JAO seria 3

Bartosz Kucypera

28 maja 2023

Zadanie 3

/* wszystkie oznaczenia jak w treści */
/* jeśli u_i kodem maszyny turinga z treści, to p_i programem wykonywanym przez tą maszynę */
/* zakładamy, że algorytm nie musi być skończony dla każdych danych,
ale zakładamy, że umiemy go zaprogramować w maszynie turinga */

a) Czy język L jest obliczalny?
b) Czy język L jest częściowo obliczalny?
c) Czy dopełnienie języka L jest częściowo obliczalne?

Zauważmy, że jeśli b) i c) są prawdziwe, to a) też jest prawdziwe.
Jeśli L jest częściowo obliczalny to istnieje program, który dla danej pary $< u_1, u_2 >$, jeśli należy ona do L, wykaże to w skończonym czasie. Analogicznie dla dopełnienia L.
Jeśli zastanawiamy się czy dana para $< u_1, u_2 >$ należy do L, możemy uruchomić te programy współbieżnie, np. wykonując po jednym kroku w każdym, na zmiane. Jeśli jeden z nich się zakończy to będziemy wiedzieć czy $< u_1, u_2 >$ należy do L. Zawsze któryś z nich się zakończy, bo każda para $< u_1, u_2 >$, należy albo do L, albo do jego dopełnienia.

Skoro z prawdziwości $[b) \wedge c)]$ wynika prawdziwość a), to jeśli wykażemy, że a) nie jest prawdziwe, oraz że c) jest prawdziwe, to wiemy też, że b) nie jest prawdziwe. Jeśli b) było by prawdziwe, to z wcześniejszej implikacji, a) też by było, czyli sprzeczność.

Nieprawdziwość a)

Załóżmy, że a) prawdziwe. Zauważmy, że w takim razie jesteśmy wstanie rozwiązać nierozwiązywalny problem *stopu*. Pokażmy, że dla każdego programu i każdego wejścia jesteśmy wstanie stwierdzić czy program się zatrzyma.
Niech p_1 rozważanym programem i niech s rozważanym słowem wejściowym.
Niech teraz p_2, i p_3 takimi programami, że $p_2(s) = 0$ i $p_3(s) = 1$. Dla reszty danych wejściowych p_2 i p_3 nie zatrzymują się. Wystarczy teraz sprawdzić czy pary kodów $< u_1, u_2 >$, $< u_1, u_3 >$ należą do L. Robimy to w skończonym czasie, z założenia. Jeśli obie pary należą do L to niewątpliwie p_1 nie zatrzymuje się dla s. Jeśli zatrzymywało by się i $p_1(s) = s'$ to mamy $p_2(s) = p_1(s) = p_3(s)$ czyli $0 = s' = 1$, sprzeczność. Jeśli tylko jedna para należy, to p_1 zatrzymuje się i $s' = 0$ lub $s' = 1$, jeśli żadna para nie należy to p_1 zatrzymuje się i $s' \neq 0$ i $s' \neq 1$.
Opisany algorytm jest zawsze skończony, gdyż, kody u_2 i u_3 są skończone i można je bardzo łatwo wygenerować (jeden if i pętla $while(true)$, przetłumaczone na kod maszyny turinga), natomiast obliczalość sprawdzenia czy pary kodów należą do L, wynika z założonej prawdziwości a).
Czyli faktycznie a) nie może być prawdziwe, bo problem *stopu* jest nierozwiązywalny.

Prawdziwość c)

Skonstruujmy algorytm, który dla każdej pary $< u_1, u_2 >$, jeśli należy ona do dopełnienia L, stwierdzi to w skończonym czasie.
Niech $< u_1, u_2 >$ parą należącą do dopełnienia L. Istnieje, więc takie słowo s, że p_1 i p_2 zatrzymują się na s oraz, $p_1(s) \neq p_2(s)$. Słów w $\{0, 1\}^*$ jest przeliczalnie wiele, więc możemy je ponumerować, np. sortujac leksykograficznie.

Nasz algorytm będzie działał tak, że będziemy na raz mieli "uruchomionych" dużo kopi programów p_1 i p_2 z kolejnymi słowami z $\{0,1\}^*$ jako argumentami wejściowymi. Taśma maszyny turinga jest nieskończona, więc możemy mieć zapisane stany skończenie wielu kopi p_1 i p_2. W każdej turze algorytmu, przechodzimy się po każdej kopi, ładujemy jej stan, wykonujemy kolejny krok programu, zapisujemy jej nowy stan. Dodatkowo na końcu tury tworzymy dwie świerze kopie p_1 i p_2, podając im jako argument wejściowy kolejne słowo z $\{0,1\}^*$. Jeśli jakaś para kopi się zatrzyma (para w sęsie kopie p_1 i p_2 uruchomione na tym samym słowie), i da różne wyniki, to zatrzymujemy cały algorytm i zwracamy, że $< u_1, u_2 >$ należy do dopełnienia L. Powtarzmy przebieg takiej tury dopóki nie napotkamy takiej pary.

Teraz jeśli wiemy, że istnieje takie słowo dla którego p_1 i p_2 dają różny wynik, to nasz algorytm na pewno się zatrzyma. Niech s będzie takim słowem. Zauważmy, że każda tura algorytmu wykonuje się w skończonym czasie. Dodajemy dwie kopie programów i dla skończonej ilości kopi wykonujemy po jednym kroku. Teraz skoro programy kończą się dla s to istnieje skończona liczba kroków po wykonaniu których oba się skończą dla s (max z liczby kroków dla p_1 i liczby kroków dla p_2). Niech k będzie tą liczbą.
Słowo s ma jakiś skończony idenks w posortowaniu leksykograficznym. Niech i będzie tym indeksem. Wiemy teraz, że nasz algorytm zatrzyma się najpóźniej po $k+i$ turach. W i-tej turze dodamy kopie programów ze słowem wejściowym s. Po kolejnych k turach obie kopie się zatrzymają, (będą zatrzymane, jedna mogła skończyć się wcześniej), i ponieważ $p_1(s) \neq p_2(s)$ cały nasz algorytm się zakończy, stwierdzając, że $< u_1, u_2 >$ należy do dopełnienia L.

Algorytm można łatwo zaimplementować w jakimś normalnym języku programowania, czyli jest też to wykonalne na maszynie turinga, czyli dopełnienie L faktycznie jest częściowo obliczalne.

Synteza

Skoro z prawdziwości $[b) \wedge c)]$ wynika prawdziwość $a)$, oraz zachodzi $[\neg a) \wedge c)]$ to niewątpliwe zachodzi, też $[\neg b)]$.