Do czego komputer nigdy się nie przyda?

Tomasz KAZANA[*]

Czym jest programowanie? Zapewne ilu programistów, tyle opinii. Na potrzeby tego artykułu przyjmijmy taką definicję: programowanie to sztuka formalnego (w ściśle określonym języku) zapisu sposobu postępowania (algorytmu) w celu rozwiązywania określonego problemu dla konkretnych danych. A więc programista tworzy *przepis*, który żmudnie wykonywany na krzemowych bramkach procesora, pozwala komputerowi odpowiadać na różnorodne pytania. Chcemy szybko obliczyć n-tą liczbę Fibonacciego? Powiedzmy komputerowi, *jak* ma to robić, tzn. napiszmy odpowiedni program. Wówczas możemy bez trudu wyznaczyć, na przykład, setny wyraz tego ciągu. Oczywiście, od wyboru algorytmu, a więc od naszej inwencji, zależy, jak długo takie obliczenie będzie trwać. Jednak nie to zagadnienie zamierzamy w tym artykule rozważać. Chcemy pokazać coś dużo bardziej zaskakującego. Otóż istnieją problemy, dla których nie istnieje żadna metoda postępowania! To znaczy, że nie da się napisać programu, który by je rozwiązywał. Zaznaczmy to wyraźnie: to nie tak, że jeszcze nikt nie wpadł na pomysł, jak taki problem rozwiązać. Po prostu można w ścisły sposób udowodnić, że nigdy nikomu się to nie uda. Problemy tego typu nazywamy nierozstrzygalnymi. Kilka z nich spróbujemy w tym artykule naszkicować.

Pierwszym przykładem jest tak zwany problem Posta. Ustalmy jakiś alfabet, dla uproszczenia niech będą to dwie litery a i b. Nasze zadanie określmy następująco: dla danych n par słów nad tym alfabetem, tj. dla wejścia

$$(u_1, w_1), \ldots, (u_n, w_n),$$

chcemy sprawdzić, czy istnieje taki skończony ciąg indeksów $i_1, i_2, \ldots, i_k \in \{1, \ldots, n\}$, że

$$u_{i_1} u_{i_2} \ldots u_{i_k} = w_{i_1} w_{i_2} \ldots w_{i_k}.$$

Na przykład, dla danych $(u_1 = aa, w_1 = aaa)$, $(u_2 = aba, w_2 = b)$, $(u_3 = ababa, w_3 = aaaaab)$ odpowiedź brzmi *tak*, a opisaną własność realizuje ciąg $(1, 2, 1)$, ponieważ $u_1 u_2 u_1 = aa \cdot aba \cdot aa = aaa \cdot b \cdot aaa = w_1 w_2 w_1$.

Okazuje się, że problem Posta, choć dość łatwo go sformułować, jest nierozstrzygalny. Jest on istotny także z jeszcze innego powodu. Otóż często wykorzystuje się go do tego, żeby udowodnić, że inne problemy są nierozstrzygalne.

Mowa tu o tak zwanym sprowadzeniu badanego problemu do jakiegoś innego problemu nierozstrzygalnego. Tzn. aby udowodnić, że problem P jest nierozstrzygalny, możemy zastosować następujące rozumowanie nie wprost. Załóżmy, że P jest rozstrzygalny. Wówczas pokazujemy pewną ustaloną *konstrukcję*, która przekształca dane wejściowe problemu Posta na dane wejściowe problemu P, w taki sposób, aby na podstawie wyniku działania P można było odczytać rozwiązanie problemu Posta (dla tych konkretnych danych). To już oznacza sprzeczność, bo tym samym pokazaliśmy program, który (uruchamiając program rozwiązujący P) potrafi rozstrzygać własność Posta. Ta sprzeczność dowodzi nierozstrzygalności P.

Jest więc trochę „ważniejszym” przykładem niż inne, stąd pokazujemy go na początku.

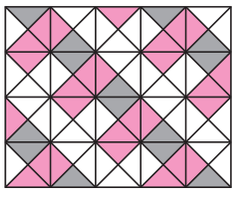
Podobny status mają niektóre problemy NP-zupełne, na przykład problem spełnialności formuł logicznych (SAT).

We współczesnej informatyce teoretycznej badanie problemów nierozstrzygalnych (a raczej: rozstrzyganie, które problemy są, a które nie są rozstrzygalne) ma niebagatelne znaczenie. Wachlarz znanych zagadnień nierozstrzygalnych jest przez to bardzo szeroki. Są to zadania często głęboko teoretyczne i trudne do przejrzystego, nawet powierzchownego omówienia. My jednak spróbujemy przedstawić kilka bardzo prostych do opisania problemów, które okazują się nierozstrzygalne. (Pierwsze dwa zostały zaczerpnięte z prac Davida Harela, wybitnego informatyka izraelskiego.) Przykłady są zróżnicowane, aby Czytelnik wyrobił sobie jakąś intuicję, co może być niemożliwe, a co – tylko trudne.

Problem 1. Klocki

Na wejściu otrzymujemy skończoną liczbę rodzajów kwadratowych klocków, które mają pokolorowane ćwiartki (jak na rysunku 1) – klocków każdego rodzaju jest w zestawie nieskończenie wiele. Chcielibyśmy móc odpowiadać na pytanie, czy mając tylko takie klocki, można nimi pokryć całą płaszczyznę (rys. 2). Przy tym dwa klocki wolno postawić obok siebie, gdy stykają się ćwiartkami w tym samym kolorze. W ogólności komputer nam w tym nie pomoże, bo to problem nierozstrzygalny...

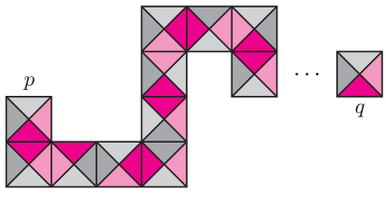
Rys. 1



Rys. 2

Problem 2. Wąż z klocków na półpłaszczyźnie

Wejście tego problemu jest takie samo jak poprzednio. Powiedzmy, że dla rodzajów klocków określonych w wejściu i wskazanych dwóch punktów na kratowanej płaszczyźnie chcielibyśmy sprawdzić, czy istnieje *wąż z klocków* łączący te punkty (rys. 3). Tak opisany problem jest rozstrzygalny. Jednak jeśli nieco zmienimy zasady, tzn. zażądamy, by węże mieściły się w całości w górnej półpłaszczyźnie – nasz problem stanie się nierozstrzygalny.



Rys. 3

[*]doktorant, Instytut Informatyki, Uniwersytet Warszawski

Problem 3. Uniwersalne programy do debugowania

Nierozstrzygalne są wszelkie problemy, które miałyby weryfikować odpowiednio nietrywialne własności programów podanych na wejściu. Nierozstrzygalny jest, na przykład, problem sprawdzania, czy dany program nie ma wycieków pamięci, czy zawsze się kończy, czy daje poprawny wynik (w jakimś określonym sensie) itp. Na dużym poziomie ogólności można o tym myśleć tak, iż nie istnieją (i nie mogą istnieć) żadne uniwersalne programy do automatycznego *debugowania*, czyli usuwania błędów z innych programów.

Problem 4. Dziesiąty problem Hilberta

Nie jest możliwe napisanie programu, który stwierdzałby, czy zadane równanie diofantyczne ma choć jedno rozwiązanie, czy nie.

Równanie diofantyczne to równanie wielomianowe, którego rozwiązania szukamy w zbiorze liczb całkowitych. Najczęściej rozważa się równania diofantyczne z kilkoma niewiadomymi. Przykłady: $x^5 + y^5 = z^5$, $x^2 - 3y^2 = 1$.

Problem ten postawiony został na przełomie XIX i XX w. przez Hilberta. Oczekiwania były raczej pozytywne. Niestety, marzenia prysły za sprawą Jurija Matijasiewicza, który w latach 70. XX wieku jako pierwszy wykazał nierozstrzygalność tego problemu. Uwaga: dla części wejść oczywiście można ten problem rozwiązać. Być może nawet da się napisać program, który zadziała na pewnym ograniczonym podzbiorze danych, chociażby na równaniach liniowych. Nierozstrzygalność oznacza tyle, że nie da się napisać programu zawsze poprawnego, działającego dla każdych możliwych danych. Tu: rozwiązującego każde możliwe równanie diofantyczne, lub choćby stwierdzającego, czy jego rozwiązanie istnieje.

Problem 5. Automatyczne rozstrzyganie hipotez

Nierozstrzygalny jest problem, czy podana na wejściu hipoteza matematyczna jest prawdziwa, czy fałszywa. Tzn. jakąś hipotezę formułujemy ściśle i czekamy na weryfikację. Zwrot *ściśle* oznacza tutaj – w języku jakiejś odpowiednio silnej logiki (nieformalnie: chcielibyśmy, aby hipoteza była wyrażona w dostatecznie bogatym *języku matematycznym*, np. przy użyciu takich pojęć, jak „dla każdego" czy „istnieje").

Okazuje się, że pewne wąskie podproblemy tego zagadnienia są rozstrzygalne. Na przykład, Mojżesz Presburger udowodnił, iż hipotezy z dziedziny arytmetyki, ale te nieużywające mnożenia (a więc samo dodawanie), które da się wyrazić w tzw. języku logiki pierwszego rzędu, można weryfikować automatycznie za pomocą komputera.

Problem 6. Problem stopu

Nierozstrzygalne jest pytanie, czy dany program dla konkretnych danych wejściowych pętli się czy nie. W języku programisty powiedzielibyśmy, że nie da się napisać programu `super-program.c`, który – dostawszy na wejściu jakiś inny program `do-testowania.c` oraz dane wejściowe `dane.in` – stwierdzałby (w skończonym czasie), czy dla danych `dane.in` program `do-testowania.c` kończy się czy nie. Podamy pełny dowód tego twierdzenia.

Dowód. Załóżmy nie wprost, że taki program jednak udało nam się napisać. Tzn. mamy funkcję

bool stop(*program*, *dane*),

która zawsze działa w skończonym czasie i zwraca prawdę wtedy i tylko wtedy, gdy podany *program* dla wejścia *dane* zatrzymuje się. Napiszmy teraz inną funkcję:

```
bool fun(program)
  if stop(program, program) then
    while true do sleep();
  else
    return true;
```

Zauważmy, że w powyższej funkcji jako drugi parametr funkcji stop, a zatem jako dane wejściowe, podany jest sam *program*!

Zastanówmy się, jaki jest wynik działania fun(fun). Każdy program dla określonych danych albo się pętli, albo nie – rozważmy więc dwa przypadki. Najpierw załóżmy, że wykonanie fun(fun) zatrzymuje się. Wówczas, na mocy określenia problemu stopu, stop(fun, fun) zwraca **true**. Jednak wtedy analiza kodu funkcji fun pokazuje, że fun(fun) pętli się, gdyż warunek po **if** jest prawdziwy. Uzyskaliśmy sprzeczność, co oznacza, że fun(fun) musi się pętlić. To jednak (znów patrzymy na instrukcję **if** w funkcji fun) implikuje, że stop(fun, fun) zwraca prawdę, a to, z definicji funkcji stop, oznacza dokładnie tyle, że fun(fun) zatrzymuje się.

W każdym przypadku uzyskaliśmy zatem sprzeczność, więc teza została udowodniona. □

Autor ma świadomość, że ten artykuł w dużej mierze jest tylko namiastką, prezentacją ledwie kilku przykładów, i ma stanowić raczej zachętę do dalszych poszukiwań. Przy okazji ma jednak nadzieję, że udało się, choćby mgliście, dać ogląd, czym się – między innymi – zajmuje współczesna informatyka teoretyczna. Samą tematykę problemów nierozstrzygalnych można oczywiście rozwijać dalej. Ciekawy, być może wart oddzielnego przedstawienia, wydaje się temat hierarchii problemów nierozstrzygalnych. Mówimy tu o takiej hierarchii, w której problemy nierozwiązywalne (tzn. nierozstrzygalne) dzielimy na mniej i bardziej beznadziejne przypadki. Porządek jest tu taki, że problem P_1 jest *trudniejszy* od P_2, jeśli założenie o tym, że P_1 byłby rozstrzygalny, pozwoliłoby hipotetycznie rozstrzygać P_2, a odwrotnie – już nie. Pewnie można by tu mieć wątpliwości, po co w ogóle porównywać problemy, które i tak są niemożliwe do rozwiązania. Z pewnością jest tu jakiś element sztuki dla sztuki (jak w każdej bardzo teoretycznej dyscyplinie nauki), jednak tego typu zagadnienia nie są wyłącznie domeną smakoszy.

Rezultaty z dziedziny nierozstrzygalności to tak zwane twierdzenia o niemożności – zjawisko fascynujące samo w sobie! To znaczy udaje nam się dowieść, że czegoś nie da się rozwiązać. Wskutek tego musimy mieć świadomość, że istnieją problemy, na które nie pomogą ani nasze szkiełka, ani nasze oczy, ani nawet tona krzemu zaprogramowana przez najlepszych algorytmików.