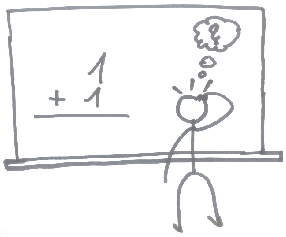
Jest to zapis odczytu wygłoszonego na XXXVIII Szkole Matematyki Poglądowej, *Nieskończoność*, styczeń 2007.

Nieskończone obliczenia

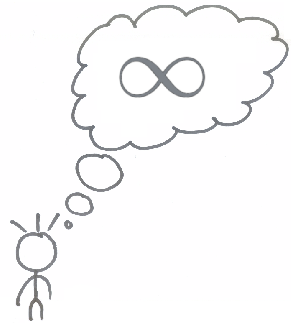
Filip MURLAK, Warszawa



1. Problem z nieskończonością

Informatyka ma z nieskończonością kłopot. Komputer jest skończonym urządzeniem, a użytkownicy życzą sobie, żeby ich programy kończyły się nie tylko w skończonym, ale w jak najkrótszym czasie. Nieskończone obliczenie nie kojarzy się z niczym dobrym.

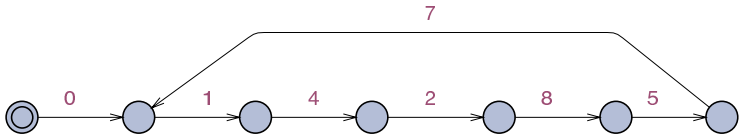
Mimo to, coraz częściej informatycy muszą pisać programy analizujące nieskończone struktury. Rozważmy następujący prosty problem. Mamy dany skończony zbiór zakazanych słów skończonych nad alfabetem $\{a, b\}$. Należy stwierdzić, czy istnieje nieskończone słowo nad alfabetem $\{a, b\}$, które nie zawiera jako podsłowa żadnego ze słów zakazanych. Czasami odpowiedź na to pytanie jest łatwa. Na przykład dla zestawu słów aa, bb, słowo $ababab\ldots$ spełnia nasz warunek. Natomiast dla słów ab, aa, bb takie słowo nieskończone nie istnieje. Dlaczego? Przypuśćmy, że istnieje i zaczyna się od a. Drugą literą nie może być ani a, ani b, więc otrzymujemy sprzeczność. Gdyby pierwszą literą było b, to drugą musiałoby być a i dalej znowu nic by nie pasowało.



My jednak nie szukamy odpowiedzi dla konkretnych przypadków, tylko ogólnej metody. Chcemy stworzyć komputerowy algorytm rozwiązujący ten problem dla każdego zbioru zabronionych podsłów. W tym celu będziemy musieli nauczyć komputer myśleć o nieskończonych słowach.

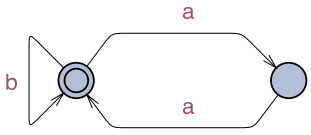
2. Nieskończone, ale regularne

Skończonych reprezentacji nieskończonych napisów używa każdy z nas. Na przykład nieskończony napis $0,3333333\ldots$ reprezentujemy zwyczajowo jako $\frac{1}{3}$, a $0,1428571428\ldots$ jako $\frac{1}{7}$. Zauważmy, że oba powyższe napisy składają się z powtarzającego się w nieskończoność skończonego ciągu cyfr. Stąd, czasami stosujemy inną reprezentację tych napisów: $0,(3)$ i $0,(142857)$. Notacja $0,(142857)$ jest swojego rodzaju przepisem na napis $0,1428571428\ldots$. Przepis mówi: „Napisz 0, potem przecinek, potem 142857, a potem jeszcze raz 142857, i jeszcze raz, i tak dalej”. Przepis ten można zilustrować następującym grafem.



Napis $0,1428571428\ldots$ generujemy startując ze stanu oznaczonego podwójnym kółkiem i poruszając się zgodnie ze strzałkami.

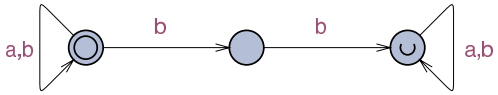
Łatwo można sobie wyobrazić grafy, które generują więcej niż jedno słowo. Wystarczy, żeby z pewnych wierzchołków wychodziło kilka strzałek. Na przykład graf



generuje nieskończone ciągi, które mają parzyste ilości liter a między kolejnymi wystąpieniami liter b.

Właśnie takich grafów będziemy używali do reprezentowania zbiorów nieskończonych słów. Niestety dotychczas opisana wersja jest trochę za słaba na nasze potrzeby. W sformułowaniu zadania, które chcemy rozwiązać, pojawia się na przykład własność „zawieranie podsłowa w”. Nie jest trudno pokazać, że zbiór słów zawierających podsłowo w nie daje się wygenerować w tym prostym modelu. Aby objąć ten przypadek musimy wzmocnić nasz model. W grafie dodatkowo wyróżnimy pewne wierzchołki, nazwijmy je *ulubionymi*

wierzchołkami. Za słowa generowane przez grafy z wyróżnionymi wierzchołkami ulubionymi będziemy uznawali jedynie te, które powstają na ścieżkach przechodzących nieskończenie często przez wierzchołki ulubione. Teraz możemy już łatwo generować zbiory słów zawierających w dla dowolnego w. Na przykład zbiór słów zawierających podsłowo bb jest generowany przez następujący graf:



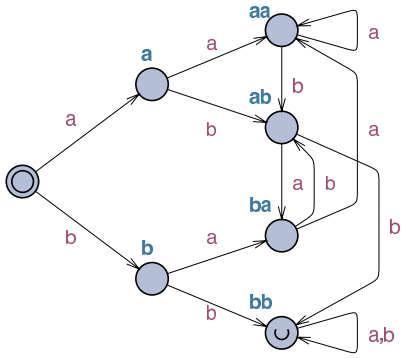
Rzeczywiście, najpierw możemy wygenerować dowolnie długi prefiks, ale żeby odwiedzić stan ulubiony (oznaczony literą U), musimy w końcu wygenerować kolejno dwie litery b. Ciąg dalszy słowa znowu jest dowolny. Grafy dla innych słów w są bardzo podobne do powyższego.

Skoro potrafimy już generować słowa nieskończone zawierające ustalone podsłowo w, to może udałoby się wygenerować zbiór słów nie zawierających żadnego słowa zabronionego. Gdyby faktycznie nam się to udało, to aby rozwiązać problem, od którego wyszliśmy, wystarczyłoby sprawdzić, czy nasz graf generuje jakiekolwiek słowo. To ostatnie zadanie jest jednak bardzo łatwe: wystarczy sprawdzić, czy z wierzchołka startowego da się dojść do wierzchołka ulubionego u, z którego istnieje ścieżka powrotna do u.

Spróbujmy podejść do problemu etapami. Najpierw skonstruujemy graf generujący słowa nie zawierające ustalonego podsłowa zabronionego.

3. Determinizacja i dopełnienie

Graf generujący słowa zawierające bb miał pewną specyficzną cechę. Dla każdego nieskończonego słowa istniało całe mnóstwo ścieżek, które były zgodne z tym słowem, ale tylko dla słów zawierających w istniała ścieżka przechodząca nieskończenie wiele razy przez stan ulubiony. Powiemy, że graf jest *deterministyczny*, jeśli dla każdego słowa istnieje dokładnie jedna ścieżka zgodna z tym słowem. Spróbujmy skonstruować deterministyczny graf generujący słowa zawierające w. Żeby stwierdzić, że już wypisaliśmy słowo w wystarczy pamiętać końcówkę generowanego słowa o długości równej długości w. Dla $w = bb$ graf wygląda tak:



Wierzchołkiem ulubionym jest oczywiście wierzchołek oznaczony literami bb.

Po co skonstruowaliśmy taki graf? Jest przecież dużo większy i bardziej skomplikowany niż wcześniejsza, niedeterministyczna wersja. Okazuje się, że bardzo prosto można przerobić go na graf generujący słowa *nie zawierające* bb. W tym celu wystarczy jako wierzchołki ulubione wybrać aa, ab i ba. Istotnie, każda ścieżka, która nie przechodzi nieskończenie wiele razy przez bb, musi przechodzić nieskończenie wiele razy przez któryś z wierzchołków aa, ab, ba. Ale skoro na każde słowo przypada dokładnie jedna ścieżka, to tym właśnie ścieżkom odpowiadają słowa niezawierające bb. Tak jak poprzednio, uogólnienie tej konstrukcji na dowolne słowa pozostawiamy Czytelnikom.

Na koniec warto zauważyć, że nie każdy zbiór generowany przez jakiś graf jest generowany przez graf deterministyczny. Mimo to, nasz model jest zamknięty na

dopełnienie: jeśli istnieje graf generujący pewien zbiór, to istnieje również graf generujący jego dopełnienie. Dowód tego faktu jest trudny, a o jego konsekwencjach będzie jeszcze mowa.

4. Przecięcie

Ostatnim etapem rozwiązania zadania jest skonstruowanie grafu generującego zbiór słów niezawierających żadnego ze słów $w_1, w_2, \ldots, w_n$. Posiadamy już grafy generujące zbiory słów nie zawierających jednego słowa, wystarczy z nich zrobić graf generujący przecięcie tych zbiorów. Własność zamknięcia na przecięcia – podobnie jak zamknięcia na dopełnienie – przysługuje wszystkim zbiorom generowanym przez grafy i tym razem udowodnimy ogólne stwierdzenie.

Pokażemy jak z grafów G i H generujących zbiory L i M zrobić graf generujący $L \cap M$. Intuicyjnie, musimy sprawdzać jednocześnie dwie własności. Służy do tego konstrukcja grafu produktowego $G \times H$. Wierzchołkami grafu $G \times H$ są pary (u, v), takie że u jest wierzchołkiem G, a v jest wierzchołkiem H. Między wierzchołkami (u, v) i (u', v') jest krawędź z etykietą a jeśli istnieją krawędzie z etykietą a między u i u' (w G) oraz między v i v' (w H). Podobnie dla etykiety b. Stanem startowym jest para stanów startowych oryginalnych grafów. Ścieżki w grafie $G \times H$ odpowiadają parom ścieżek w G i H zgodnym z jednym słowem nieskończonym.

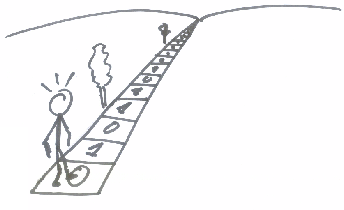
Wydaje się, że jesteśmy blisko rozwiązania. Pozostaje tylko wybrać wierzchołki ulubione w $G \times H$ – i tu pojawia się problem. Aby uzyskać przecięcie zbiorów, powinniśmy zobaczyć nieskończenie wiele wierzchołków o pierwszej składowej ulubionej i nieskończenie wiele wierzchołków o drugiej składowej ulubionej. Jeśli za wierzchołki ulubione uznamy pary (u, v), takie że u jest ulubiony w G, lub v jest ulubiony w H, to nasz graf będzie generował nie przecięcie, ale sumę zbiorów: ścieżka w $G \times H$ przechodząca przez nieskończenie wiele ulubionych stanów odpowiada parze ścieżek, z których *co najmniej* jedna przechodzi przez nieskończenie wiele wierzchołków ulubionych. Spróbujmy inaczej: za wierzchołki ulubione przyjmijmy te pary, których oba komponenty są ulubione.Tym razem nie wygenerujemy wszystkich słów z przecięcia. Jeśli jakaś ścieżka w G przechodzi przez wierzchołki ulubione w krokach parzystych, a zgodna z tym samym słowem nieskończonym ścieżka w H – w krokach nieparzystych, to odpowiadająca jej ścieżka w grafie produktowym nie przejdzie przez żaden wierzchołek ulubiony!

Sztuczka, która pozwala ominąć powyższe trudności polega na rozważeniu dwóch kopii grafu produktowego i rejestrowaniu stanów ulubionych pochodzących od G i od H na zmianę. Niech kopie grafów produktowych nazywają się $(G \times H)_G$ i $(G \times H)_H$. W pierwszej kopii za ulubione uznamy te wierzchołki, których pierwsza współrzędna jest ulubiona, a w drugiej – te których druga współrzędna jest ulubiona. Teraz powinniśmy tak poprowadzić ścieżki w grafie, żeby po odwiedzeniu wierzchołka ulubionego w jednej kopii, przechodziły do tego samego miejsca w drugiej kopii. Aby to osiągnąć, należy przerobić wszystkie strzałki wychodzące z wierzchołków ulubionych w obu kopiach. Powiedzmy, że $(u, v)_G$ jest ulubiony w $(G \times H)_G$ i że wychodzi z niego strzałka z etykietą a do $(u', v')_G$. Należy odczepić koniec tej strzałki od $(u', v')_G$ i przyczepić do $(u', v')_H$. Podobnie postępujemy ze strzałkami wychodzącymi ze stanów ulubionych w $(G \times H)_H$. Wierzchołkiem startowym nowego grafu niech będzie wierzchołek startowy pierwszej kopii.

W otrzymanym grafie ścieżki przechodzące przez nieskończenie wiele wierzchołków ulubionych przeskakują pomiędzy kopiami $G \times H$. W ten sposób na przemian odwiedzają wierzchołki o pierwszej i drugiej składowej ulubionej. Zatem, jeśli pewne słowo jest generowane przez nasz graf, to jest również generowane przez G i H. Odwrotnie, jeśli słowo jest generowane przez G i H, to ścieżka odpowiadająca parze ścieżek z G i H, będzie nieskończenie często przeskakiwała między $(G \times H)_G$ i $(G \times H)_H$, a zatem będzie nieskończenie często przechodziła przez wierzchołki ulubione (mimo, że wielu odwiedzin w wierzchołkach ulubionych G i H nie zarejestruje).

5. Autorefleksja

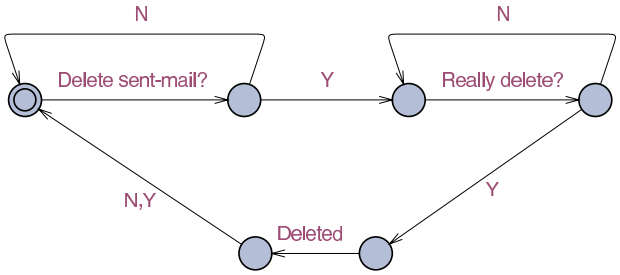
Rozwiązując nasze zagadnienie przy okazji zauważyliśmy, że rodzina zbiorów generowanych przez grafy z wyróżnionymi wierzchołkami jest zamknięta na przecięcie i dopełnienie. Nie jest trudno pokazać, że jest zamknięta również na operację odpowiadającą użyciu kwantyfikatora, a stąd już tylko krok do logiki. Michael Rabin udowodnił w 1969 roku, że zbiory nieskończonych słów generowane przez automaty z wyróżnionymi wierzchołkami to dokładnie te zbiory, które można zdefiniować w pewnym rozszerzeniu logiki pierwszego rzędu, zwanym logiką monadyczną drugiego rzędu. Rozszerzenie polega na tym, że wolno nam kwantyfikować nie tylko po elementach, ale również po ich zbiorach.



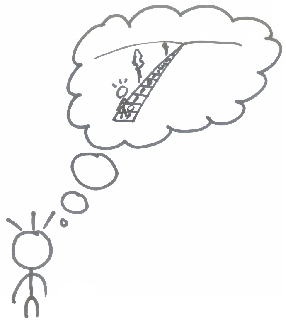
Najważniejszym wnioskiem z twierdzenia Rabina jest to, że dla zadanej formuły w logice monadycznej potrafimy algorytmicznie stwierdzić, czy istnieje choć jedno nieskończone słowo, które posiada opisywaną przez nią własność. Wystarczy w tym celu przerobić formułę na graf, a następnie stwierdzić, czy graf generuje choć jedno słowo – dokładnie tak, jak w przypadku naszego prostego zadania.

Fakt odkryty przez Rabina legł u podstaw automatycznej weryfikacji. Przypuśćmy, że mamy system komputerowy, który ma działać potencjalnie w nieskończoność. W nieskończoność?! Owszem. Pomyślmy na przykład o serwerze poczty elektronicznej – wcale nie chcemy, żeby wykonał zadanie i się wyłączył. Wręcz przeciwnie, chcemy, żeby działał dowolnie długo, potencjalnie w nieskończoność.

Są liczne własności, które taki system powinien spełniać – nie może sie zawieszać, listy powinny dochodzić do adresatów, itd. Okazuje się, że własności tego typu można zapisywać formułami wspomnianej logiki monadycznej. W jaki sposób? Należy popatrzeć na system jak na graf. Wierzchołki reprezentują stany systemu, a strzałki możliwe wydarzenia i reakcje systemu.



Wtedy możemy myśleć o wszelkich potencjalnych wykonaniach systemu, jako o nieskończonych słowach generowanych przez ten graf i wyrażać ich własności w logice monadycznej. Aby stwierdzić, czy istnieje wykonanie systemu, które nie spełnia żądanej własności wystarczy skonstruować graf, który będzie generował przecięcie zbioru wszystkich wykonań systemu ze zbiorem słów *nie* spełniającym żądanej własności, a następnie sprawdzić, czy taki graf generuje choćby jedno słowo. Jeśli tak, to znaleźliśmy niepoprawne wykonanie systemu, a jeśli nie, to system jest zgodny z naszą specyfikacją, a więc poprawny.



Mówiąc o nieskończonych wykonaniach systemów komputerowych wykorzystaliśmy pojęcie nieskończoności w roli zupełnie innej niż dotychczas. Nieskończoność jest już nie tylko cechą problemu, który ktoś chce rozwiązać za pomocą komputera, ale jest aspektem działania samego komputera. Automatyczna weryfikacja zmusza skończony system komputerowy do rozmyślania o swoim własnym, potencjalnie nieskończonym losie.