Matematyka dyskretna dla informatyków (2)

1 Logika

1.1 Historia

Logika jest jedną z najstarszych nauk. Traktowana przez Greków, jako podstawa wszelkich nauk — dostarczała przecież języka do przeprowadzania poprawnych wywodów naukowych — uzyskała swoją pierwszą formalizację w postaci podanej przez Arystotelesa, która z niewielkimi zmianami dotrwała do XX wieku w zasadzie w czystej formie.

Logika zajmuje się formalizacją procesu wnioskowania. Przez określenie sztywnych praw, które można stosować formułuje się teorie logiczne stwierdzające prawdziwość pewnych faktów w różnych dziedzinach. Musimy zdać sobie sprawę, że w konkretnych zastosowaniach zawsze mamy do czynienia z dwoma poziomami logicznymi: aparatem logiki klasycznej, który umożliwia dokonywanie pewnych operacji na zdaniach upraszczających rozumowanie oraz z opisem świata, który chcemy zbadać i najczęściej ta część dodana do logiki klasycznej tworzy konkretną teorię.

Tematem tego wykładu będzie przedstawienie klasycznego rachunku zdań, czyli wspólnego fragmentu wszelkich teorii.

1.2 Logika w informatyce

1.2.1 Poprawność programów

Logika zawsze odgrywała dużą rolę w informatyce, a ostatnio jej rola wzrasta. Wzrost znaczenia logiki wiąże się przede wszystkim z potrzebą zapewnienia poprawności oprogramowania. Ogromna rola, jaką programy komputerowe pełnią w naszym życiu, z jednoczesnym powszechnym przekonaniem o ich wysokiej zawodności powodują, że informatycy starają się uzasadnić poprawność swoich wyrobów metodami formalnymi, co najczęściej sprowadza się do wyboru logiki, jako podstawowego narzędzia.

1.2.2 Sztuczna inteligencja

Naturalnym zastosowaniem logiki jest sztuczna inteligencja — dział informatyki zajmujący się próbą odtworzenia schematów myślenia, którymi kieruje się ludzki mózg. Założenie, że mózg działa zgodnie z prawami logiki powoduje, że szuka się formalizmów logicznych odzwierciedlających procedy kojarzenia i wnioskowania.

1.2.3 Bazy danych

Naturalnym językiem specyfikacji żądań w bazach danych jest język logiki. Zapytania formułuje się w formie zdań logicznych, których spełnienia oczekujemy od interesujących nas obiektów.

1.2.4 Języki funkcyjne i programowanie w logice

Poza paradygmatem imperatywnym istnieją dwa popularne paradygmaty programowania oparte na logice. Pierwszy z nich, to paradygmat *programowania funkcyjnego*, stworzony przez wybitnego logika Johna McCarthy'ego i zrealizowany najpierw pod koniec lat 50-tych w postaci języka LISP, a poźniej w wersjach nieco bardziej nowoczesnych języków takich jak Linda, Scheme, ML, EML, Hope. W programowaniu funkcyjnym definiuje się funkcje, których zastosowanie do konkretnych wartości daje efekt w postaci pożądanego obliczenia. Napisanie programu sprowadza się do zdefiniowania odpowiedniej funkcji. Argumentami funkcji mogą być inne rozmaite złożone obiekty, w szczególności inne funkcje, w tym także funkcja aktualnie definiowana (rekurencja). Operowanie na funkcjach wymaga wzniesienia się na nieco wyższy poziom abstrakcji i przyzwyczajenia się do niestandardowej notacji rachunku lambda wymyślonej w latach 30-tych przez innego logika — Alonso Churcha.

Programowanie w logice, to zupełnie inny paradygmat. Napisanie programu w logice polega na zdefiniowaniu pewnego świata obiektów i zależności między nimi. W szczególności definiuje się reguły wnioskowania pozwalające preprowadzić rozumowanie o tym konkretnym świecie. Reguły te podaje się w postaci szczególnego typu formuł zdaniowych zwanych *klauzulami Horna*. Po zdefiniowaniu świata napisanie programu sprowadza się do napisania odpowiedniego stwierdzenia w postaci zdania logicznego (zazwyczaj to stwierdzenie jest negacją interesującej nas własności). Wykonanie programu polega na podjęciu próby udowodnienia prawdziwości tego zdania. W trakcie dowodzenia program próbuje dopasować elementy występujące w tym zdaniu do wiedzy, która zadana jest przez opis świata. Jeżeli twierdzenie okaże się nieprawdziwe, to program stwierdzi istnienie kontrprzykładu na prawdziwość tego twierdzenia i kontrprzykład ten będzie stanowił wynik obliczeń naszego programu (ze względu na poprzednie zanegowanie będzie on pozytywnym przykładem interesującego nas rozwiązania). Programowanie w logice zostało zrealizowane w końcu lat 70-tych w postaci języka Prolog i do dziś rośnie popularność tego narzędzia.

Oba te paradygmaty znajdują duże zastosowanie w sztucznej inteligencji.

1.3 Zdania

Ciekawą dyskusję na temat tego, co zdaniem jest a co nie można przeczytać w podręczniku ,,Matematyka dyskretna" Rossa i Wrighta. Na nasz użytek przyjmijmy, że zdaniem jest stwierdzenie, o którym można powiedzieć, czy ma wartość prawda, czy fałsz. Zdania proste można łączyć w zdania złożone za pomocą spójników logicznych. Spośród wielu różnych spójników najpopularniejszymi są $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ oznaczające kolejno negację, koniunkcję, alternatywę, implikację i równoważność. Aby prowadzić wnioskowanie na metapoziomie wprowadzimy *zmienne zdaniowe*, oznaczane literami $p, q, r, \ldots$, które będą oznaczały dowolne zdania. Uwaga: na tym metapoziomie też używamy intuicyjnego języka logiki. Żeby nie wprowadzać niejasności będziemy wyraźnie rozróżniali symbole metajęzyka i języka. Na poziomie metajęzyka będziemy używali słów języka polskiego ,,nie, i, lub" na oznaczenie związków pomiędzy stwierdzeniami o zdaniach, jak również — ze względu na krótkość zapisu — będziemy używać symboli $\Rightarrow, \Leftrightarrow$ na oznaczenie odpowiednio wynikania, które zawsze zachodzi oraz równoważności zawsze prawdziwej. Ta druga zależnoć określa semantyczną równoważność dwóch zapisów. Oznacza ona, że takie dwa napisy zawsze mają tę samą wartość logiczną. W odróżnieniu od zapisu $p \leftrightarrow q$, które oznacza zdanie p *jest równoważne* q, które to zdanie zbudowane z dwóch części p i q połączonych spójnikiem $\leftrightarrow$, może być prawdziwe, a może być fałszywe, w zależności od tego czym jest p, a czym jest q, zapis $p \Leftrightarrow q$ oznacza stwierdzenie na metapoziomie, że zdania p oraz q są zawsze równoważne. Często w tej formie wyraża się *tautologie*, czyli zdania zawsze prawdziwe (w ogólnym przypadku niekoniecznie muszą mieć one postać równoważności). Symbole $\Rightarrow$ i $\Leftrightarrow$ nie są zatem spójnikami zdaniotwórczymi na poziomie języka logiki i nie tworzą nowego zdania poprzez odpowiednie połączenie obu stron. Należy je traktować jako stwierdzenia o wzajemnej zależności formuł zdaniowych występujących po obu stronach.

Przyzwyczajmy się do myśli, że nie ma jednej logiki. Różne logiki tworzy się po to, żeby opisywać konkretne światy. Świat liczb naturalnych, świat liczb rzeczywistych, świat obiektów w pewnej bazie danych, świat obiektów pewnej rzeczywistości wirtualnej, świat norm prawnych obowiązujących w danym państwie, świat wartościowań zmiennych w trakcie wykonywania programu komputerowego, świat stanu urządzeń w trakcie działania systemu operacyjnego są typowymi przykładami stosowania konkretnych logik, z których każda odwołuje się do konkretnych własności charakterystycznych dla danej dziedziny. Niektóre zapisy mogą być zdaniami prawdziwymi w jedny modelu, fałszywymi w drugim, a w ogóle mogą nie być zdaniami w trzecim. Na przykład formuła zdaniowa $a < b \rightarrow a^2 < b^2$ jest zawsze prawdziwa w modelu liczb naturalnych, czasami prawdziwa a czasami fałszywa — w zależności od tego czym są a i b w modelu liczb rzeczywistych i w końcu zapisem nie dającym się w ogóle zinterpretować w świecie norm prawnych.

Postaramy się natomiast podać zestaw praw obowiązujących w każdej interesującej nas logice.

Wprowadźmy następującą terminologię:

- *Zdanie atomowe*, to jest zapis, który w danej dziedzinie ma wartość prawda lub fałsz. Zdaniem atomowym są zawsze dwie stałe *true* i *false*, których wartości niezależnie od kontekstu są zawsze równe odpowiednio *prawda* i *fałsz.* Przykłady zdań atomowych, to:
  - Teraz pada deszcz.
  - $2 + 3 = 5$
  - $1 + 1 = 0$
  - $x > 0$
- *Zdanie* jest to wyrażenie złożone ze zdań atomowych połączonych spójnikami logicznymi. Przykłady zdań, to:
  - Jeżeli teraz pada deszcz, to ulica jest mokra.
  - $2 + 3 = 5 \leftrightarrow 3 + 2 = 5$
  - $1 + 1 = 0 \rightarrow (1 * 1 = 1 \wedge 1 - 1 = 0)$
  - $x > 0 \Leftrightarrow x^2 > 0$

- *Formuła zdaniowa* jest to zdanie, zmienna zdaniowa lub napis złożony ze zdań i zmiennych zdaniowych połączonych spójnikami logicznymi. Wartość formuły zdaniowej może zależeć od interpretacji zmiennych zdaniowych, czyli od tego, czy konkretne zmienne będą reprezentowały zdania prawdziwe, czy fałszywe. Przykłady formuł to:
  - $p \wedge q \to p \vee q$
  - $x > 0$
  - $1 + 1 = 0$

Zwróćmy uwagę na to, że zdania i formuły zdaniowe są napisami. Ich wartość najczęściej zależy od interpretacji w konkretnej dziedzinie. Interpretacja taka polega na wyjaśnieniu, co konkretnie oznaczają wszystkie ogólnie zapisane symbole i jak wartościowane są zmienne. Wartościowanie zmiennych logicznych, to przypisanie każdej zmiennej w formule jednej z dwóch wartości: *true* lub *false*. W formułach bardzo często występują też *termy*, czyli napisy, które oznaczają pewne wyrażenia interpretowane w określonej dziedzinie.

1.4 Rachunek zdań

Określimy teraz wartości spójników logicznych. Jednoargumentowy operator $\neg$, zwany *negacją* neguje wartość: z prawdy robi fałsz, a z fałszu — prawdę. Zatem $\neg p = true$ gdy $p = false$ oraz $\neg p = false$, gdy $p = true$.

Dwuargumentowy operator $\vee$, zwany *alternatywą*, przyjmuje wartość *false*wtedy i tylko wtedy, gdy oba jego argumenty są fałszywe, a w pozostałych przypadkach przyjmuje wartość *true*. Zatem $p \vee q = false$ wtedy i tylko wtedy gdy $p = false$ i $q = false$, a gdy choć jeden z argumentów jest prawdziwy, wówczas ich alternatywa jest prawdziwa. Należy uważać, aby zwykłej alternatywy nie mylić z alternatywą wyłączającą, która wartość *true*przyjmuje tylko wtedy, gdy dokładnie jeden argument przyjmuje wartość *true*. Taka alternatywa nazywana jest *albo* i oznaczana symbolem xor. Trzeba uważać, bo w języku potocznym często te dwie alternatywy są mylone. Jeżeli ojciec mówi synowi ,,Pójdź do kina albo zaproś dziewczynę na kolację", a syn robi jedno i drugie, to nie do końca jest jasne, czy postąpił zgodnie z jego wolą. Czasami dla podkreślenia użycia zwykłej alternatywy *lub* używa się w języku potocznym osobnego omówienia przypadku, gdy oba argumenty są prawdziwe, na przykład mówi się: ,,Będę zadowolona, jeśli skosisz trawnik lub jeśli zrobisz obiad, albo i jedno i drugie". To ostatnie ,,jedno i drugie" z formalnego punktu widzenia nie jest konieczne: pierwsze ,,lub" zawiera w sobie informację, że spełnienie obu życzeń spowoduje stan zadowolenia u rozmówcy.

Koniunkcję oznaczaną przez $\wedge$ definiujemy jako operator przyjmujący wartość *true*wtedy i tylko wtedy gdy oba argumenty mają wartość *true*. Zatem $p \wedge q = true$ wtedy i tylko wtedy gdy $p = true$ i $q = true$. Możemy też wydefiniować operator $\wedge$ używając operatorów $\neg$ i $\vee$ w następ[ujący sposób: $p \wedge q$ wtedy i tylko wtedy gdy $\neg(\neg p \vee \neg q)$.

Największe kłopoty sprawia *implikacja*, której zrozumienie czasami budzi sprzeciw. Operator $p \to q$ przyjmuje wartość *false*wtedy i tylko wtedy gdy p jest zdaniem prawdziwym, a q fałszywym. We wszystkich pozostałych przypadkach implikacja jest prawdziwa. Znowu formalnie można wyrazić implikację za pomocą dwóch pierwszych operatorów:

$p \rightarrow q$ wtedy i tylko wtedy gdy $\neg p \vee q$. Implikację $p \rightarrow q$ czytamy ,,jeżeli p to q''. Zdanie p nazywamy wtedy poprzednikiem implikacji (lub przesłanką), a q — jej następnikiem (lub wnioskiem). Wątpliwości budzi sytuacja, gdy wartością zdania p jest *false*. Czy jesteśmy skłonni bez zastrzeżeń przyjąć, że zdanie *Jeżeli Biegun Północny znajduje się w Polsce, to naturalnym środowiskiem dla fok jest wnętrze krateru czynnego wulkanu*? Zgodnie z naszą definicją takie zdanie uznajemy za prawdziwe. Albo zdanie *Jeżeli Biegun Północny znajduje się w Polsce, to stolicą Polski jest Warszawa.* Niby wszystko jest w porządku: stolicą Polski jest faktycznie Warszawa, tylko czy ma to wynikać z tego, że w Polsce jest Biegun Północny?

Otóż nie ma to wynikać, choć oba przytoczone wyżej zdania uznajemy za prawdziwe. Przyjmijmy, że związek między poprzednikiem, a następnikiem implikacji jest czysto formalny i nie wyraża żadnej zależności ani bezpośredniego wpływu między przesłanką, a wnioskiem. Gwarantuje nam tylko tyle, że w przypadku prawdziwości przesłanki wniosek musi być też prawdziwy, natomiast gdy presłanka jest fałszywa, wówczas niczego o prawdziwości wniosku nie wiemy. Ograniczenie implikacji do takich, w których istnieje rzeczywisty związek między przeslanką, a wnioskiem, doprowadziło do powstania działu logiki zwanego logiką modalną, a prace w tej dziedzinie mają doniosłe znaczenie dla informatyki. Pozostawiamy ten temat jednak z boku naszych rozważań.

Kolejnym operatorem jest *równoważność*, którą możemy interpretować jako zwykłą równość wartości logicznych. Zdanie $p \leftrightarrow q$ jest prawdziwe wtedy i tylko wtedy gdy albo p i q są jednocześnie prawdziwe, albo są jednocześnie fałszywe. Zachodzi zatem $p \leftrightarrow q$ wtedy i tylko wtedy gdy $p \rightarrow q$ oraz $q \rightarrow p$.

1.5 Użyteczne tożsamości logiczne rachunku zdań

Jedną z najczęściej stosowanych tożsamości logicznych jest tożsamość

$$(p \rightarrow q) \Leftrightarrow (\neg q \rightarrow \neg p)$$

Jest ona podstawą dowodów nie wprost. Jeżeli naszym celem jest pokazanie, że z p wynika q, równie dobrze możemy pokazać, że z $\neg q$ wynika $\neg p$.

Często stosowanym sylogizmem logicznym jest wynikanie

$$(p \rightarrow q) \wedge (\neg q) \Rightarrow \neg p$$

Zauważmy, że wypowiadane przez nas zdania w mowie potocznej często poprzedzamy jak gdyby ukrytą klauzulą ich prawdziwości. Na przykład mówiąc ,,Jeśli zdasz ten egzamin to mi tu kaktus na ręce wyrośnie'' mamy na myśli po pierwsze, że to wypowiedziane zdanie jest zdaniem prawdziwym (p to ,,zdasz ten egzamin'', q, to ,,mi tu kaktus na ręce wyrośnie''), czyli prawdziwe jest zdanie $p \rightarrow q$. Po drugie ogłaszając jednocześnie, że w następniku implikacji jest oczywisty absurd ($\neg q$ jest przecież ewidentną prawdą) poparty precyzyjnym wskazaniem gdzie ten kaktus ma wyrosnąć, chcemy w ten zawoalowany nieco sposób wyrazić swoją opinię, że zdanie egzaminu jest poza zasięgiem naszego rozmówcy ($\neg p$).

Ta zgrabna figura retoryczna ma wiele wariantów. Jako q możemy podstawić rozmaite smakowite zdania typu

- ja jestem biskupem

- ja jestem chiński uczony
- niech mnie szlag trafi (dunder świśnie, etc.)

przy czym trzeci z przykładów niewątpliwie na miano zdania nie zasługuje, choć doskonale pełni tu swoją rolę.

Przyjrzyjmy się teraz podstawowym tożsamościom, które dotyczą zdań logicznych. Zauważymy przy okazji, jak duży związek mają one z tożsamościami znanymi z teorii zbiorów.

Zachodzą następujące równości:

$$(p \vee q) \vee r = p \vee (q \vee r) \quad \text{łączność alternatywy}$$

$$p \vee p = p \quad \text{idempotentność alternatywy}$$

$$p \wedge p = p \quad \text{idempotentność koniunkcji}$$

$$-(-p) = p \quad \text{prawo podwójnej negacji}$$

$$(p \wedge q) \wedge r = p \wedge (q \wedge r) \quad \text{łączność koniunkcji}$$

$$p \vee q = q \vee p \quad \text{przemienność alternatywy}$$

$$p \wedge q = q \wedge p \quad \text{przemienność koniunkcji}$$

$$(p \wedge q) \vee r = (p \vee r) \wedge (q \vee r) \quad \text{rozdzielność alternatywy względem koniunkcji}$$

$$(p \vee q) \wedge r = (p \wedge r) \vee (q \wedge r) \quad \text{rozdzielność koniunkcji względem alternatywy}$$

$$-(p \vee q) = (-p) \wedge (-q) \quad \text{prawo negacji alternatywy}$$

$$-(p \wedge q) = (-p) \vee (-q) \quad \text{prawo negacji koniunkcji}$$

Dwa ostatnie prawa nazywają się też *prawami de Morgana.*

$$p \vee \mathit{false} = p$$

$$p \vee \mathit{true} = \mathit{true}$$

$$p \wedge \mathit{false} = \mathit{false}$$

$$p \wedge \mathit{true} = p$$

Cztery ostatnie prawa nazywane są prawami identyczności.

Zauważmy, że wszystkie te prawa mają swoje odpowiedniki w teorii zbiorów. Suma teoriozbiorowa odpowiada alternatywie, iloczyn teoriozbiorowy — koniunkcji, dopełnienie — negacji, zbiór pusty zdaniu *false*, cała przestrzeń U — zdaniu *true*.

W istocie ten związek między światem logiki, a teorią zbiorów jest bardzo głęboki. Czasami, żeby wyrazić wszystko w języku teorii zbiorów, pewne własności, które wyrażają się za pomocą formuł logicznych, interpretuje się jako zbiory tych obiektów, które te własności spełniają. Na przykład zamiast formuły zdaniowej „n jest liczbą parzystą" rozważa się zbiór liczb parzystych.

Zauważmy też, że istnieje związek między zawieraniem się zbiorów w teorii zbiorów, a implikacją w logice. Spójrzmy się na formuły logiczne zawierające zmienną x intepretowaną w pewnej dziedzinie, jak na definicje pewnych podzbiorów rozważanej dziedziny. Zobaczymy wtedy, że ilekroć zachodzą implikacje między formułami, to zbiory, które te formuły opisują zawierają się jeden w drugim. Na przykład jeśli rozważymy formuły $\phi(x) = x$jest podzielne przez2 oraz $\varphi(x) = x$jest podzielne przez4, to pierwsza formuła definiuje nam zbiór liczb parzystych, zaś druga — zbiór liczb podzielnych przez 4. Rzecz jasna dla każdego x zachodzi $\varphi(x) \to \phi(x)$: jeżeli liczba jest podzielna przez 4, to jest parzysta. Jednocześnie $\{x \in I\!N : \varphi(x)\} \subseteq \{x \in I\!N : \phi(x)\}$. Faktycznie zbiór liczb podzielnych przez 4 zawiera się w zbiorze liczb parzystych. Jest to szczególny przypadek prawa

$$(\varphi(x) \Rightarrow \phi(x)) \Leftrightarrow (\{x \in X : \varphi(x)\} \subseteq \{x \in X : \phi(x)\})$$

Zawieranie się zbiorów tworzy pewną hierarchię. Formuły, które odpowiadają coraz to zawężającym się zbiorom tworzą łańcuch implikacji. Zauważmy, że jeśli mamy do czynienia implikacją zawsze prawdziwą, to formuła, która występuje w poprzedniku implikacji jest mocniejsza, niż ta, która występuje w jej następniku. Stwierdzenie, że x jest podzielne przez 4 jest mocniejsze, niż stwierdzenie, że x jest podzielne przez 2. Im zbiór definiowany przez formułę jest mniejszy, tym formuła jest mocniejsza. Najmocniejszym stwierdzeniem jest zatem formuła odpowiadająca zbiorowi pustemu, czyli *false*, a najsłabszym — formuła odpowiadająca całej przestrzeni, czyli *true*. Jeżeli potraktujemy logikę jako język umożliwiający definiowanie pojęć, to zdania zawsze prawdziwe nic nie mówią nam o świecie. Niosą zero informacji. Z punktu widzenia określenia interesującej nas własności nie przedstawiają żadnego znaczenia — ot takie masło maślane. Spełnione są wszędzie, więc nie wyróżniają żadnego zbioru.

Uogólnieniem praw rządzących zbiorami i zdaniami jest *algebra Boole'a* — struktura algebraiczna (czyli zbiór z określonymi działaniami spełniającymi zadane prawa) zdefiniowana na poziomie abstrakcyjnym, której modelami są w szczególności zarówno zdania logiczne, jak i podzbiory dowolnego zbioru.

1.6 Kwantyfikatory

Gdy chcemy wypowiadać pewne zdania o interesującej nas dziedzinie, często bardzo użyteczne są *kwantyfikatory*. Najbardziej klasyczne z nich, to *kwantyfikator ogólny „dla każdego”*, oznaczany symbolem $\forall$ oraz *kwantyfikator szczegółowy „istnieje”*, oznaczany symbolem $\exists$ (odwrócone A bierze się z angielskiego słowa All — wszystkie, a odwrócone E — z Exists — istnieje). Za ich pomocą możemy z formuły zdaniowej, której wartość zależy od interpretacji zmiennych utworzyć zdanie prawdziwe bądź fałszywe. Na przykład formuła zdaniowa $x^2 > 0$ interpretowana w zbiorze liczb rzeczywistych może stać się zdaniem prawdziwym lub fałszywym, w zależności od wartości zmiennej x. Predykaty $\forall x \in \mathrm{R} : x^2 > 0$ oraz $\exists x \in \mathrm{R} : x^2 > 0$ są już zdaniami: pierwsze z nich jest fałszywe, a drugie — prawdziwe.

Kwantyfikatorów poprzedzających formułę może być więcej. Każdy z nich musi się odnosić do innej zmiennej. Formuła zdaniowa $x > y$ ma jakby 2 stopnie swobody: występują w niej dwie zmienne wolne, czyli takie, które nie są związane żadnym kwantyfikatorem. Zgodnie z tym, co powiedzieliśmy poprzednio, może ona służyć do definiowania pewnego zbioru (w tym przypadku tych par (x, y), w których $x < y$). Jeżeli ją jednak poprzedzimy kwantyfikatorem na przykład $\exists x \in \mathrm{R}_{-}$, to wartość tego predykatu będzie zależała jedynie od wartościowania zmiennej y. Dla dodatnich wartości y będzie to zdanie prawdziwe, a dla nieujemnych — fałszywe. Dodanie dodatkowego kwantyfikatora wiążącego zmienną y usunie ostatni stopień swobody i uczyniz predykatu zdanie. Zauważmy, że kolejność kwantyfikowania może być istotna. Zdanie $\forall y \in \mathrm{R} : \exists x \in \mathrm{R}_- : x < y$ jest zdaniem prawdziwym, podczas gdy zdanie $\exists x \in \mathrm{R}_- : \forall y \in \mathrm{R} : x < y$ jest zdaniem fałszywym.

Kolejność podawania kwantyfikatorów nie wpływa na wartość zdania, gdy kwantyfikatory te są jednego typu i stoją obok siebie. Zachodzi zatem $\forall x \in X : \forall y \in Y : \varphi(x,y) \Leftrightarrow \forall y \in Y : \forall x \in X : \varphi(x,y)$ dla każdej formuły zdaniowej $\varphi(x,y)$. Analogiczna przemienność kwantyfikatorów dotyczy kwantyfikatora $\exists$.

Zauważmy, że w mowie potocznej często brak kwantyfikatora utożsamiany jest z domniemanym kwantyfikatorem ogólnym. Na przykład, gdy ktoś mówi ,,Ludzie są głupi", to tak naprawdę znaczy, że ,,*Wszyscy* ludzie są głupi" (albo przynajmniej wszyscy, o których się mówi), a nie że *istnieją* głupi ludzie.

Jeżeli kwantyfikator ogólny zawęża dziedzinę określoności zmiennej, która w nim występuje, to działa on na zasadzie implikacji. Załóżmy, że U jest przestrzenią określoności zmiennej x, zaś A jest pewnym podzbiore tej przestrzeni. Pisząc $\forall x \in A : \phi(x)$ czytamy: dla każdego x ze zbioru A zachodzi $\phi(x)$, a wartość tego zdania jest identyczna z wartością zdania $\forall x \in U : x \in A \to \phi(x)$. Z przyjęcia tej konwencji wynika pewien ważny fakt, który będzie miał zastosowanie w dowodach poprawności programów. Zdanie

$$\forall x \in \emptyset : \phi(x)$$

ma wartość prawda, niezależnie od postaci formuły $\phi(x)$. Rozwinięcie tego zdania do postaci implikacyjnej uświadomi nam, że poprzednik implikacji jest fałszywy.

Kwantyfikatory ogólny i szczegółowy są powiązane ze sobą następującymi zależnościami:

$$\forall x \in X : \phi(x) \Leftrightarrow \neg \exists x \in X : \neg phi(x)$$

oraz

$$\exists x \in X : \phi(x) \Leftrightarrow \neg \forall x \in X : \neg phi(x),$$

co w skrócie łatwo zapamiętać jako napisy $\forall = \neg\exists\neg$ oraz $\exists = \neg\forall\neg$.

Ważne są też prawa negacji kwantyfikatorów:

$$\neg \forall x \in X : \phi(x) \Leftrightarrow \exists x \in X : \neg \phi(x)$$

oraz

$$\neg \exists x \in X : \phi(x) \Leftrightarrow \forall x \in X : \neg \phi(x),$$

1.7 Dowody

Pojęcie dowodu jest w matematyce znane od dawna. Formalizacja tego pojęcia jest dziełem matematyków XX-wiecznych. Dowody starożytnych opierały się na zdrowym rozsądku i tak naprawdę były przeprowadzane w dziedzinie semantycznej. Do przeprowadzenia dowodu była konieczna znajomość dziedziny, której ten dowód dotyczył.

Współczesne spojrzenie na dowód matematyczny jest nieco inne. Zakładamy, że z góry mamy dany zestaw zdań prawdziwych opisujących interesujący nas świat, a zwanych *aksjomatami* oraz zestaw *reguł dowodzenia*, czyli reguł, które mówią, jak do zadanego zbioru zdań dołączyć kolejne. Aksjomaty łącznie ze wszystkimi zdaniami, które da się otrzymać przez stosowanie reguł wnioskowania tworzą zbiór zdań zwany *teorią*.

Zauważmy, że dowód jest pojęciem syntaktycznym: stosujemy operacje na napisach i formalnie przeprowadzenie dowodu oznacza doprowadzenie zbioru zdań udowodnionych do takiej postaci, żeby znalazło się w nim interesujące nas zdanie. W jakiej kolejności stosować reguły wnioskowania, aby dowód był możliwie krótki, na ogół nie wiadomo: często po latach odkrywa się znacznie prostsze (krótsze) dowody pewnych faktów.

Początkowy zestaw aksjomatów zazwyczaj opisuje nam dziedzinę, w której prowadzimy dowód. Powinien być on *niesprzeczny*, czyli taki, aby nie dało się z niego wywieść fałszu oraz *pełny*, czyli taki, żeby każde zdanie prawdziwe dawało się dowieść. Szczególnie to drugie żądanie jest trudne do spełnienia. Zazwyczaj sprowadza się do tego, żeby w skończony sposób przedstawić istotę badanego modelu tak, aby nic co nie wynika z przyjętych założeń nie umknęło naszej uwadze, czyli wszystko, co wynika z przyjętych założeń (aksjomatów) dawało się dowieść w ramach badanej teorii.

Euklides z przyjętych dwunastu aksjomatów wyprowadził wszystkie znane starożytnym twierdzenia geometrii, a metoda jego postępowania choć nie była dowodzeniem w ścisłym sensie, była na tyle blisko ścisłości, że luk w jego rozumowaniu nie zauważono.

Zmiana jednego z aksjomatów (słynnego 5. postulatu głoszącego, że przez punkt nie leżący na prostej można przeprowadzić dokładnie jedną prostą równoległą do niej) doprowadziła do powstania innych teorii geometrycznych: geometrii nieeuklidesowych Łobaczewskiego i Riemanna. Okazuje się, że dla potrzeb teorii względności geometria Riemanna jest znacznie lepszym modelem rzeczywistości, niż geometria euklidesowa.

Wielki matematyk niemiecki, Dawid Hilbert, na przełomie XIX i XX wieku opracował układ 10 aksjomatów, równoważny układowi Euklidesa oraz pokazał jego pełność. Jednocześnie wytrwale poszukiwał niesprzecznego i pełnego układu aksjomatów pozwalającego opisać arytmetykę liczb naturalnych, czyli takiego, że dawałoby się wywieść z niego dowolne zdanie prawdziwe o liczbach naturalnych. W 1930 roku świat obiegło sensacyjne odkrycie austriackiego matematyka Kurta Goedla, który pokazał, że takiego układu nie ma. Wynik ten uważany jest przez niektórych za największe twierdzenie w historii matematyki.

Dowód jest ciągiem kolejnych zdań już udowodnionych, który zaczyna się od pewnego zdania zwanego aksjomatem i przyjętego a priori jako prawdziwe. Kolejne zdania uzyskujemy przez wykorzystanie któregoś z aksjomatów lub którejś z reguł dowodzenia, w której w miejsce zmiennych zdaniowych wstawiono zdania już udowodnione.

1.8 Logika, a C

W języku C wyrażenia logiczne nie różnią się od innych wyrażeń (np. arytmetycznych). Przy wyliczaniu wartości wyrażenia bierze się pod uwagę tylko pierwszy bajt wyniku i sprawdza się, czy jego wartość wynosi zero. Jeli tak, to wartością wyrażenia jest *false*, ajeśli nie, to dla dowolnej niezerowej wartości wartością wyrażenia jest *true*.

Wyrażenia arytmetyczne można łączyć za pomocą następujących spójników: `==`, `!=`, `<`, `<=`, `>`, `>=` (odpowiednio: równość, różność, mniejszość, mniejszość lub równość, większość, większość lub równość) tworząc z nich wyrażenia logiczne. Natomiast spójniki logiczne, to:

- `!`, czyli negacja (np. `!(x>y)` oznacza nieprawda że `x` jest większe od `y`, co inaczej można równoważnie zapisać `x<=y`.

- `||`, czyli alternatywa (np. `(x>y)||(y>z)`.
- `&&`, czyli koniunkcja (np. `(x>y)&&(y>z)`.

Bardzo ważną cechą C jest leniwe wyliczanie warunków logicznych, które polega na ty, że w przypadku alternatywy wartości składników wylicza się od lewej do prawej i w pierwszym momencie, w którym uzyska się wartość *true*, przerywa się dalsze wyliczanie wychodząc z założenia, że wynik i tak będzie prawdą. Podobnie w przypadku koniunkcji przerywa się wyliczanie w przypadku pierwszego od lewej napotkania wartości *false* wśród czynników koniunkcji.

Paradoksalnie zmienia to zasadniczo prawa logiki. Okazuje się bowiem, że logika używana w językach programowania jest co najmniej trójwartościowa: poza wartościami *true* i *false* istnieje jeszcze możliwość nieokreślenia wartości w przypadku, gdyby wartość wyrażenia nie dała się policzyć.

Zilustrujmy to na przykładzie. Rozważmy dwie instrukcje warunkowe: `if ((x>0) &&(y/x>1))`$\{I\}$ oraz `if ((y/x>1) &&(x>0))` $\{I\}$, gdzie I jest pewną instrukcją. Pierwsza z tych konstrukcji jest bezpieczna i nigdy nie spowoduje przerwania programu: do sprawdzenia, a więc wyliczenia, warunku `x/y>1` dojdzie tylko gdy `x>0`. Gdyby `x` było mniejsze lub równe zero, wówczas wartością pierwszego z czynników koniunkcji byłoby *false* i nie doszłoby do wyliczenia niebezpiecznego dzielenia `y/x`. W drugim przypadku najpierw zostanie sprawdzony warunek `y/x>1`, co dla $x = 0$ spowoduje błąd dzielenia przez zero i przerwanie programu.

Zatem w C przy leniwym wyliczaniu wartości logicznych nie zachodzi prawo przemienności alternatywy ani koniunkcji.

W innych językach programowania jest różnie. Najczęściej programista może wybrać sobie opcję zgodnie z którą następuje wyliczanie leniwe bądź pełne wartości wyrażeń logicznych. Tak jest w szczególności w realizacjach Pascala firmy Borland.

1.9 Zadania

1. Czy jeśli w instrukcjach ilustrujących nieprzemienność koniunkcji w C zamienimy koniunkcję na implikację, to którakolwiek z nich będzie bezpieczna?
2. Zad 1,2,3,4,5/98,12,14/99,19/100,6,7,8,9/132, 1,2/796, 3,4,7,8/797, 13–17/789 z Matematyki dyskretnej Rossa, Wrighta.