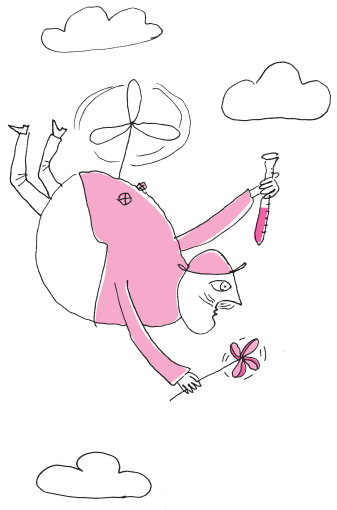
Modelowanie

Wojciech CZERWIŃSKI



Dziś modeluje się prawie wszystko. Przykładowo prognozę pogody tworzy się na podstawie modelu atmosfery. Przestrzeń nad ziemią dzieli się na prostopadłościany szerokości kilku kilometrów, wysokości kilkudziesięciu, może kilkuset metrów; w każdym z nich ustala się, jaka jest temperatura, wilgotność, ciśnienie, prędkość wiatru, jego kierunek i jeszcze wiele innych parametrów. Taki opis sytuacji to stan modelu. Ponadto opierając się na prawach fizyki, ustala się, jak ten stan będzie ewoluował w czasie. To, oczywiście, będzie przybliżenie sytuacji rzeczywistej. Na przykład, liczymy, w jakim stanie model będzie za 12 godzin, dobę, dwie. Często takie obliczenia wymagają wielkiej mocy obliczeniowej, szczególnie jeśli chcemy zrobić to dokładnie, jak np. w przypadku prognozy ICM (`meteo.pl`).

Modelowanie stosuje się również w wielu innych przypadkach, gdy chcemy przewidzieć w sposób przybliżony, co będzie w przyszłości. Konstruuje się modele opisujące ceny akcji na giełdzie, wielkość pokrywy lodowej w Arktyce, erupcje wulkanów, wielkość populacji danego kraju czy świata, ewolucję chorób na danym terenie itd. We wszystkich tych sytuacjach chcemy czegoś dowiedzieć się o przyszłości ważnego dla nas zjawiska czy procesu, ale jest on na tyle skomplikowany, że nie jesteśmy w stanie zrobić tego dokładnie. Dlatego idziemy na kompromis i zajmujemy się pewnym przybliżeniem rzeczywistości. Będzie to, co prawda, przybliżenie, ale za to będziemy w stanie z nim pracować i rzeczywiście obliczać, jak się ono zachowa w przyszłości. Modele rzeczywistości mogą być dosyć dokładne, wtedy jednak (ze względu na złożoność) ciężko analizować ich własności i obliczać, jak będą ewoluowały. Ale za to, jeśli to już zrobimy, będziemy mieli dobre przybliżenie tego, co się stanie *naprawdę*. Mogą być też mniej dokładne, wtedy będą zachowywały się istotnie inaczej niż rzeczywistość. W zamian za to będzie nam je łatwiej analizować, a analiza przyniesie pewne wnioski, które choć częściowo pozwolą zrozumieć badane zjawisko.

Czasem model, który tworzymy, wcale niekoniecznie ma wiele wspólnego z rzeczywistością, ale konstruujemy go tak, by dawał dobre wyniki dla niektórych eksperymentów, mając nadzieję, że przyda nam się on do przewidywania wyników innych eksperymentów. Ten sposób patrzenia na modelowanie bliższy jest fizyce, gdzie siłą rzeczy nie znamy *istoty rzeczywistości*, więc możemy jedynie konstruować model tak, by dobrze przybliżał znane nam obserwacje. Dobrymi przykładami są tu: kopernikański model układu słonecznego czy model standardowy opisujący świat w mikroskali.

Skoro modelowanie stosuje się w ekonomii, meteorologii, biologii, fizyce, socjologii, to dlaczego by nie zastosować go w informatyce do lepszego badania zachowań programów? Istotnie się to robi i modele programów mają sporo różnych zastosowań. W informatyce znamy zasady kierujące działaniem programów, natomiast nie umiemy przewidywać, do czego doprowadzą. Dlatego stosujemy podejście, w którym budowany model odzwierciedla rzeczywistość w sposób przybliżony. Jednym z ważnych zastosowań jest *automatyczna weryfikacja programów*, czyli automatyczne wykrywanie błędów w programach. Oczywiście, nie jest ono zupełnie automatyczne, ale jego część wykonuje się automatycznie. Chcemy wykryć, czy nasz program może potencjalnie wykonać pewnego rodzaju błędny przebieg. Powiedzmy, pytamy, czy może on w pewnym momencie próbować podzielić coś przez 0. W tym celu konstruuje się model programu M i opisuje w precyzyjny sposób, co rozumiemy przez błędny przebieg. W naszej sytuacji byłby to ciąg instrukcji, który kończy się instrukcją: podziel coś przez 0. Takie błędne ciągi instrukcji charakteryzuje się przy użyciu formuł pewnych logik. Powiedzmy, że formuła logiczna ϕ opisuje ciągi instrukcji kończące się podzieleniem przez 0, to znaczy wylicza się do *prawdy* przy podstawieniu takich ciągów, a do *fałszu* przy pozostałych. Teraz wystarczy już tylko sprawdzić, czy w modelu M możliwe jest obliczenie, dla którego formuła ϕ *zwraca* wartość *prawda*. To podejście nazywa się z angielska *model checking*, czyli sprawdzanie modelu, i cieszy się dużym sukcesem komercyjnym. Dla przykładu, firma Intel wydaje na rozwój tej techniki miliony dolarów.

Rozwiązanie zadania M 1576.
Rozważmy dowolny czworościan $ABCD$ w którym K, L, M, N są punktami styczności sfery wpisanej i ścian leżących odpowiednio naprzeciw wierzchołków A, B, C, D.

Wyznaczymy miary kątów $\sphericalangle ABN$ oraz $\sphericalangle BAN$ w zależności od miar kątów wewnętrznych ścian czworościanu (które są dane, gdy dana jest siatka).

Zauważmy, że na mocy cechy bok–bok–bok, pary: ABM i ABN, ACN i ACL, ADL i ADM to pary trójkątów przystających; oznaczmy kąty wewnętrzne przy wierzchołku A w poszczególnych z nich odpowiednio przez β, γ, δ. Wówczas

$$\beta+\gamma=\sphericalangle BAN+\sphericalangle CAN=\sphericalangle BAC,$$
$$\gamma+\delta=\sphericalangle CAL+\sphericalangle DAL=\sphericalangle CAD,$$
$$\delta+\beta=\sphericalangle DAM+\sphericalangle BAM=\sphericalangle DAB,$$

skąd wniosek, że

$$\sphericalangle BAN=\beta=$$
$$=\tfrac{1}{2}(\sphericalangle BAC+\sphericalangle DAB-\sphericalangle CAD).$$

Podobnie uzyskujemy równość

$$\sphericalangle ABN=\tfrac{1}{2}(\sphericalangle ABC+\sphericalangle DBA-\sphericalangle CBD).$$

Ponieważ dodawanie i odejmowanie kątów, a także prowadzenie dwusiecznej są wykonalne konstrukcyjnie, więc powyższe równości bezpośrednio przenoszą się na żądaną konstrukcję punktu styczności ze ścianą ABC. Dla pozostałych ścian konstrukcja jest w pełni analogiczna.

Rozwiązanie zadania M 1578.
Trójkąt równoboczny $A'B'C'$ o wierzchołkach w węzłach nazwijmy *czapeczką*, jeżeli $AB \parallel A'B'$ oraz punkty C i C' leżą po tej samej stronie prostej $A'B'$.

Każdemu trójkątowi T, spełniającemu warunki zadania, przyporządkujmy *najmniejszą* zawierającą go czapeczkę. Precyzyjniej: jeżeli T jest czapeczką, to przyporządkowujemy mu siebie samego, natomiast w przeciwnym przypadku — czapeczkę ograniczoną prostymi: równoległą do AB przechodzącą przez wierzchołek T leżący najbliżej AB, równoległą do BC przechodzącą przez wierzchołek T leżący najbliżej BC oraz równoległą do CA przechodzącą przez wierzchołek T leżący najbliżej CA.

Każdej czapeczce o boku $k = 1, \ldots, n$ zostało w ten sposób przyporządkowanych dokładnie k trójkątów spełniających warunki zadania. Co więcej, liczba czapeczek o boku k jest równa

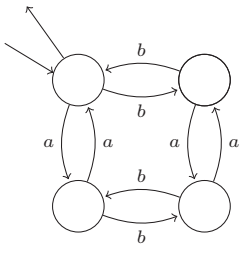
$$1 + 2 + \ldots + (n-k+1) = \binom{n-k+2}{2}.$$

Wobec tego szukana liczba trójkątów równobocznych to

$$\sum_{k=1}^{n} k\binom{n-k+2}{2} = \binom{n+3}{4},$$

przy czym równość ta wynika z następującej obserwacji. Czteroelementowy podzbiór zbioru $\{0, 1, \ldots, n+2\}$, którego drugim co do wielkości elementem jest $k \in \{1, \ldots, n\}$, można wybrać na dokładnie $k\binom{n-k+2}{2}$ sposobów (wybierając najmniejszy element spośród $\{0, \ldots, k-1\}$ oraz dwa większe od k spośród $\{k+1, \ldots, n+2\}$). Sumując po wszystkich możliwych k, uzyskujemy liczbę sposobów wyboru 4 spośród $n+3$ elementów.

Uwaga. Zachęcamy Czytelnika do znalezienia naturalnej bijekcji między obiektami zliczanymi w tym zadaniu (dla trójkąta o boku n) oraz w poprzednim (dla trójkąta o boku $n+1$).



Automat rozpoznający język słów nad alfabetem $\{a, b\}$, które zawierają parzyście wiele liter a oraz parzyście wiele liter b.

Jednak korzyść z analizowania modeli programów nie jest tylko komercyjna, ale również teoretyczna. Jednym z modeli programów, bardzo dokładnym i skomplikowanym, jest maszyna Turinga. Dzięki temu, że ustalony został precyzyjny model programu, można zdefiniować, co dokładnie oznacza, że dany problem da się rozwiązać w czasie wielomianowym albo że jest w klasie NP, wielokrotnie wspominanej na łamach *Delty*. Czyli właściwe definicje modeli pomagają nam lepiej zrozumieć świat programów, powiązania między różnymi pojęciami, a często także pozwalają opracować szybkie algorytmy.

Żeby przejść do konkretów, przyjrzyjmy się najprostszemu modelowi programu, mianowicie *automatowi skończonemu*. Dowolny program w każdej chwili swojego działania jest w jakimś stanie pamięci. Przykładowo, program szukający maksimum w tablicy o rozmiarze 100 może być w stanie: jestem w 37 komórce tablicy, do tej pory największa liczba to 178, a aktualnie spoglądam na liczbę 110. Przy praktycznym założeniu, że każdy komputer ma skończoną pamięć, liczba możliwych stanów programu jest również skończona. Natomiast obliczenie programu polega na przechodzeniu pomiędzy tymi stanami według pewnego ustalonego zestawu reguł (ten zestaw, oczywiście, zależy od programu). Nasz program szukający maksimum może przejść np. do stanu: jestem w 38 komórce tablicy, największa znaleziona liczba to 178, aktualnie widzę liczbę 88. Dodatkowo zmiany stanu programu mogą być różnych typów – w naszym przykładzie jest to uaktualnienie maksimum albo przejście dalej w prawo w tablicy. Model powinien to móc uwzględniać. Dodatkowo model powinien zawierać informację, w jakim stanie program zaczyna swoje działanie oraz w jakich stanach się kończy. Sprecyzujmy teraz opisane intuicje i zdefiniujmy dokładnie, czym jest automat skończony.

Deterministyczny automat skończony $\mathcal{A}$ nad skończonym alfabetem Σ składa się ze zbioru *stanów* Q, wyróżnionego *stanu początkowego* $q_0 \in Q$, zbioru *stanów akceptujących* $F \subseteq Q$ oraz zbioru *tranzycji* $T \subseteq Q \times \Sigma \times Q$. Jeśli dla pewnych stanów $p, q \in Q$ oraz litery $a \in \Sigma$ zachodzi $(p, a, q) \in T$, to piszemy $p \xrightarrow{a} q$. *Biegiem* automatu $\mathcal{A}$ po słowie $w = a_1 \cdots a_n$ składającym się z liter $a_i \in \Sigma$ nazwiemy ciąg takich stanów $\rho = p_0, \ldots, p_n$, że $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} p_n$. Jeśli dodatkowo p_0 jest stanem początkowym, a p_n pewnym stanem akceptującym, to bieg ρ nazwiemy *akceptującym* i powiemy, że słowo w jest akceptowane przez automat. Natomiast *językiem* rozpoznawanym przez automat $\mathcal{A}$ nazywamy zbiór wszystkich słów przez niego akceptowanych. Intuicyjnie język automatu $\mathcal{A}$, oznaczany $L(\mathcal{A})$, opisuje zbiór wszystkich ciągów akcji programu, który modeluje dany automat. Jeśli słowo $w = a_1 \cdots a_n$ należy do $L(\mathcal{A})$, to oznacza, że w programie jest pewien przebieg, w którym po kolei program wykonuje akcje typu a_1, a_2 itd. aż na końcu wykonuje akcję typu a_n. Stan natomiast intuicyjnie reprezentuje to, co w danym momencie trzeba pamiętać o prefiksie słowa, żeby pod koniec stwierdzić, czy całe słowo należy do języka, czy też nie.

Okazuje się, że automaty skończone przydatne są nie tylko do modelowania działania programów, ale są po prostu bardzo eleganckim pojęciem, które jest użyteczne w wielu kontekstach. Przykładowo, znajdują zastosowanie przy sprawdzaniu, czy model M spełnia formułę ϕ, albo przy analizie algorytmów operujących na znanych niektórym Czytelnikom wyrażeniach regularnych. Wyrażenia regularne są zbudowane przy użyciu sumy (oznaczanej $+$), konkatenacji (oznaczanej $\cdot$) i gwiazdki (oznaczanej *). Jeśli r i s opisują języki L_r oraz L_s, to $r + s$ opisuję sumę $L_r \cup L_s$, $r \cdot s$ opisuje język słów, których pierwsza część należy do L_r, a druga do L_s, natomiast r^* opisuje język słów, które dadzą się podzielić na skończenie wiele części, z których każda należy do L_r. Przykładowo wyrażenie $(a + b) \cdot (b + c)$ opisuje język czterech słów, których pierwsza litera to a lub b, a druga to b lub c. Natomiast wyrażenie $b^* \cdot c$ opisuje język nieskończenie wielu słów, które zaczynają się pewną liczbą liter b, a kończą się literą c. Okazuje się, że język automatu na rysunku również da się opisać wyrażeniem regularnym:

$$\big(a \cdot a + b \cdot b + (a \cdot b + b \cdot a)(a \cdot a + b \cdot b)^*(a \cdot b + b \cdot a)\big)^*,$$

zachęcamy Czytelników do uzasadnienia tego faktu. Co ciekawe, wyrażenia regularne opisują tak naprawdę dokładnie te same języki, co automaty skończone. To jednak temat na zupełnie inną opowieść.