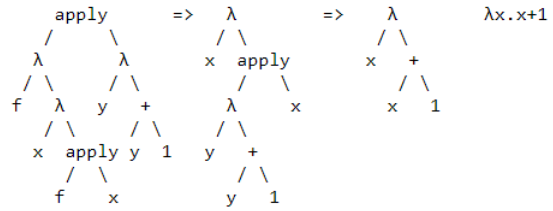
Drzewo dla tego przykładu:



Troszkę trudniejszy przykład:

$$(\lambda f.\lambda x.fx)(\lambda y.y + 1)$$

Drzewo:



Zauważmy na wstępie, że w tym przypadku wynikiem jest funkcja. Tak będzie w większości przypadków.

W tym przykładzie mamy dwie aplikacje. Pierwsza z nich to po prostu fx, oznacza, że za f podstawiamy x. Druga z aplikacji to aplikacja wyrażenia $(\lambda y.y + 1)$ do wyrażenia $(\lambda f.\lambda x.x)$. Z pierwszą aplikacją w tym wypadku nic nie możemy zrobić, przechodzimy więc od razu do aplikacji 'top-level'.

$$(\lambda f.\lambda x.fx)(\lambda y.y + 1) = \lambda x.(\lambda y.y + 1)x = \lambda x.x + 1$$

6.3.2. Problem z ewaluacją przez naiwne podstawianie

Do tej pory realizowaliśmy aplikację przez podstawienie za wszystkie zmienne innej zmiennej. Spójrzmy jednak na przykład, który pokaże, że takie podejście nie zawsze daje prawidłowy rezultat i w związku z tym musimy wprowadzić inne podejście.

$$(\lambda x.(x + ((\lambda x + 1)3)))2$$

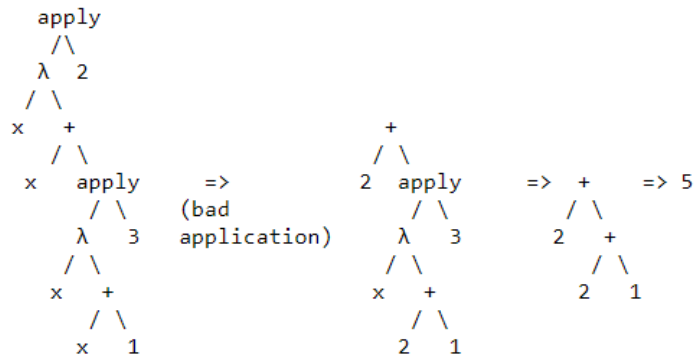
Wyrażenie to powinno redukować się do 6. Jeśli najpierw wykonamy podstawienie:

$$(\lambda x + 1)3 = 4$$

To następnie nasze wyrażenie będzie postaci:

$$(\lambda(x.x + 4))2 = 6$$

W przypadku gdybyśmy jednak postępowali jak dotychczas i zaczęli od podstawienia za wszystkie zmienne otrzymalibyśmy 5:



Problem ten nazywany jest często *name clash* lub *capture*.

Aby temu zapobiec musimy zmienić dotychczas stosowane podejście, które polegało na tym, że zamienialiśmy wszystkie wystąpienia zmiennych na inne wyrażenie. Aby zastosować nowe podejście, najpierw musimy powiedzieć sobie czym są zmienne wolne i związane.

6.4. Zmienne wolne i związane

Najpierw podejdźmy do sprawy intuicyjnie, a później podamy bardziej formalną definicję. Rozważmy term:

$$\lambda x.xy$$

W tym termie zmienna x jest zmienną związaną, ponieważ jej wystąpienie jest poprzedzone przez λx. Zmienna y jest w tym termie zmienną wolną, ponieważ ta zmienna nie jest zawarta 'przed kropką'.

Rozważmy bardziej skomplikowane wyrażenie:

$$(\lambda x.x)(\lambda y.yx)$$

W pierwszym termie zmienna x jest związana. W drugim termie zmienna y jest związana, a zmienna x jest wolna. Zauważmy, że zmienna x z drugiego termu jest zupełnie niezależna od tej z pierwszego termu.

A teraz bardziej formalnie.

6.4.1. Zmienne wolne

Zmienna x jest wolna jeśli zachodzi jeden z trzech warunków

1. x jest wolny w x
2. x jest wolny w $\lambda y.M$ wtedy gdy $x \neq y$ oraz x jest wolny w M
3. x jest wolny w MN jeżeli x wolny w M lub x wolny w N

6.4.2. Zmienna związane

Zmienna x jest związana jeśli zachodzi jeden z dwóch warunków

1. x jest związany w $\lambda y.M$ jeżeli $x = y$ lub x jest związany w M
2. x jest związany w MN jeśli x jest związany w M lub x jest związany w N

6.4.3. Wolna i związana ?!

UWAGA: Zmienna może być w termie zarówna związana jak i wolna!
Spójrzmy na przykład

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

W pierwszym podwyrażeniu:

- x jest związany z warunku 1.
- y jest wolny z warunku 2. 3. i ostatecznie 1.

W drugim podwyrażeniu:

- y jest związany z warunku 1.

W całym wyrażeniu y występuje więc jako zmienna związana, ale też wolna z warunków 3 oraz 2.

Wiemy już czym są zmienne wolne i związane. Ta wiedza pozwoli nam na poprawną eweluację każdego wyrażenia lambda przy pomocy $\alpha - konwersji$ oraz $\beta - redukcji$.

6.5. α-konwersja

Sposobem na rozwiązanie problemu *capture* jest zastosowanie α-konwersji. Idea tej konwersji polega na tym, że jeśli wiemy, że dwie zmienne o tej samej nazwie są od siebie niezależne, to możemy zastąpić jedną z nich inną nazwą. Podejście to bazuje na założeniu, że nazwy zmiennych w wyrażeniu nie mają znaczenia więc możemy je zamienić tak aby uniknąć *capture*.

- α-konwersja jest stosowana do wyrażeń typu $\lambda x.M$. W jej wyniku zamieniamy wszystkie zmienne x, które są wolne w M na inną zmienną z, której nie było do tej pory w M. Natomiast $\lambda x.M$ zostaje zamienione na $\lambda z.M$.

α-konwersję zapisujemy jako:

$$\lambda x.M \xrightarrow{\alpha} \lambda y.M[x/y]$$

Zapis $[x/y]$ oznacza: *zamień wszystkie wolne wystąpienia zmiennej x w M przez zmienną y*.

Przykład

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

Zauważmy, że w powyższym przykładzie M stanowi $(\lambda x.x)$, a w nim nie ma żadnej zmiennej wolnej x.

Inny przykład

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

Pamiętajmy, że zgodnie z konwencją drugą ten zapis tak naprawdę oznacza:

$$\lambda x.\lambda y.(x + y)$$

W tym przypadku M stanowi $\lambda y.x + y$. Zastosujemy α-konwersję dla zmiennej x. Do konwersji zastosujemy zmienną z, ponieważ nie ma jej w M.

$$\lambda x.\lambda y.x + y \xrightarrow{\alpha} \lambda z.\lambda y.z + y[x/z]$$

6.6. β-redukcja

α-konwersja zamienia nazwy zmiennych w wyrażeniu, a β-redukcja odpowiedzialna jest za redukcję wyrażenia zgodnie z aplikacją. β-redukcje realizujemy według poniższej formuły:

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

Znak $[x/N]$ oznacza dokładnie to samo co w α-konwersji, a więc zamianę wolnych zmiennych x w M przez N.

Teraz jesteśmy gotowi aby móc w poprawny sposób przeprowadzić aplikacje w rachunku lambda. Aplikację realizujemy przez stosowanie α-konwersji oraz β-redukcji aż do maksymalnego uproszczenia wyrażenia. Spójrzmy na przykłady:

1.

$$(\lambda x.x)y \xrightarrow{\beta} x[x/y] = y$$

2.
$$\underbrace{(\lambda x.x)}_{M}\underbrace{(\lambda x.x)}_{N} \xrightarrow{\alpha_N} = (\lambda x.x)(\lambda z.z) \xrightarrow{\beta} x[x/\lambda z.z] = \lambda z.z \xrightarrow{\alpha} \lambda x.x$$

3.
$$(\lambda x.\underbrace{\lambda y.xyy}_{M})y \xrightarrow{\alpha_M} (\lambda x.\lambda v.xvv)y \xrightarrow{\beta} \lambda v.xvv[x/y] = \lambda v.yvv$$

6.7. η-redukcja

To specjalny przypadek β-redukcji taki, że mając dane:

$$\lambda x.Mx$$

Jeżeli w M nie ma zmiennych wolnych, tzn. M jest termem domkniętym, to wówczas całe wyrażenie możemy przepisać jako:

$$\lambda x.Mx \xrightarrow{\eta} M$$

6.8. Rozwiązanie problemu *capture*

Dlaczego potrzebna jest nam zarówno α jak i β konwersja ? β-konwersja redukuje wyrażenie, zamienia je na inną postać. α-konwersja pozwala na uniknięcie problemu *capture* podczas wykonywania przekształceń.

Skąd mamy widzieć czy na danym etapie przekształceń zastosować β czy α konwersję ? Możemy stosować β-redukcję, o ile nie wprowadzi to do naszego wyrażenia problemu z *capture*.

Ogólna zasada mówi, że mając dane wyrażenie:

$$\lambda x.MN$$

- β-redukcję możemy zastosować jeśli nie ma zmiennych, które są wolne w N, a związane w M. Równoważnie możemy powiedzieć, że warunkiem jest aby każda zmienna wolna w N pozostawała wolna w M, czyli jeszcze inaczej ujmując, jeżeli zmienna jest wolna w N, a związana w M, to należy wykonać α-konwersje
- jeśli powyższy warunek nie jest spełniony, to musimy zastosować α-konwersję, tak aby zamienić nazwy zmiennych, które są wolne w N, ale związane w M, zmiany tej dokonujemy tylko w wyrażeniu M, N pozostawimy tak jak było

6.9. Zadania do domu

Zadanie

Wypisać, które zmienne w wyrażeniach są związane, a które wolne. Dodać nawiasy do wyrażeń.

- $\lambda s.sz\lambda q.sq$
- $(\lambda s.sz)\lambda q.w\lambda w.wqzs$
- $(\lambda s.s)(\lambda q.qs)$

1. ✓ $\lambda s.sz\lambda q.sq = \lambda s.((sz)(\lambda q.(sq)))$
   - s - związana, bo występuje przed kropką w lambdzie
   - q - związana w wyrażeniu $\lambda q.sq$, bo występuje przed kropką w lambdzie, a więc też związana w całym wyrażeniu, bo jest postaci $\lambda s.M$, gdzie q jest związane w M
   - z - wolna, bo jest wolna w M, a całe wyrażenie jest postaci $\lambda s.M$
2. UWAGA: To jest chyba źle (na obrazku poniżej dobre rozwiązanie) $(\lambda s.sz)\lambda q.w\lambda w.wqzs = (\lambda s.sz)((\lambda q.w)(\lambda w.wqzs))$
   Mamy tu wyrażenie postaci $(\lambda s.M)((\lambda q.N)(\lambda w.P))$
   - w - związana i wolna, bo wolna w N, związana w P więc w całym wyrażeniu wolna i związana
   - q - związana i wolna, bo związana w N, wolna w P, więc w całym wyrażeniu wolna i związana
   - z - wolna, bo wolna w M, wolna w P
   - s - wolna i związana, bo wolna w P, ale związana w M

```
λ > ((lambda s.s z) lambda q .w lambda w.w q z s)
α > ((λs.(s z))(λq.(w(λw.(((w q)z)s)))))
ε >       unbound variable: z rename as x'0
>         unbound variable: w rename as x'1
>         unbound variable: z rename as x'2
>         unbound variable: s rename as x'3
((λs.(s [x'0] ))(λq.( [x'1] (λw.(((w q) [x'2] ) [x'3] )))))
β > ((λq.(x'1(λw.(((w q)x'2)x'3))))x'0)
β > (x'1(λw.(((w x'0)x'2)x'3)))
```

3. ✓ $(\lambda s.s)(\lambda q.qs)$
   - s - związana i wolna
   - q - związana

7 Ćwiczenia 7 (K2) - 06.04.2020 - r. lambda, redukcje c.d., funkcja if-then-else

Na początku tych ćwiczeń pokazaliśmy dwa przykłady dla których brak użycia α-konwersji powodował, że wynik końcowy był nieprawidłowy. Przykłady:

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

Zacznijmy od drugiego:

- Bez użycia α-konwersji (ŹLE!):

$$(\lambda x.\lambda y.xyy)y \xrightarrow{\beta} \lambda y.yyy$$

- Z użyciem α-konwersji (DOBRZE!)

$$(\lambda x.\lambda y.xyy)y \xrightarrow{\alpha} (\lambda x.\lambda v.xvv)y \xrightarrow{\beta} \lambda v.yvv$$

Widzimy, że już w wyrażeniu startowym warunek na β-redukcje nie jest spełniony. W wyrażeniu M mamy bowiem zmienną związaną y, która jest wolna w wyrażeniu N.

Teraz pierwszy z przykładów:

- Bez użycia α-konwersji (ŹLE!):

$$(\lambda x.\lambda y.(xy))(\lambda x.\lambda y.(xy)) \xrightarrow{\beta} \lambda y.(xy)[x/(\lambda x.\lambda y.(xy))] \xrightarrow{\beta} \lambda y.((\lambda x.\lambda y.(xy))y) \xrightarrow{\beta} \lambda y.\lambda y.(yy)$$

- Z użyciem α-konwersji (DOBRZE!)

$$(\lambda x.\lambda y.(xy))(\lambda x.\lambda y.(xy)) \xrightarrow{\beta} \lambda y.(xy)[x/(\lambda x.\lambda y.(xy))] =$$
$$\lambda y.((\lambda x.\lambda y.(xy))y) \xrightarrow{\alpha} \lambda y.((\lambda x.\lambda v.(xv))y) \xrightarrow{\beta} \lambda y.(\lambda v.yv)$$

7.1. Postacie termów

- **Postać normalna** - mówimy, że term jest w postaci normalnej jeśli nie da się wykonać żadnej β-redukcji
- **Term domknięty** - mówimy, że term jest domknięty jeżeli nie zawiera zmiennych wolnych

7.2. Zastosowanie rachunku lambda do zdań logicznych

Przy pomocy rachunku lambda można wyrażać funkcje logiczne takie jak AND, OR itd. W tym celu definiujemy sobie na początek dwie lambdy, jedna z nich oznacza prawdę, a druga fałsz:

$$T = \lambda x.\lambda y.x$$
$$F = \lambda x.\lambda y.y$$

Term T będzie oznaczał prawdę, a term F fałsz. T to po prostu funkcja, która przyjmuje dwa argumenty i zwraca pierwszy z nich, a F, to funkcja, która przyjmuje dwa argumenty i zwraca drugi z nich.

Korzystając z tych funkcji spróbujemy zrobić funkcję *if-then-else*

7.2.1. if-then-else

Funkcja taka ma działać w następujący sposób:

$$\lambda(fun)Tab = a$$
$$\lambda(fun)Fab = b$$

Takie warunki spełnia funkcja:

$$\lambda c.\lambda x.\lambda y.(cxy)$$

Pamiętając, że

$$T = \lambda x.\lambda y.x = \lambda m.\lambda n.m$$

Sprawdźmy:

$$(\lambda c.\lambda x.\lambda y.(cxy))Tab \xrightarrow{\beta} \lambda x.\lambda y.(cxy)[c/\lambda m.\lambda n.m]ab =$$
$$(\lambda x.\lambda y.((\lambda m.\lambda n.m)xy))ab \xrightarrow{\beta} \lambda x.\lambda y.((\lambda n.x)y))ab \xrightarrow{\beta} (\lambda x.\lambda y.x)ab \xrightarrow{\beta} (\lambda y.a)b \xrightarrow{\beta} a$$

7.3. Zadania do domu

1. ✓$(\lambda z.z)(\lambda z.zz)(\lambda z.zq)$
2. ✓$(\lambda xyz.zyx)aa(\lambda pq.q)$
3. ✓$(\lambda s.\lambda q.sqq)(\lambda a.a)b$
4. $(\lambda y.yyy)(\ (\lambda ab.a)\ (\lambda x.x(\lambda x(yz.xz(yz\lambda y\lambda xyz.xz(yz))))$

1. ✓$\underbrace{(\lambda z.z)}_{M}\underbrace{(\lambda z.zz)}_{N}\underbrace{(\lambda z.zq)}_{Q} = ((MN)Q) \xrightarrow{\beta_{MN}} (\lambda z.zz)(\lambda z.zq) \xrightarrow{\alpha_{[z/v]}} (\lambda v.vv)(\lambda z.zq) \xrightarrow{\beta}$
   $(\lambda z.zq)(\lambda z.zq) \xrightarrow{\alpha_{[z/t]}} (\lambda t.tq)(\lambda z.zq) \xrightarrow{\beta} (z.zq)q \xrightarrow{\beta} qq$
2. ✓$\underbrace{(\lambda xyz.zyx)}_{M}\underbrace{a}_{N}\underbrace{a}_{Q}\underbrace{(\lambda pq.q)}_{R} = (((MN)Q)R) \xrightarrow{\beta_{MN}} (\lambda yz.zya)a(\lambda pq.q) \xrightarrow{\alpha_{[a/t]}}$
   $(\lambda yz.zyt)a(\lambda pq.q) \xrightarrow{\beta} (\lambda z.((za)t))(\lambda pq.q) \xrightarrow{\beta} ((\lambda pq.q)a)t \xrightarrow{\beta} (\lambda q.q)t \xrightarrow{\beta} t$
3. ✓$(\lambda s.\lambda q.sqq)(\lambda a.a)b \xrightarrow{\beta} (\lambda q.(\lambda a.a)qq)b \xrightarrow{\beta} ((\lambda a.a)b)b \xrightarrow{\beta} bb$

8 Ćwiczenia 8 (K2) - 20.04.2020 - r. lambda, logika, liczby, wyrażenia arytmetyczne

8.1. Definiowanie logiki przez rachunek lambda c.d.

Na ostatnich zajęciach zdefiniowaliśmy sobie funkcję prawda, fałsz oraz if-then-else dla logiki pierwszego rzędu. Teraz dodamy do tego kilka kolejnych funkcji logicznych.

8.1.1. NOT

Najpierw zastanówmy się jak ma działać NOT w kontekście funkcji, które już znamy. NOT ma odwracać wartość logiczną na przeciwną, a więc możemy zapisać, że

$$if\ x\ then\ F\ else\ T$$

Wiemy, że nasza funkcja ma być funkcją jednoargumentową, bo NOT jest jednoargumentowy więc zapiszemy:

$$NOT = \lambda x.(if\ x\ then\ F\ else\ T)$$

Okej, spróbujmy rozwiązań wyrażenie wewnątrz podstawiając znaną nam już funkcje if-then-else:

$$if\ x\ then\ F\ else\ T = (\lambda c.\lambda x.\lambda y(cxy))xFT = xFT$$

a więc ostatecznie nasza funkcja NOT ma postać:

$$NOT = \lambda x.xFT$$

Wszystkie z funkcji, które są w wyrażeniu powyżej już znamy, a więc możemy zapisać:

$$\lambda x.(xFT)$$

8.1.2. AND

Podobnie wyprowadzimy sobie wyrażenie na AND. Skorzystajmy z tego, że AND można zapisać jako:

$$if\ x\ then\ y\ else\ F$$

AND jest funkcją dwóch zmiennych więc zapiszemy, że:

$$\lambda x.\lambda y.(if\ x\ then\ y\ else\ F)$$

Wyrażenie ze 'środka' zapiszemy korzystając ze znanej nam już funkcji if-then-else:

$$(if\ x\ then\ y\ else\ F) = (\lambda c.\lambda x.\lambda y(cxy))xyF = xyF$$

a więc ostatecznie:

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

8.1.3. OR

Analogicznie do powyższych przykładów tworzymy OR: OR możemy utworzyć korzystając z:

$$if\ x\ then\ x\ else\ Ty$$

czyli korzystając z if then else:

$$(\lambda c.\lambda x.\lambda y(cxy))xxTy = xTy$$

i ostatecznie:

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

8.2. Zadania do domu - Część I

1. ✓Zdefiniować funkcje implikacji
2. ✓Zdefiniować funkcje równoważności
3. ✓Zdefiniować funkcje XOR

Rozwiązania:

1. ✓Implikacja to funkcja dwuargumentowa, zauważmy, że jeżeli x jest prawdziwe, to implikacja jest równa y, jeżeli x jest prawdziwe, to implikacja jest zawsze prawdziwa:

   $$if\ x\ then\ y\ else\ T$$

   Stąd:

   $$\lambda x.\lambda y.(if\ x\ then\ y\ else\ T) = \lambda x.\lambda y.((\lambda cpq.(cpq))xyT) = \lambda x.\lambda y.xyT$$

   Sprawdźmy poprawność:

   $$(\lambda x.\lambda y.xyT)TF \xrightarrow{\beta} (\lambda y.TyT)F \xrightarrow{\beta} TFT = (\lambda p.q.p)FT = F$$

   $$(\lambda x.\lambda y.xyT)TT \xrightarrow{\beta} (\lambda y.TyT)T \xrightarrow{\beta} TTT = (\lambda p.q.p)TT = T$$

   $$(\lambda x.\lambda y.xyT)FT \xrightarrow{\beta} (\lambda y.FyT)T \xrightarrow{\beta} FTT = (\lambda p.q.q)TT = T$$

   $$(\lambda x.\lambda y.xyT)FF \xrightarrow{\beta} (\lambda y.FyT)F \xrightarrow{\beta} FFT = (\lambda p.q.q)FT = T$$

2. ✓równoważność jest również dwuargumentowa, znów spróbujemy to jakoś podstępnie napisać przy pomocy if-then-else

   $$\lambda x.\lambda y.(if\ x\ then\ y\ else\ NOTy) = \lambda x.\lambda y.((\lambda cpq.(cpq))xy(NOTy)) =$$
   $$\lambda x.\lambda y.xy(NOTy) = \lambda x.\lambda y.xy((\lambda q.qFT)y) = \lambda x.\lambda y.xy(yFT)$$

3. ✓XOR jest również dwuargumentowy, znów spróbujemy go jakoś podstępnie napisać przy pomocy if-then-else

   $$\lambda x.\lambda y.(if\ x\ then\ NOT\ y\ else\ y) = \lambda x.\lambda y.((\lambda cpq.(cpq))x(NOTy)y) =$$
   $$\lambda x.\lambda y.x(NOTy)y = \lambda x.\lambda y.x((\lambda q.qFT)y)y = \lambda x.\lambda y.x(yFT)y$$

8.3. Definiowanie liczb przy pomocy rachunku lambda

W rachunku lambda możemy wyrażać tylko funkcje, a więc również liczby musimy wyrażać przy pomocy funkcji.

Zastosujemy podejście podobne do tego, które stosowaliśmy w funkcjach prymitywnie rekurencyjnych. Zdefiniujemy funkcję reprezentującą zero, a następnie funkcję następnika. Jedynkę będziemy więc reprezentować jako $succ(zero)$, dwójkę jako $succ(succ(zero))$ itd.

Zero będziemy reprezentować w rachunku lambda jako funkcję równoważną fałszowi, podejście takie jest stosowane np. w języku C++.

$$\bar{0} = \lambda s.\lambda z.z$$

Jedynkę reprezentujemy jako funkcję, która reprezentuje identyczność:

$$\bar{1} = \lambda s.\lambda z.s(z) = (\lambda sz.s)z = \lambda z.z$$

Następnie dwójkę możemy zapisać jako:

$$\bar{2} = \lambda s.\lambda z.s(s(z))$$

Idąc dale tym schematem możemy zdefiniować sobie dowolną liczbę n jako:

$$\bar{n} = \lambda sz.s^n(z)$$
$$gdzie$$
$$s^n = \underbrace{s(s(...(s}_{n}(z))...))$$

8.4. Następnik i przykłady

Na podstawie powyższych rozważań zdefiniujemy sobie funkcję następnika:

$$S \equiv \lambda nsz.(s(nsz))$$

Przykłady:

nastepnik dla 0:

$S\bar{0} = \lambda nsz(s(nsz))\lambda sz.z \xrightarrow{\alpha} \lambda ns'z'.(s'(ns'z'))\lambda sz.z =$
$\lambda s'z'.(s'((\lambda sz.z)s'z')) = \lambda s'z'.(s'(\lambda z.zz')) = \lambda s'z'.s'z' = \bar{1}$

Podobne przykłady z literatury:

$$\mathsf{S0} \equiv (\lambda wyx.y(wyx))(\lambda sz.z)$$

In the body of the first expression we substitute all occurrences of w with $(\lambda sz.z)$ and this yields

$$\lambda yx.y((\lambda sz.z)yx) = \lambda yx.y((\lambda z.z)x) = \lambda yx.y(x) \equiv 1$$

That is, we obtain the representation of the number 1 (remember that variable names are "dummies").

Successor applied to 1 yields:

$$\mathsf{S1} \equiv (\lambda wyx.y(wyx))(\lambda sz.s(z)) = \lambda yx.y((\lambda sz.s(z))yx) = \lambda yx.y(y(x)) \equiv 2$$

Notice that the only purpose of applying the number $(\lambda sz.s(z))$ to the arguments y and x is to "rename" the variables used in the definition of our number.

Sprawdzmy czy
$S\bar{n} = \overline{n+1}$?
$\lambda nsz.(s(nsz))\lambda sz.s^n(z) = \lambda ns'z'.(s'(ns'z'))\lambda sz.s^n(z) = \lambda s'z'.(s'((\lambda sz.s^n(z))s'z')) =$
$\lambda s'z'.s'((\lambda z.s'^n(z))z') = \lambda s'z'.(s'(s'^n(z'))) = \lambda s'z'.s'^{n+1}(z') = \overline{n+1}$

8.5. Operacje arytmetyczne

Zdefiniowaliśmy już liczby, teraz spróbujemy sobie zdefiniować operacje arytmetyczne.

8.5.1. Dodawanie

W rachunku lambda najpierw zapisujemy funkcję, a potem jej argumenty, a więc przykładowo dodawania zapisalibyśmy jako:

$$2 + 3 \xrightarrow{jako} +23$$

Dodawanie z rachunku lambda definiujemy w następujący sposób:

$$\lambda mnsx.(ms(nsx))$$

Spróbujmy zastosować dodawanie do dwóch liczb i sprawdzić czy działa. Mamy dane dwie liczby p oraz q:

$$p = \lambda sz.s^p(z)$$
$$q = \lambda sz.s^q(z)$$

Dodajmy do siebie te liczby:

$$Dpq = \lambda mns'x.(ms'(ns'x))pq = \lambda ns'x(ps'(ns'x))q = \lambda s'x(ps'(qs'x)) =$$
$$\lambda s'x.(\ (\lambda sz.s^p(z))\ s'\ (\lambda sz.s^q(z)s'x)\ ) = \lambda s'x.(\ \lambda z.s'^p(z)\ (\lambda s'^q(x))\ ) =$$
$$\lambda s'x.(s'^p(s'^q(x))) = \lambda s'x.s'^{p+q}(x)$$

8.6. Zadania do domu - Część II

1. Zdefiniować mnożenie
2. Zdefiniować potęgowanie

8.6.1. Mnożenie

Mnożenie opisane jest poniższym wzorem:

$$\lambda xyz.x(yz)$$

(Nie udało się znaleźć wyprowadzenia tego wzoru)

8.6.2. Potęgowanie

Potęgowanie opisane jest poniższym wzorem:

$$\lambda b.\lambda e.eb$$

(Nie udało się znaleźć wyprowadzenia tego wzoru)

9 Ćwiczenia 9 (K2) - 27.04.2020

9.1. Kolokwium I

- Kolokwium Online - 06.05.2020, godzina 13.00
- Książki do kolokwium:
  - N.J. Cutland Computability, an introduction to recusive function theory (rozdział 2)

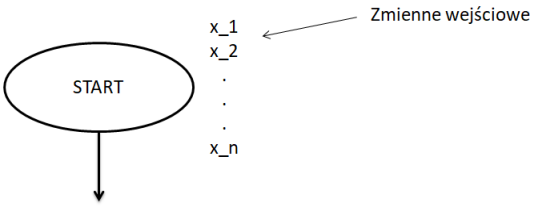
9.2. c.d. odejmowanie w wyrażeniach lambda TODO

Na tym kończymy lambdy i przechodzimy do nowego modelu obliczalności...

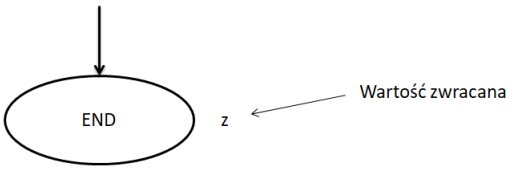
9.3. Schematy blokowe

Podstawowe bloki w schematach blokowych z których możemy korzystać:

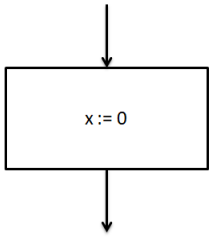
- START



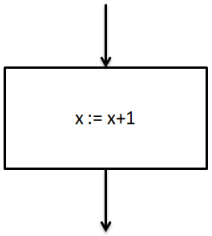
- END



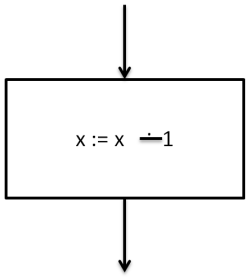
- Deklaracja zmiennej (wartość początkowa zawsze musi być równa 0)



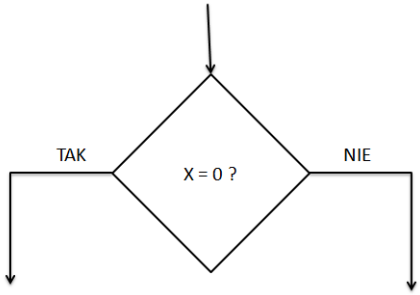
- Inkrementacja zmiennej



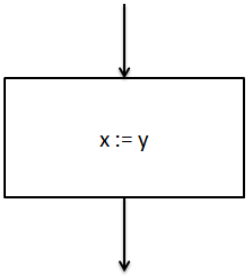
- Dekramentacja zmiennej (odejmowanie z kropką)



- WYBÓR

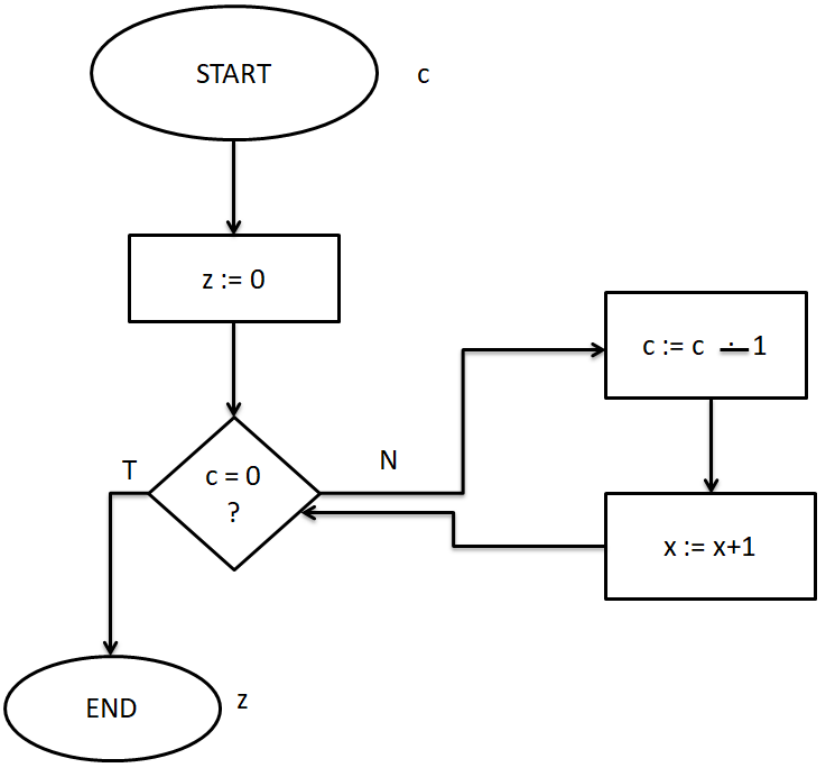


- Podstawienie



- W schemacie można również deklarować nowe zmienne, ale mają one zawsze na początku wartość 0

Przykład - program, który zwraca stałą c, która jest na wejściu.



9.4. Schematy blokowe jako model obliczalności

Aby pokazać, że schematy blokowe są modelem obliczalności musimy pokazać, to co pokazywaliśmy dla funkcji rekurencyjnych, a więc musimy przy ich pomocy zapisać takie funkcje jak:

- projekcja
- następnik
- składanie schematów
- rekurencja
- wyszukiwanie
- itd.

9.5. Składanie funkcji

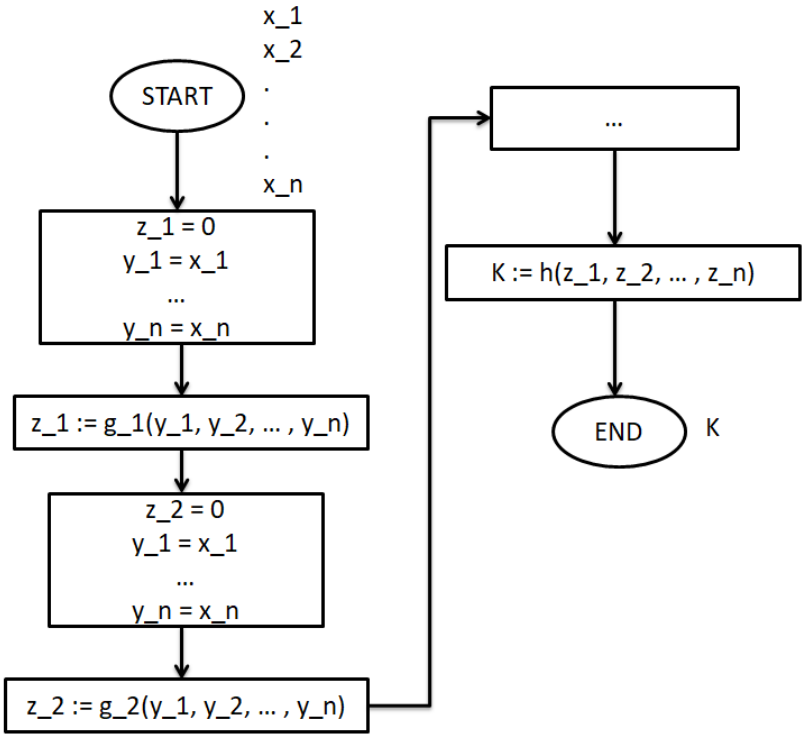
Teraz zapiszemy składanie funkcji przy pomocy schematu blokowego. Załóżmy, że chcemy obliczyć funkcję h, która wyraża się wzorem:

$$h(g_1(x_1, \ldots, x_n), g_2(x_1, \ldots, x_n), \ldots, g_n(x_1, \ldots, x_n))$$

Aby zapisać taką funkcję przy pomocy schematu blokowego stosujemy następujące podejście:

- Wejściem do schematu są argumenty $x_1, x_2, \ldots x_n$
- W kolejnych bloczkach obliczamy wartości funkcji g_i i zapisujemy wynik do nowej zmiennej z_i
- Zmienne wejściowe przekazujemy do każdej kolejnej funkcji g_i jako argumenty
- W ostatnim kroku obliczamy funkcję h, jako argumenty przekazujemy do niej wyniki zapisane w zmiennych z_i

Schemat prezentuje poniższy rysunek

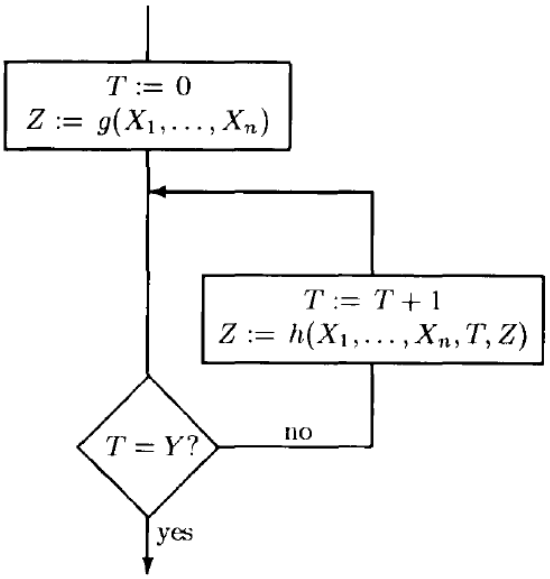


9.6. Prymitywna rekurencja

Teraz zapiszmy sobie prymitywną rekurencje przy pomocy schematu blokowego. Jak pamiętamy prymitywna rekurencja wyraża się wzorem:

$$f(x_1, x_2, \ldots, x_n, 0) = g(x_1, x_2, \ldots, x_n)$$
$$f(x_1, \ldots, x_n, y+1) = h(x_1, \ldots, x_n, y, f(x_1, x_2, \ldots, x_n, y))$$

Delikatnie upraszczając możemy to narysować jako:



W schemacie powyżej mamy sprawdzanie $t = y?$, M.A. stwierdził na zajęciach, że nie będzie tego pokazywać, bo skoro takie sprawdzanie łatwo zrobić dla 0, to łatwo jest też zrobić dla dowolnego y.

9.7. Programowanie przy użyciu WHILE TODO

(Uzupełnić po następnych ćwiczeniach, bo dopiero zaczęliśmy) W programowaniu przy użyciu WHILE możemy korzystać z następujących operacji:

- $x_j := x_i + c$
- $x_j := x_i - c$
- $x := x + 1$
- $x := x - 1$

Programy oznaczamy symbolami:

- P_i, np. P_1, P_2

Każdy program może też być podprogramem. Możemy zapisać np.

$$WHILE\ x! = 0\ DO\ P$$

9.8. Zadania do domu cz. I TODO

- Narysować schemat następnika
- Narysować schemat projekcji
- Przeszukiwanie

10 Ćwiczenia 10 (K2/K3) - 04.05.2020 - WHILE c.d. + wstęp do maszyn Turinga

10.1. Programowanie WHILE c.d. (K2)

W programowaniu z użyciem WHILE możemy używać operacji:

$$x := 0$$
$$x := x + 1$$
$$x := x \dot{-} 1$$

Ponadto zachodzi:

Jeżeli S jest programem, to również
$WHILE(x \neq 0)\ DO$
S
jest programem.

a także

Jeżeli $S_1, S_2, \ldots, S_n$jest programem, to również
$BEGIN$
S_1
S_2
...
S_n
END
jest programem.

10.1.1. Przykład

- Przy pomocy programowania $WHILE$ zapiszemy funkcję postaci:

$$if(x = y)\ do$$
$$\quad S_1$$
$$else$$
$$\quad S_2$$

$$
\begin{aligned}
&WHILE(y \neq 0)DO\\
&\quad x := x - 1\\
&\quad y := y - 1\\
&\quad f := 1\\
&\quad s := x\\
&\quad t := y\\
&\quad WHILE(s \neq t)DO\\
&\qquad f := 0\\
&\qquad s := t\\
&\quad WHILE(f \neq 0)DO\\
&\qquad S_1\\
&\qquad f := 0\\
&\quad WHILE(x \neq y)DO\\
&\qquad S_2\\
&\qquad x := y
\end{aligned}
$$

Programowanie $WHILE$ jest również modelem obliczalności, ponieważ przy jego pomocy możemy wyrażać takie funkcje jak następnik, projekcja, składanie funkcji itd.

10.2. Zadania do domu cz. I TODO (K2)

Zadanie

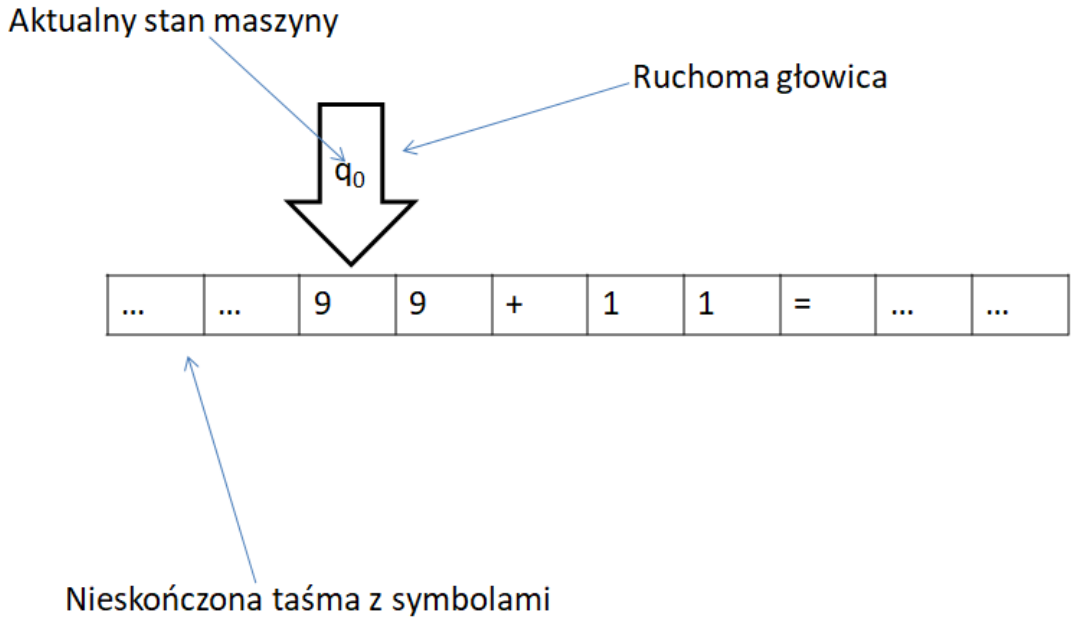
1. Utworzyć $x := y$ przy pomocy programu $WHILE$

10.3. Maszyna Turinga (K3)

Kolejne podejście do obliczalności, które będziemy rozważać zostało zaproponowane przez Turinga. Model ten nazywany jest maszyną Turinga.

Turing zaproponował nieskończenie długą taśmę (dowolnie długą), pociętą na bloki (komórki). Obok taśmy znajduje się ruchoma głowica, która przechowuje stan i odczytuje po jednej komórce z wejścia.

Na podstawie odczytanego symbolu i stanu w którym maszyna się aktualnie znajduje, głowica zapisuje w komórce nowy symbol i wybiera przejście o jedną komórkę w lewo albo prawo, a także nowy stan.



Algorytm działania maszyny Turinga można więc zapisać jako:

1. Odczytaj symbol wejściowy (ten na który wskazuje głowica)
2. Odczytaj aktualny stan (ten który przechowuje głowica)
3. Na podstawie symbolu wejściowego i aktualnego stanu zapisz symbol w komórce na którą wskazuje głowica
4. Na podstawie symbolu wejściowego i aktualnego stanu podejmij decyzję o przejściu w lewo albo prawo
5. Na podstawie symbolu wejściowego i aktualnego stanu podejmij decyzję o nowym stanie maszyny

10.3.1. Formalna definicja

Teraz podamy bardziej formalną definicję, a potem zobaczymy przykład. Formalnie ujmując, maszyna Turinga to zbiór siedmiu zbiorów:

$M = (Q, \Sigma, \Gamma, \sigma, q_0, \square, F_0)$

Q - skończony zbiór stanów głowicy maszyny

Σ - skończony alfabet (zestaw znaków na taśmie)

Γ - podzbiór Σ , zestaw znaków startowych (możliwych do użycia na początku działania maszyny)

$\sigma : Q \times \Sigma \to Q \times \Sigma \times \{L, P\}$ - funkcja opisująca działanie maszyny

q_0 - stan początkowy

$\square : \square \subset \Sigma$ - oznacza wszystkie puste pola na maszynie

$F_0 : F_0 \subset Q$ - zbiór stanów końcowych

Za działanie maszyny odpowiada funkcja σ, która przyjmuje jako wejście odczytany znak i aktualny stan i na tej podstawie wpisuje w to miejsce nowy znak, przechodzi w nowy stan oraz zmienia pozycję na jeden krok w lewo (L) lub prawo (P).

10.3.2. Przykłady

- Spróbujemy zamodelować algorytm, który na wejściu przyjmuje pewną liczbę symboli 1 i zwraca o jeden symbol 1 więcej, a więc przykładowo:

$$111 \to 1111$$
$$11111 \to 111111$$

Określmy sobie niektóre z elementów maszyny:

1. $\Sigma = \{1, \square\}$
2. $\Gamma = \{1\}$
3. $F_0 = \{q_k\}$

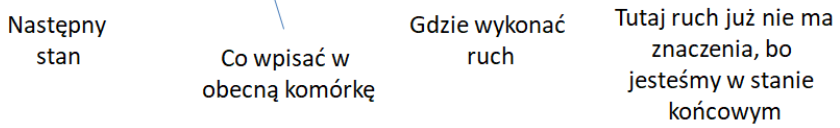
Teraz zastanawiamy się nad funkcją σ. Możemy ją zaprezentować na dwa sposoby:

- Tabelka
- Graf

Nasz algorytm będzie przesuwał głowicę w lewo. Jeśli napotka jedynkę to nie będzie jej modyfikował i zostanie w aktualnym stanie (zapisze 1 ponownie do tej komórki), a jeśli puste miejsce, to wpiszę w nie jedynkę i skończy działanie.

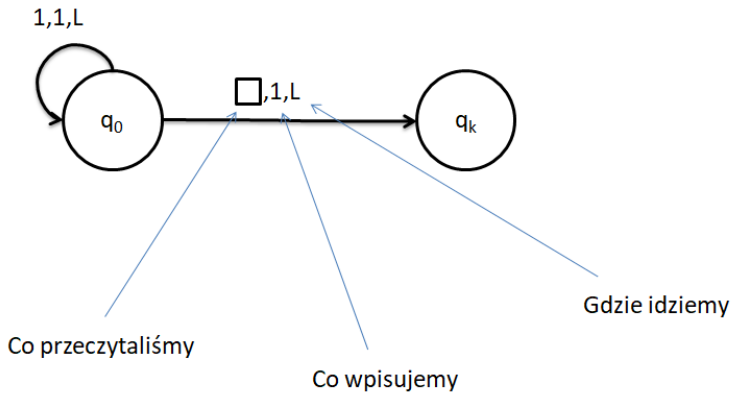
Przy pomocy tabelki:

	1	□
q_0	$(q_0, 1, L)$	$(q_k, 1, L)$
q_k		



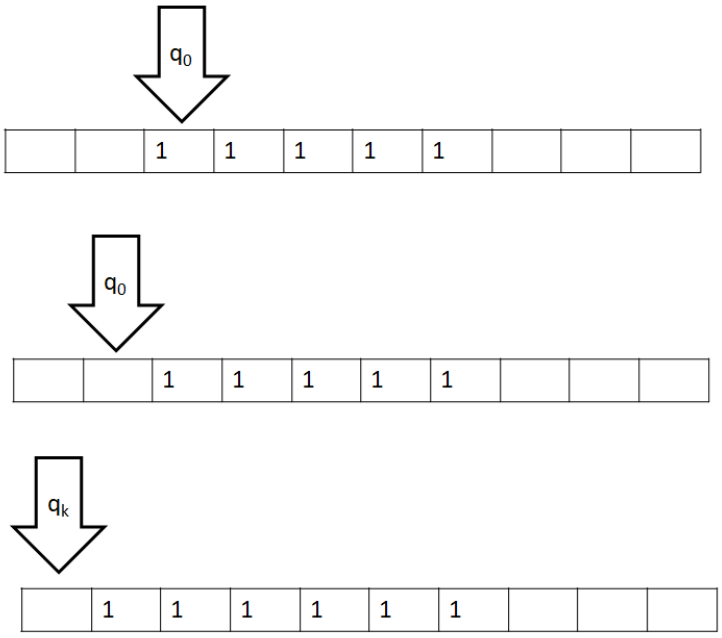
Zauważmy, że w drugim rzędzie nie ma już potrzeby wpisywania wyrażeń, ponieważ jest to stan końcowy z którego nigdzie dalej nie przechodzimy.

Przy pomocy grafu:



Zauważmy podobieństwo tych reprezentacji do automatów deterministycznych z lingwistyki formalnej.

Przykładowe działanie tej maszyny:



- Teraz spróbujmy zapisać bardziej skomplikowany przykład przy pomocy maszyny Turinga. Chcemy zamodelować sytuację w której na wejściu mamy n symboli 1, a maszyna 'zwraca' $2n$ symboli 1, czyli przykładowo:

$$111 \rightarrow 111111$$
$$11111 \rightarrow 1111111111$$

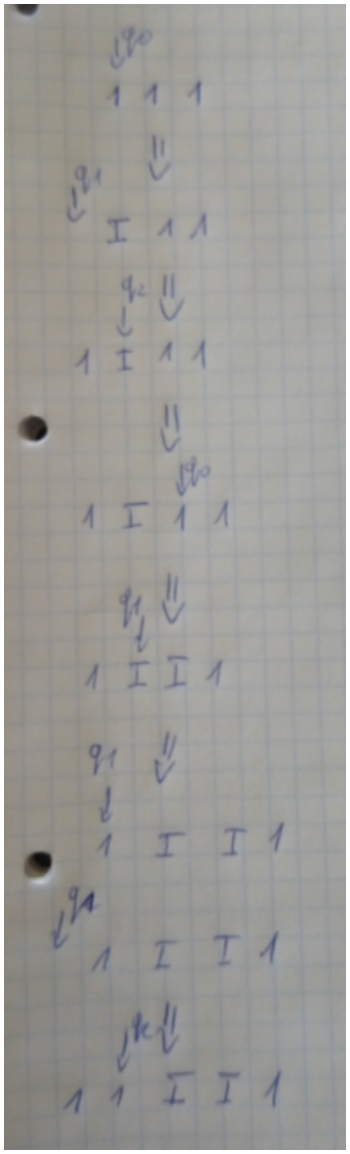
Zaczynamy tak jak ostatnio, definiujemy alfabet i wejście:

- $\Sigma = \{1, \square, I\}$
- $\Gamma = \{1\}$

Teraz myślimy nad algorytmem. Chcemy aby działał on w następujący sposób:

- Jedynkę wejściową zastępujemy symbolem I i przechodzimy na pierwsze wolne miejsce w lewo aby wpisać tam 1 (odpowiadające jedynce, którą zastąpiliśmy przez I)
- Następnie przechodzimy w prawo aż do napotkania kolejnej wejściowej jedynki, którą znów zastępujemy przez I, a następnie przechodzimy maksymalnie w lewo aby zastąpić kolejne puste pole symbolem 1
- Na koniec zastępujemy wszystkie symbole I symbolami 1

Fragment tego algorytm jest zaprezentowany poniżej:



Ten algorytm będzie implementowany przez taką funkcję:

	1	I	□
q_0	(q_1, I, L)	(q_0, I, P)	$(q_3, □, L)$
q_1	$(q_1, 1, L)$	(q_1, I, L)	$(q_2, 1, P)$
q_2	$(q_2, 1, R)$	(q_0, I, R)	
q_3	$(q_3, 1, R)$	$(q_3, 1, L)$	

10.4. Zadania do domu cz. II (K3)

Zadanie

✓Napisać maszynę Turinga, która będzie zwracać cztery razy więcej symboli 1 niż podane jest na wejściu.

Korzystamy z maszyny, którą mieliśmy już dla mnożenia razy dwa. Wtedy będąc w stanie q_1 gdy wejście było puste, to wpisywaliśmy jedynkę i rozpoczynaliśmy powrót w prawo poprzez przejście do q_2. Teraz zamiast tego dodamy jeszcze dodatkowo dwie jedynki i dopiero potem przejdziemy do stanu q_2. Dodatkowe dwie jedynki dodamy poprzez dodanie dwóch dodatkowych stanów:

	1	I	□
q_0	(q_1, I, L)	(q_0, I, P)	$(q_3, □, L)$
q_1	$(q_1, 1, L)$	(q_1, I, L)	$(q_4, 1, L)$
q_2	$(q_2, 1, R)$	(q_0, I, R)	
q_3	$(q_3, 1, R)$	$(q_3, 1, L)$	
q_4			$(q_5, 1, L)$
q_5			$(q_2, 1, P)$

11 Ćwiczenia 11 (K3) - 11.05.2020 - Rozwiązywanie zadań z Turinga

Zapowiedź kolokwium 2 (około 25.05.2020):

- Rachunek lambda
- Schematy blokowe
- Programowanie WHILE

11.1. Rozwiązywanie zadań

Na tych zajęciach rozwiązujemy różne zadania z maszyn Turinga. Najpierw mamy około 45 min na rozwiązanie, a potem prezentujemy co się udało zrobić.

Zadanie

Napisać maszynę Turinga taką, że:

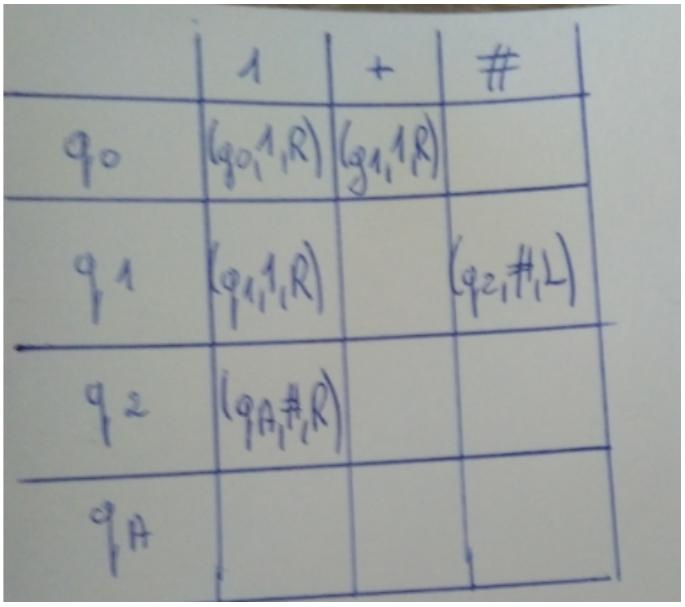
1. ✓Dodawanie dwóch ciągów jedynek (dodawanie w systemie unarnym)
   - Przykładowe wejście: 111+11111
   - Wyjście: 11111111
2. ✗Mnożenie w systemie unarnym
   - Przykładowe wejście: 111*11
   - Wyjście: 111111
3. ✓Dodawanie jedynki w systemie binarnym
   - Przykładowe wejście (liczba 5): 101
   - Wyjście (liczba 6): 110
4. ✓Przesuń napis o jedno pole w prawo
   - Przykładowe wejście: _101_
   - Wyjście: __101_
5. ✓Zamień znaki na przeciwne
   - Przykładowe wejście: 1101
   - Wyjście: 0010
6. ✓Pomnóż razy dwa liczbę binarną
   - Przykładowe wejście (liczba 5): _101_
   - Wyjście (liczb 10): _1010_

Zadanie

7. ✓Usuń wszystkie jedynki w słowie
   - Przykładowe wejście: _0110101_
   - Wyjście: _000_
8. ✗Podwój słowo i podwój liczbę 0 w tym słowie
   - Przykładowe wejście: _0010_
   - Wyjście: _0000010000000100_
9. ✗Dodawanie w systemie binarnym
   - Przykładowe wejście:
   - Wyjście:

1. OPIS:
   - Idziemy w prawo aż napotkamy plusa
   - Gdy napotkamy plusa, zamieniamy go na 1 i idziemy dalej w prawo
   - Idziemy w prawo aż do napotkania pustego miejsca
   - Gdy napotkamy puste miejsce, kasujemy pierwszą jedynkę na lewo od niego

   Tabelka:



2. OPIS:

   - TODO

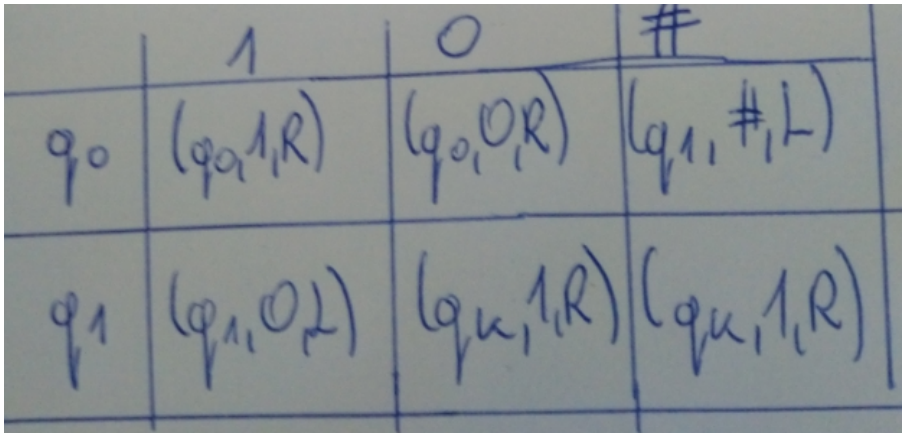
   Tabelka: TODO

3. OPIS:

   - Gdy dodajemy jedynkę, to zaczynamy analizę od prawej strony i idziemy do lewej, dlatego najpierw przechodzimy maksymalnie na prawo
   - Każdą jedynkę zastępujemy zerem aż do napotkanie pierwszego 0 lub znaku pustego
   - Gdy napotkamy pierwsze 0, to je zamieniamy na 1 i kończymy
   - Gdy napotkamy koniec liczby to dopisujemy z lewej strony jedynkę i kończymy

   $$101(5) \to 110(6)$$
   $$110(6) \to 111(7)$$
   $$111(7) \to 1000(8)$$
   $$1000(8) \to 1001(9)$$

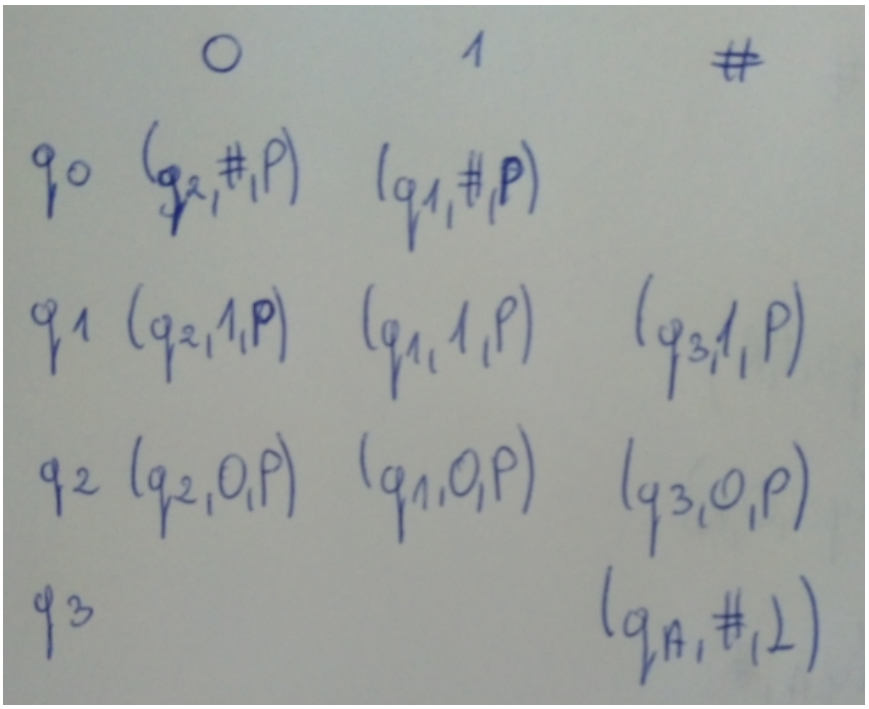
   Tabelka:



4. OPIS:

   - Idziemy od strony lewej do prawej
   - Odczytujemy symbol, jeśli jest to 0, to idziemy do q_2
   - Jeśli jest to 1, to idziemy do q_1
   - Jeśli robimy ruch ze stanu q_2, to zawsze wpisujemy 0, a jeśli ze stanu q_1, to zawsze wpisujemy 1
   - Uwzględniamy koniec wczytywania przez obsługę znaku pustego

   Tabelka:



5. OPIS:

   - Zero na jeden, a jeden na 0 i tak aż do pustego znaku...

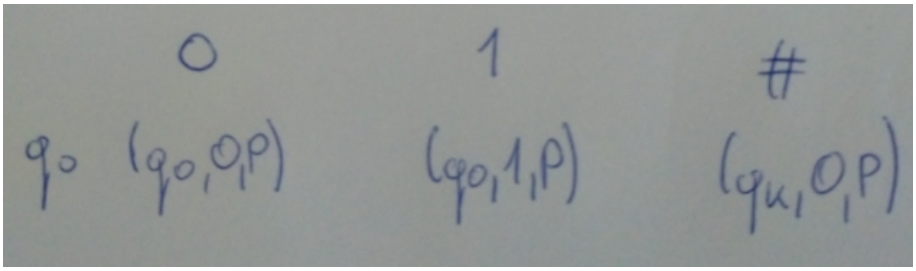
   Tabelka jest tutaj mega prosta...

6. OPIS:

   - Mnożenie liczb binarnych przez dwa jest dość proste, bo wystarczy dodać do liczby 0 na jej końcu

$$(2)10 \rightarrow 100(4)$$
$$(3)11 \rightarrow 110(6)$$
$$(4)100 \rightarrow 1000(8)$$

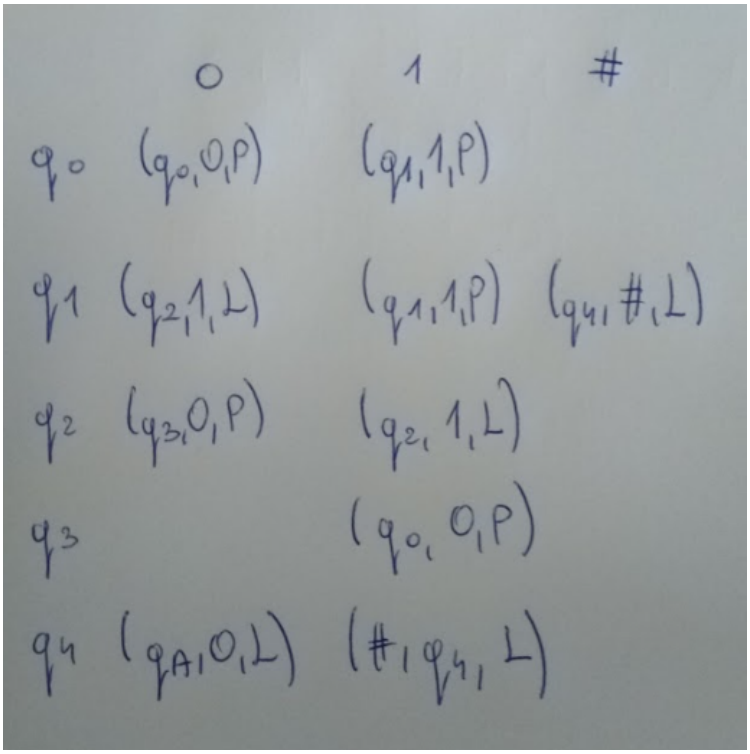
Tabelka:



7. OPIS:

   - Algorytm, który będzie realizować maszyna ma na celu w pierwszym kroku zamienić przykładowe słowo 0110101 na 0001111
   - Na koniec usuwa wszystkie jedynki
   - Na początek idziemy w prawo, aż do napotkanie pierwszego zera znajdującego się po przynajmniej jednej jedynce
   - Takie zero zamieniamy na 1 i zaczynamy wracać w lewo aż do napotkanie pierwszego 0
   - Gdy napotkamy pierwsze 0, to zaczynamy iść w prawo aż napotkamy pierwszą jedynkę
   - Pierwszą napotkaną jedynkę zamieniamy na 0 i znów idziemy w prawo aż do napotkania pierwszego zera znajdującego się po przynajmniej jednej jedynce (punkt 3)

   Tabelka:



8. OPIS:

   - TODO

   Tabelka: TODO

9. OPIS:

   - TODO

   Tabelka: TODO

11.2. Zadania do domu

Zadanie

Napisać maszynę Turinga, która sprawdzi czy słowo należy do zadanego języka. Jeśli się uda, można zrobić też tak, aby maszyna odrzucała słowo jeśli jest niepoprawne (ale wystarczy aby akceptowała poprawne).

1. ✗$L = \{ww^r : w \in \{1, 0\},\ r > 0\}$
2. ✗$L = \{0^n, 1^n, 2^n : n > 0\}$
3. ✓$L = \{a^i b^j c^k,\ i < j < k\}$
4. ✓$L = \{w \in \{a, b\}^* :$ sprawdzić czy liczba wystąpień a jest równa liczbie wystąpień b$\}$. Uwaga: symbol $\{a, b\}^*$ oznacza dowolny niepusty ciąg składający się z tych znaków np. $aabbabaa$.

1. Dość proste zadanie...
2. Dość proste zadanie...
3. Opis
   - Zaczynamy od tego, że wykreślamy po jednym a,b i c aż zostaną same b i c
   - Teraz dla każdego b ma istnieć jedno c więc chodzimy w tą i z powrotem
   - Gdy dojdziemy do lewego końca, to idziemy w prawo, teraz pierwsze napotkane c powoduje, że akceptujemy słowo
   - aabbbcccc -> xaxbbxccc -> xxxxbxxcc -> xxxxxxxc -> accept

	a	b	c	x	$\#$
q_0	(q_1, x, P)	(q_4, x, P)	q_n	(q_0, x, P)	
q_1	(q_1, a, P)	(q_2, x, P)	q_n	(q_1, x, P)	q_n
q_2	q_n	(q_2, b, P)	(q_3, x, L)	(q_2, x, P)	q_n
q_3	(q_3, a, L)	(q_3, b, L)		(q_3, x, L)	$(q_0, \#, P)$
q_4		(q_4, b, P)	(q_5, x, L)	(q_4, x, P)	q_n
q_5		(q_4, x, P)		(q_5, x, L)	$(q_6, \#, P)$
q_6			q_a	(q_6, x, P)	q_n

UWAGA: Tutaj trzeba było dodać jeszcze jakieś stany, bo przy jakimś wejściu M.A. zauważył, że maszyna coś robi źle, chodziło o jakąś niewielką modyfikację, ale już nie pamiętam jaką.

4. Opis

   - Czytamy symbol wejściowy i zamieniamy go na 0
   - Po przeczytaniu idziemy tak długo w prawo aż znajdziemy przeciwny do przeczytanego i zamieniamy go na 0
   - Wracamy na początek i odczytujemy pierwszy, który nie jest zerem
   - Na koniec sprawdzamy czy otrzymaliśmy same 0
   - aabbabaa -> 0a0babaa -> 0000abaa -> 000000aa -> q_N

	a	b	0	$\square$
q_0	$(q_1, 0, P)$	$(q_2, 0, P)$	$(q_0, 0, P)$	$(q_a, \square, L)$
q_1	(q_1, a, P)	$(q_3, 0, L)$	$(q_1, 0, P)$	$(q_n, \square, L)$
q_2	$(q_3, 0, L)$	(q_2, b, P)	$(q_2, 0, P)$	$(q_n, \square, L)$
q_3	(q_3, a, L)	(q_3, b, L)	$(q_3, 0, L)$	$(q_0, \square, P)$
q_a				
q_n				

12 Ćwiczenia 12 (K3) - 18.05.2020 - Rozstrzygalność, decydowalność, języki rekurencyjne i rekurencyjnie przeliczalne, wstęp do problemu stopu

12.1. Problemy i rozstrzygalność

Do tej pory rozważaliśmy obliczalność. Mówiliśmy np. że funkcja jest obliczalna jeśli zachodzą pewne warunki. Teraz będziemy rozważać rozstrzygalność, która odnosi się do problemów. Zacznijmy więc od tego czym jest problem.

12.1.1. Problem

Problemy powiązane są z funkcjami. Tzn. problemem nazywamy aplikację konkretnych argumentów do danej funkcji. Problem odpowiada więc na pytanie: *Jaka jest wartość funkcji f dla pewnych konkretnych wartości wejściowych ?*

Jako przykład

- Mamy daną funkcję dodawania $+$
- Dla funkcji $+$, problemem jest więc np. $3 + 4$

Kolejny przykład

- Mamy daną funkcję r, która sprawdza czy słowo należy do języka palindromów
- Dla funkcji r, problemem jest więc np. zaaplikowanie do niej argumentu 0101010

12.1.2. Rozstrzygalność

Problem może być:

- rozstrzygalny (całkowicie)
- częściowo rozstrzygalny
- nierozstrzygalny

Teraz omówimy sobie każdy z tych przypadków.

- Problem rozstrzygalny - to taki, dla którego funkcja jest obliczalna. Wiemy natomiast, że funkcje obliczalne są wyrażalne przez maszynę Turinga. A zatem problem jest rozstrzygalny wtedy gdy możemy utworzyć taką maszynę Turinga, która w skończonym czasie da jego rozwiązanie (obliczy wartość funkcji).

Przykładem problemu rozstrzygalnego jest dodawanie unarne. Pokazaliśmy na jednych z poprzednich ćwiczeń, że dla funkcji dodawania unarnego można zbudować maszynę Turinga, która zostawi na taśmie poprawne rozwiązanie.

Kolejnym przykładem może być sprawdzanie czy słowo należy do języka $L = \{0^n, 1^n, 2^n : n > 0\}$. Pokazaliśmy, że istnieje maszyna Turinga, która potrafi dać odpowiedź w skończonej liczbie kroków.

- Problem częściowo rozstrzygalny - występuje wtedy gdy powiązana z nim funkcja jest zawsze poprawnie obliczana (przez odpowiadającą jej maszynę Turinga) w przypadku gdy maszyna ta się zatrzymuje. Mogą natomiast istnieć takie wejścia dla których maszyna się nie zatrzyma. Jeśli już się jednak zatrzyma, to obliczona wartość musi być poprawna
- Problem nierozstrzygalny - nie istnieje maszyna Turinga, która potrafiłaby obliczyć powiązaną funkcje

12.2. Problemy decyzyjne i decydowalność

Wiemy już czym jest problem i że zawsze jest on powiązany z funkcją, a więc także z maszyną Turinga. Czym jest problem decyzyjny ? To po prostu problem, na który odpowiedź brzmi tak lub nie. A więc funkcja powiązana z takim problemem może przyjmować tylko dwie wartości oznaczające prawdę albo fałsz. Przykłady

- *given two regular grammars, R_1 and R_2, are they equivalent?*
- *given a number, N, is N prime?*
- *given a TM, M, and a tape T, will M halt eventually when started on the tape T?*

W przypadku problemów decyzyjnych możemy mówić o decydowalności, co jest odpowiednikiem rozstrzygalności. A więc może mówić, że problem decyzyjny jest decydowalny (rozstrzgalny), częściowo decydowalny (częściowo rozstrzygalny) czy też niedecydowalny (nierozstrzygalny).

Np. drugi z powyższych przykładów jest problem rozstrzygalnym (decydowalnym), ponieważ możemy skonstruować taką maszynę, że dla każdej wejściowej liczby poprawnie odpowie w skończonej liczbie kroków czy jest ona parzysta.

Trzeci z przykładowych problemów to tzw. problem stopu i jest on nierozstrzygalny (niedecydowalny), wrócimy do niego później.

12.3. Języki i decydowalność

Pewną szczególną klasą problemów decyzyjnych są problemy typu:
czy dane słowo wejściowe należy do danego języka ?

O takim problemie możemy oczywiście powiedzieć, że jest decydowalny, częściowo decydowalny czy też niedecydowalny. Czasami w takich problemach używamy tego pojęcia względem języka. A zatem mówimy, że to język jest decydowalny (gdy problem jest decydowalny) itd.

Aby móc określić w łatwy sposób w takich problemach czy język (problem) jest decydowalny rozważamy języki rekurencyjne oraz rekurencyjnie przeliczalne.

12.3.1. Język rekurencyjny

Język rekurencyjny to taki dla którego istnieje maszyna Turinga taka, że:

- dla każdego słowa wejściowego zatrzyma się i da odpowiedź czy słowo należy do języka czy nie

A zatem mówimy, że

- język (problem) jest decydowalny jeśli język jest rekurencyjny

12.3.2. Język rekurencyjnie przeliczalny

Język rekurencyjnie przeliczany to taki dla którego istnieje maszyna Turinga taka, że:

- jeżeli słowo wejściowe należy do języka, to maszyna się zatrzyma i da odpowiedź *true*
- maszyna może, ale nie musi się zatrzymać jeśli słowo wejściowe nie należy do języka

A zatem mówimy, że

- język (problem) jest częściowo decydowalny jeśli język jest rekurencyjnie przeliczalny

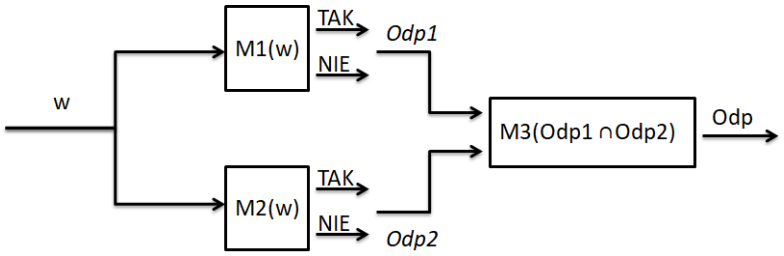
12.3.3. Język niedecydowalny

- język (problem) jest niedecydowalny jeśli nie istnieje żadna maszyna Turinga dla języka

Przykład
Dane są dwa języki L_1 oraz L_2, które są rekurencyjne. Co możemy powiedzieć o językach:

$$a)\ L_1 \cap L_2$$
$$b)\ L_1 \cup L_2$$

Do tego zadania podejdziemy graficznie. Dla iloczynu możemy stworzyć taką maszynę M_3, która będzie brała na wejściu odpowiedzi z M_1 i M_2 i liczyła ich iloczyn. Taka odpowiedź zawsze będzie jednoznaczna i maszyna zawsze się zatrzyma, bo L_1 i L_2 są rekurencyjne.



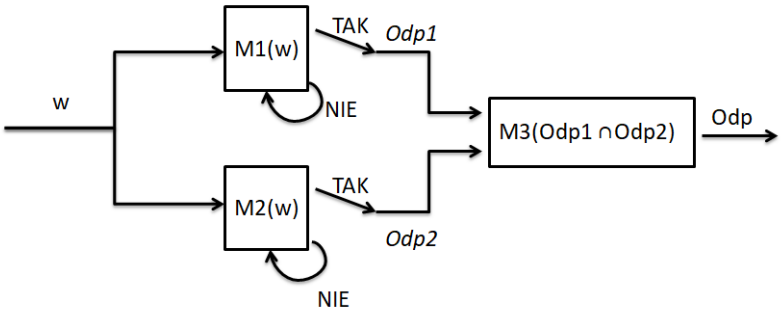
Pokazaliśmy więc, że jeśli L_1 oraz L_2 są rekurencyjne, to język $L_1 \cap L_2$ również jest rekurencyjny.

Analogiczne rozumowanie prowadzimy dla sumy, ale tym razem maszyna M_3 oblicza sumę odpowiedzi. W ten sam sposób pokazujemy, że język $L_1 \cup L_2$ jest rekurencyjny.

Przykład
Dane są dwa języki L_1 oraz L_2, które są rekurencyjnie przeliczalne. Co możemy powiedzieć o językach:

$$a)\ L_1 \cap L_2$$
$$b)\ L_1 \cup L_2$$

W przypadku $a)$ tworzymy znów graficzną reprezentacje:

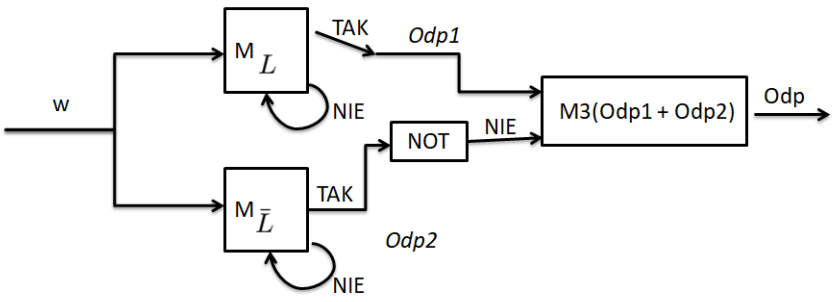


W przypadku gdy słowo będzie należało do $L_1 \cap L_2$, to zarówno maszyna M_1 jak i M_2 da wynik pozytywny i zakończy działanie (bo L_1 i L_2 są rekurencyjnie przeliczalne), a więc i maszyna M_3 da wynik pozytywny w skończonym czasie. Jeśli słowo będzie należało tylko do L_1 albo tylko do L_2, to jedna z maszyn się zapętli i maszyna M_3 nie da wyniku w skończonym czasie.

Pokazaliśmy więc, że istnieje taka maszyna M_3, że potrafi dać wynik w skończonym czasie o ile słowo należy do języka $L = L_1 \cap L_2$. Jeśli jednak nie należy do języka L, to nie da nam wyniku w skończonym czasie. Wniosek: Jeżeli L_1 oraz L_2 są rekurencyjnie przeliczalne, to również $L = L_1 \cap L_2$ jest rekurencyjnie przeliczalny.

W przypadku $b)$ będzie trochę inaczej. Może być bowiem sytuacja, że słowo należy do L_1, a nie należy do L_2. Wtedy jedna z maszyn się zablokuje i nie dostaniemy odpowiedzi. A zatem język $L = L_1 \cup L_2$ nie jest rekurencyjnie przeliczalny, pomimo, że L_1 i L_2 są.

Przykład
Wiedząc, że języki L oraz $\bar{L}$ są rekurencyjnie przeliczalne, pokazać że L jest językiem rekurencyjnym. Dla przypomnienia $\bar{L}$ oznacza dopełnienie: $\bar{L} = \sum^* \setminus L$, gdzie $\sum^*$ to wszystkie słowa generowane przez alfabet.



Aby sprawdzić czy słowo należy do L, czy też nie, wystarczy, że zanegujemy wyjście z maszyny opisującej język $\bar{L}$. Sumę odpowiedzi z maszyn kierujemy na maszynę M_3, która daje nam odpowiedź na temat tego czy słowo wejściowe należy do L czy też nie. A więc tym samem pokazaliśmy, że istnieje taka maszyna M_3, która potrafi jednoznacznie określić, czy słowo należy do L czy też nie, a więc pokazaliśmy, że L jest rekurencyjny.

12.4. Problem stopu - wstęp

Problem stopu możemy łatwiej zrozumieć przez pewną analogię do programów, które piszemy. Tak naprawdę sprawdza on, czy biorąc dowolny program (dowolną maszynę Turinga) oraz dowolne wejście do tego programu (dowolne wejście na taśmę), nasz program kiedyś się zatrzyma. Nie istnieje taka maszyna Turinga, która dałaby na to odpowiedź, a więc problem jest niedecydowalny.

W problemie stopu rozważamy tzw. uniwersalną maszynę Turinga. Zauważmy, że wejściem do tego problemu jest dowolna maszyna Turinga oraz słowo wejściowe. Uniwersalna maszyna Turinga to taka, która przyjmuje na wejście dowolną inną maszynę Turinga i wykonuje jej program z przekazanym wejściem. Więcej o tym na następnych ćwiczeniach.

12.5. Zadania do domu TODO

Zadanie

1. Język L jest rekurencyjny. Pokazać, że istnieje nieskończenie wiele maszyn Turinga, które go akceptują (decydują).
2. M, N to maszyny Turinga. Dany jest problem decyzyjny P: *Czy języki rozpoznawane przez M i N są równoliczne ?* Określić czy problem P jest rozstrzygalny, częściowo rozstrzygalny czy nierozstrzygalny. Przypomnienie: równoliczność oznacza, że dwa zbiory mają tyle samo elementów, a inaczej, że każdemu elementowi ze zbioru A można przyporządkować element ze zbioru B.

13 Ćwiczenia 13 (K3) - 25.05.2020 - problem stopu i uniwersalna maszyna Turinga

13.1. Problem stopu - c.d.

Jak już wiemy problem stopu pojawia się w sytuacji gdy chcemy opisać maszynę Turinga przy pomocy innej maszyny Turinga. W tym przypadku chcemy opisać dowolną maszynę Turinga przy pomocy uniwersalnej maszyny Turinga.

13.1.1. Symbole wejściowe dla uniwersalnej maszyny Turinga

Możemy się słusznie zastanawiać jak ma działać uniwersalna maszyna Turinga (UMT). Jeśli przyjmuje na wejście dowolną maszynę, to każda z nich może operować na innym alfabecie. A więc teoretycznie, jest nieskończenie wiele symboli wejściowych, które mogą być wejściem do UMT.

Aby obejść ten problem, następuje konwersja maszyny wejściowej na zera i jedynki. Finalnie operujemy więc tylko na zerach i jedynkach:

- 1 - reprezentuje spacje (pustą komórkę)
- 0 - koduje stan

Jak działa taka konwersja ?

Przyjmujemy, że maszyna Turinga jest reprezentowana przez funkcję f, która przyjmuje jako argument stan oraz symbol wejściowy. Wyjściem z funkcji jest kolejny stan, symbol, który wpiszemy w komórkę oraz kierunek w który pójdziemy, np.

$$f(q_0, x) = q_1, y, L$$

Oznacza, że będąc w stanie q_0, pod wpływem odczytania symbolu x, przechodzimy w stan q_1, zapisujemy symbol y i idziemy w lewo.
Każdy ze stanów i symboli jest kodowany przy pomocy zer i jedynek, np.

$$q_0 \to 000 \quad x \to 0000000$$
$$q_1 \to 0000 \quad y \to 00000000$$
$$q_2 \to 00000 \quad L \to 000000000$$
$$q_3 \to 000000 \quad P \to 0000000000$$

W takim razie

$$f(q_0, x) = q_1, y, L \to q_0, x, q_1, y, L$$

kodujemy jako

$$1000100000001000010000000010000000000$$

13.2. Problem stopu i dowód

Będziemy rozważać dwa problemy związane z UMT:

1. Dana jest maszyna Turinga M, która jest wejściem do UMT. Czy istnieje taka uniwersalna maszyna Turinga, która jest w stanie powiedzieć, że maszyna wejściowa zatrzymuje się na wejściu W. Odpowiedź: Tak
2. Dane jest maszyna Turinga M, która jest wejściem do UMT. Czy istnieje taka uniwersalna maszyna Turinga, która jest w stanie powiedzieć, że maszyna wejściowa wpada w nieskończoną pętlę ? (Problem stopu) Odpowiedź: Nie

13.2.1. Dowód

I tu był dowód przez zaprzeczenie dla powyższych problemów, jednak dość długi i dość trudny więc go sobie daruję...

13.3. Zadanie domowe

Zadanie

1. Podać przykład zbioru, który nie jest rekurencyjnie przeliczalny.

Jeśli weźmiemy uniwersalną maszynę Turinga taką, że:

- Jeśli maszyna wejściowa stopuje to zwraca TAK
- Nic nie zwraca (pętla) jeśli maszyna wejściowa nie stopuje

To zbiór takich maszyn stanowić będzie zbiór rekurencyjnie przeliczalny.

14 Ćwiczenia 14 (K3) - 01.06.2020 - Redukowalność, maszyny ATM i ETM

14.1. Redukowalność

W poprzednich rozdziałach pokazywaliśmy, że coś jest rozstrzygalne, nierozstrzygalne lub częściowo rozstrzygalne. Aby pokazać, że język jest decydowalny wystarczy pokazać, że język i jego dopełnienie są rekurencyjnie przeliczalne.

Teraz skupiamy się na pokazywaniu, że język jest niedecydowalny. Istnieje kilka metod dowodzenia, że dany język jest niedecydowalny. Możemy to zrobić np.

- poprzez zaprzeczenie - jak robiliśmy to dla UTM
- poprzez redukcję - jak pokażemy teraz
- poprzez twierdzenie Rice'a - jak pokażemy na następnych ćwiczeniach

Redukowalność mówi, że
Jeżeli język L_1 redukuje się do języka L_2 oraz L_1 jest niedecydowalny, to wtedy L_2 jest także niedecydowalny.

Przykład
Załóżmy, że mamy dwa języki:

- $HALT = \{\langle M, w\rangle$: M to maszyna Turinga, która zatrzymuje się na wejściu w$\}$
- $ATM = \{\langle M, w\rangle$: M to maszyna Turinga, która akceptuje w$\}$ - jest niedecydowalny

Możemy sprowadzić język ATM do języka HALT i pokazać tym samym, że HALT jest niedecydowalny. Dowód z książki:

Proof. We will reduce A_{TM} to HALT. Based on a machine M, let us consider a new machine $f(M)$ as follows:

```
On input x
    Run M on x
    If M accepts then halt and accept
    If M rejects then go into an infinite loop
```

Observe that $f(M)$ halts on input w if and only if M accepts w
Suppose HALT is decidable. Then there is a Turing machine H that always halts and $\mathbf{L}(H) =$ HALT. Consider the following program T

```
On input ⟨M, w⟩
    Construct program f(M)
    Run H on ⟨f(M), w⟩
    Accept if H accepts and reject if H rejects
```

T decides A_{TM}. But, A_{TM} is undecidable, which gives us the contradiction. □

Na zajęciach rozważaliśmy jeszcze język ETM taki, że:

$$ETM = \{\langle M\rangle : \text{ M to maszyna Turinga, taka, że } L(M) \text{ - zbior pusty}\}$$

M.A. kazał określić czy jest rozstrzygalny, częściowo rozstrzygany czy nierozstrzygalny i podał podpowiedź, że warto rozważyć dopełnienie ETM:

$$E\bar{T}M = \{\langle M\rangle : \text{ M to maszyna Turinga taka,że } L(M) \text{ - zbior niepusty }\}$$

15 Ćwiczenia 15 (K3) - 08.06.2020 - Twierdzenie Rice'a

15.1. Twierdzenie Rice'a

Twierdzenie Rice'a to narzędzie, które pozwala nam na decydowanie o tym czy język, który składa się z maszyn Turinga o pewnej własności jest decydowalny.

Przez L_p oznaczamy język składający się z maszyn Turinga, które opisują język spełniający własność P.

$$L_p = \{\langle M\rangle | L(M) \in P\}$$

Przykład własności P:

$$P = \{\langle M\rangle : |L(M)| = \infty\} \text{ - jezyki, ktore maja nieskoczenie wiele mozliwych slow}$$

A zatem w tym przypadku L_p oznaczałoby taki zbiór maszyn Turinga z których każda akceptuje język, który ma nieskończenie wiele słów.

Decydowalność L_p zależy od tego czy własność P jest własnością trywialną czy nietrywialną. O tym właśnie mówi Twierdzenie *Rice'a*:
Jeżeli P jest własnością nietrywialną to L_p jest niedecydowalny (nierozstrzygalny).

15.2. Własności trywialne i nietrywialne

Mówimy, że własność P jest trywialna jeżeli:

- Nie istnieje żaden język rekurencyjnie przeliczalny, który ją spełnia lub
- Każdy język rekurencyjnie przeliczalny ją spełnia

Każda własność, która nie jest trywialna jest nietrywialna.

15.3. Przykłady

Zadanie

Dane są własności P. Co można powiedzieć na temat języków L_p.

1. $\{\langle M\rangle : |L(M)| = \infty\}$
2. $\{\langle M\rangle : 1011 \in L(M)\}$
3. $\{\langle M\rangle : L(M) = \Sigma^*\}$

Z twierdzenia Rice'a wiemy, że L_p będzie zależało od tego jaka jest własność P. Chcemy więc sprawdzić czy własność P jest trywialna czy nie.

2. Sprawdzamy czy własność P jest nietrywialna. Szukamy zbioru nietrywialnego a więc chcemy pokazać, że własność P jest spełniona dla niektórych języków rekurencyjnie przeliczalnych, a dla niektórych nie jest spełniona.

   - Weźmy języki takie, że słowo 1011 należy do tych języków, tzn. $1011 \in L(M_1)$. Dla dowolnych dwóch takich języków $L(M_1) = L(M_2)$ mamy pewność, że własność P jest spełniona, a więc $L(M_1) = L(M_2) \implies \langle M_1\rangle \in P \wedge \langle M_2\rangle \in P$
   - Weźmy języki takie, że słowo 1011 nie należy o tych języków, tzn. $1011 \notin L(M_1)$. Dla dowolnych dwóch takich języków $L(M_1) = L(M_2)$ mamy pewność, że własność P nie jest spełniona, a więc $L(M_1) = L(M_2) \implies \langle M_1\rangle \notin P \wedge \langle M_2\rangle \notin P$

   Zarówno języki opisane w pierwszym jak i drugim punkcie są rekurencyjnie przeliczalne, a zatem własność jest nietrywialna, co oznacza, że L_p jest nierozstrzygalny.

16 Ściągawka z maszyn Turinga

1. Maszyna zwiększająca liczbę znaków o 1 - str. 63

$$111 \rightarrow 1111$$
$$11 \rightarrow 111$$

2. Maszyna podwajająca liczbę znaków - str. 65

$$111 \rightarrow 111111$$
$$11 \rightarrow 1111$$

3. Maszyna mnożąca liczbę znaków przez 4 - str. 66

$$11 \rightarrow 11111111$$
$$111 \rightarrow 111111111111$$

4. dodawanie w systemie unarnym - str.68
5. dodawania jedynki w systemie binarnym - str. 69
6. przesuń napis o jedno pole w prawo - str. 70
7. pomnóż razy dwa liczbę binarną - str. 71
8. usuń wszystkie jedynki w słowie - str. 72
9. $L = \{a^i b^j c^k,\ i < j < k\}$ - str. 73
10. $L = \{w \in \{a,b\}^* :$ sprawdzić czy liczba wystąpień a jest równa liczbie wystąpień b$\}$. Uwaga: symbol $\{a,b\}^*$ oznacza dowolny niepusty ciąg składający się z tych znaków np. $aabbabaa$. - str. 74