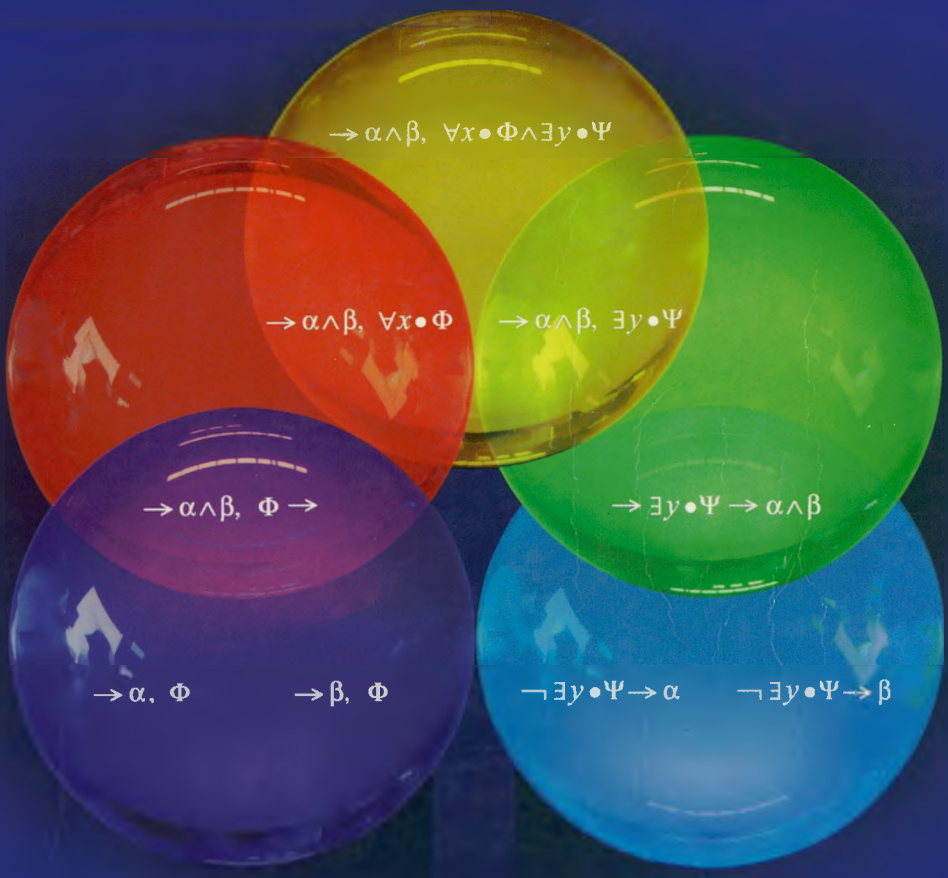
Zbigniew Huzar



Elementy logiki i teorii mnogości dla informatyków

Zbigniew Huzar

Elementy logiki i teorii mnogości dla informatyków

Oficyna Wydawnicza Politechniki Wrocławskiej
Wrocław 2007

Recenzenci
Marian Adamski
Leszek Pacholski
Wiesław Szwarc

Opracowanie redakcyjne i korekta
Alicja Kordas

Projekt okładki
Zofia i Dariusz Godlewscy

Niniejszy podręcznik jest poprawioną i uzupełnioną wersją, wydanej w 2002 roku, książki „Elementy logiki dla informatyków"

ISBN 978-83-7493-349-0

Oficyna Wydawnicza Politechniki Wrocławskiej
Wybrzeże Wyspiańskiego 27, 53-370 Wrocław
http://www.oficyna.pwr.wroc.pl
oficwyd@pwr.wroc.pl

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Zam. nr 710/2007

Spis treści

Przedmowa

Informatyka jest dyscypliną młodą, liczącą około pięćdziesiąt lat. Sam termin *informatyka* pojawił się w języku polskim na początku lat siedemdziesiątych, a termin *komputer* zadomowił się na dobre dopiero w końcu lat siedemdziesiątych ubiegłego wieku. Rozwój informatyki był i pozostaje stymulowany dwoma czynnikami. Pierwszym jest rozwój technologii, głównie elektronicznej. Dzięki postępowi w tej dziedzinie stało się możliwe technicznie zrealizowanie najpierw urządzeń liczących, których koncepcje były rozważane znacznie wcześniej, a później zbudowanie uniwersalnych urządzeń liczących – współczesnych komputerów. Drugim czynnikiem jest potencjalnie ogromne pole zastosowań informatyki. Oba te wzajemnie sprzężone czynniki doprowadziły do sytuacji, że komputer staje się powszechnym narzędziem pracy niemal w każdej dziedzinie.

Miarą tempa rozwoju obecnie dominującej technologii półprzewodnikowej jest fakt, że wydajność sprzętu komputerowego, mierzona częstotliwością zegara sterującego pracą komputera, i – podobnie – rozmiar pamięci operacyjnej komputera podwaja się co półtora roku. Przewiduje się, że taka tendencja może się utrzymać do lat 2015–2020. Wyznacznikiem rozpowszechnienia zastosowań informatyki jest obecnie nie tylko Internet – globalna sieć komputerowa, stanowiąca federację setek tysięcy sieci komputerowych – ale również rozpoczynający się proces integracji Internetu, telefonii komórkowej oraz telewizji cyfrowej.

Informatyka dostarcza specyficznych narzędzi i metod, które można wykorzystywać do rozwiązywania problemów w różnych dziedzinach. Do zrozumienia tej specyfiki i możliwości zastosowań informatyki potrzeba trwałych i niezawodnych podstaw. Tak jak w przypadku innych nauk ścisłych, podstawy informatyki są oparte na matematyce, a dokładniej na wybranych jej działach, odpowiednio przystosowanych do potrzeb informatyki. Podstawy informatyki są wprawdzie ciągle kształtowane, ale pewne ich elementy można obecnie uznać za ustabilizowane. W historii matematyki był okres na przełomie XIX i XX wieku, gdy uświadomiono sobie konieczność ustalenia podstaw matematyki, bez których nie byłby możliwy spójny rozwój różnych działów: algebry, analizy matematycznej, rachunku prawdopodobieństwa, topologii itp. Podstawy matematyki, które uformowały się w pierwszej połowie ubiegłego wieku, objęły dwa niezależne wcześniej działy – teorię mnogości i logikę. Podobnie jest z podstawami informatyki, za które również można uważać teorię mnogości i logikę, z położeniem

akcentu, silniejszego niż w podstawach matematyki, na logikę. Wynika to także z tego, że informatykę traktuje się niekiedy jako dyscyplinę wyrosłą z podstaw matematyki.

Szczególna rola matematyki w podstawach informatyki nie jest jednak uzasadniona wyłącznie względami historycznymi. Zasadniczy powód wynika z roli, jaką pełni informatyka w zastosowaniach praktycznych. Rozwiązywanie problemów, przed którymi staje informatyk, wymaga od niego – po pierwsze – zrozumienia danej dziedziny zastosowań i zrozumienia na czym dany problem polega, po wtóre – informatyk musi znać i rozumieć narzędzia i metody, którymi może dysponować, wreszcie – po trzecie – musi zaproponować jak, za pomocą posiadanych środków, dany problem rozwiązać. Opis problemu na tle specyficznej dziedziny jest początkowo wyrażany w języku naturalnym. Precyzyjny jego opis wymaga wyrażenia go w języku sformalizowanym, czyli języku o ściśle określonej składni i semantyce. Potrzeba taka wynika stąd, że przedstawione ostatecznie komputerowi do policzenia rozwiązanie problemu musi być wyrażone w języku programowania, czyli również pewnym sformalizowanym języku. Komputer, w odróżnieniu od człowieka, nie potrafi bowiem podjąć żadnych innych akcji niż te, które są wyrażone w pewnym języku sformalizowanym.

Konieczność formalizacji informatyki wynika więc z dwóch powodów. Po pierwsze – ze specyfiki komputera i tego, co potrafi, a to – co potrafi – można sprowadzić do umiejętności przetwarzania symboli. Po drugie – wynika z potrzeby zrozumienia abstrakcji, czyli procesu budowy formalnego opisu problemu na podstawie opisu wyrażonego w języku naturalnym. Formalny opis problemu jest pewnym modelem rzeczywistego problemu występującego w realnej dziedzinie. Model jest opisem uproszczonym, to znaczy koncentruje się tylko na wybranych aspektach rzeczywistego problemu. Na jakich aspektach i jak szczegółowo ma skupiać się model zależy oczywiście od celu jego budowy. Sposób budowy modelu, ocena zgodności modelu z opisywaną rzeczywistością, związek pomiędzy modelem opisu a modelem rozwiązania są typowymi zagadnieniami, wokół których wyrastają teorie i działy matematyki. Klasycznym przykładem są analiza matematyczna i teoria równań różniczkowych, które rozwinęły się na skutek zapotrzebowania dziewiętnastowiecznej techniki i fizyki.

Na początku XX wieku z formalizacją matematyki wiązano zbyt wielkie nadzieje. Program formalizacji matematyki, związany głównie z nazwiskiem Dawida Hilberta, załamał się w latach trzydziestych ubiegłego wieku, po odkryciach Kurta Gödla, który wskazał na swoistą ograniczoność metod formalnych. Gdyby się okazało, że program Hilberta jest realizowalny, wówczas można byłoby przypuszczać, że wszystko to, co potrafi człowiek, mógłby również zrealizować komputer. Tak jednak nie jest, dlatego intuicja i kreatywność są tymi właściwościami człowieka, które stanowią o jego przewadze nad komputerem. Oznacza to, że informatyk w swojej pracy powinien traktować metody formalne jako uzupełnienie i wsparcie własnej pomysłowości i twórczości.

W informatyce metody formalne stanowią fundament podstawowych pojęć, takich jak: pojęcie algorytmu, obliczalności czy złożoności obliczeniowej.

Logika dostarcza języka do przedstawiania i badania własności modeli informatycznych, w tym systemów komputerowych i języków programowania, a zwłaszcza środków do definiowania składni i semantyki języków programowania. W języku logiki można specyfikować wymagania stawiane projektowanym systemom oprogramowania. Język logiki może ponadto być bezpośrednio używany jako język programowania. Ogromną rolę odgrywa logika w zastosowaniach informatyki, na przykład w tworzeniu i funkcjonowaniu baz wiedzy i systemów ekspertowych.

Szczególną rolę odgrywa logika w procesie wytwarzania oprogramowania. Na gruncie logiki stało się możliwe sformułowanie pojęcia poprawności programów, a następnie opracowanie metod weryfikacji ich poprawności. Proces wytwarzania oprogramowania, jak na przykład w inżynierii oprogramowania, jest obecnie w coraz większym zakresie wspomagany przez komputer. Budowa narzędzi wspomagających ten proces opiera się na formalnych metodach, mających oparcie na gruncie logiki.

Niniejszy podręcznik pt. *Elementy logiki i teorii mnogości dla informatyków* jest poprawioną i rozszerzoną wersją wydania z 2002 roku: *Elementy logiki dla informatyków*, Oficyna Wydawnicza Politechniki Wrocławskiej. Tak jak wydanie poprzednie, obejmuje głównie logikę klasyczną wraz z krótkimi informacjami o logikach nieklasycznych.

W języku polskim jest wiele bardzo dobrych podręczników logiki, pisanych przede wszystkim dla matematyków – na przykład: [Adamowicz, Zbierski 1991], [Grzegorczyk 1975], [Hunter 1982], [Rasiowa 1998], [Słupecki, Hałkowska, Piróg-Rzepecka 1978], a w ostatnim okresie: skrypt [Tiuryn 2003] oraz [Guzicki, Zakrzewski 2005], [Kraszewski 2007], przy czym dwie ostatnie pozycje ograniczają się tylko do teorii mnogości. Warto wspomnieć o krótkiej i dawno wydanej książce o przeglądowym charakterze [Lyndon 1978]. Natomiast, oprócz nielicznych – na przykład: [Mostowski, Pawlak 1970], [Szałas 1992] – praktycznie nie ma takich podręczników dla informatyków. Zwraca uwagę fakt, że w ostatnim okresie powstają różne podręczniki logiki dla informatyków w języku angielskim – na przykład: [Fitting 1990], [Kelly 1997], [Nissanke 1999], [Socher-Ambrosius, Johann 1996], [Ben-Ari 2001]. Ostatnia pozycja – pod wieloma względami podobna do niniejszego podręcznika – ukazała się w 2005 roku w tłumaczeniu na język polski. Jedną z istotnych różnic między tymi dwiema kategoriami podręczników jest sposób przedstawiania systemów dowodzenia. Dla matematyków zwykle jako podstawowy wybiera się system dowodzenia Hilberta, który dobrze oddaje praktykę dowodową „klasycznego" matematyka, w informatyce natomiast większą rolę odgrywają systemy dowodowe, które – inaczej niż system Hilberta – pozwalają na automatyzowanie procesu dowodzenia. W niniejszym podręczniku omawia się za-

tem – jako podstawowe systemy dowodzenia – system sekwentów Gentzena i system oparty na regule rezolucji.

Pierwszy rozdział podręcznika jest ogólnym wprowadzeniem, wyjaśniającym czym jest logika. Pozostały materiał można podzielić na cztery części.

Pierwsza część, obejmująca rozdziały od 2. do 8., jest krótką prezentacją elementów teorii mnogości, algebr abstrakcyjnych i języków formalnych. W obecnym wydaniu książki wprowadzono dodatkowe informacje na temat liczb kardynalnych oraz nieco poszerzono rozdział dotyczący języków formalnych.

W drugiej części, obejmującej rozdziały od 9. do 12., omówiono rachunek zdań i kwantyfikatorów – ich składnię, semantykę oraz związane z nimi systemy dowodzenia oparte na sekwentach Gentzena i regule rezolucji.

W trzeciej części, obejmującej rozdziały 13. i 14. o charakterze informacyjnym, krótko omówiono inne systemy dowodzenia oraz dokonano przeglądu innych, nieklasycznych logik. W obecnym wydaniu zamieszczono opis systemu dowodzenia opartego na tablicach analitycznych, jako systemu alternatywnego w stosunku do systemu dowodzenia opartego na regule rezolucji.

Część czwarta – dołączona do tego wydania książki, obejmująca rozdziały 15. i 16. – przedstawia zastosowanie metod logiki do definiowania składni i semantyki języków programowania oraz klasyczną logikę programów Hoare'a, służącą dowodzeniu poprawności programów.

Prezentacja materiału jest sformalizowana tylko częściowo i odnosi się w zasadzie do definiowania pojęć oraz do sformułowania i udowodnienia wybranych twierdzeń. Zwrócono uwagę przede wszystkim na twierdzenia o poprawności i zupełności systemu dowodzenia opartego na rachunku sekwentów Gentzena, w przypadku pozostałych przedstawianych systemów dowodzenia ograniczono się tylko do sformułowania odpowiednich twierdzeń.

Do każdego rozdziału są dołączone ćwiczenia. Zebrane tu zadania pochodzą z różnych źródeł: część stanowi opracowania autora lub współpracowników, część jest zaczerpnięta z podręczników przedstawionych w spisie literatury: [Fitting 1990], [Gabbay 1998], [Gerstig 1993], [Kelly 1997], [Marek, Onyszkiewicz 1975], [Nissanke 1999], [Stanosz 2002], [Pacholski 2004], [Ławrow, Maksimowa 2004], [Guzicki, Zakrzewski 2005].

W celu czytelnego wyodrębnienia przykładów wprowadzono w tekście linie rozdzielające na początku i końcu odpowiednich akapitów. Zakończenia dowodów są zaznaczone czarnym kwadracikiem ■.

Podręcznik jest przeznaczony w zasadzie dla studentów pierwszego roku informatyki na studiach politechnicznych. Zakres materiału jest jednak szerszy i dlatego mogą skorzystać z niego, jako z lektury uzupełniającej, także studenci lat starszych.

Składam serdeczne podziękowania moim kolegom z Instytutu Informatyki Stosowanej Politechniki Wrocławskiej za pomoc podczas przygotowywania tego podręcznika. Szczególnie gorąco dziękuję profesorowi Iwanowi Tabakowowi oraz doktorowi Zdzisławowi Spławskiemu za uważną lekturę rękopisu, wskazanie usterek, cenne wskazówki oraz propozycje ulepszenia tekstu, a doktor Bogumile Hnatkowskiej – za wnikliwe uwagi dotyczące dwóch ostatnich rozdziałów.

Oddzielne podziękowanie kieruję do recenzentów – profesora Mariana Adamskiego, profesora Leszka Pacholskiego i doktora hab. Wiesława Szwarca, którzy – oprócz wskazania usterek – przedstawili sugestie i propozycje dotyczące sposobu prezentacji materiału.

Dziękuję też studentom Wydziału Informatyki i Zarządzania Politechniki Wrocławskiej, którzy wychwycili usterki edytorskie w poprzednim wydaniu książki.

Niezależnie od wszelkich uwag i sugestii, podręcznik, jak każdy obszerniejszy tekst, może zawierać przeoczenia bądź nieścisłości. Czytelników, którzy spostrzegą takie usterki, autor prosi o przekazanie odpowiedniej informacji pocztą elektroniczną na adres: zbigniew.huzar@pwr.wroc.pl

Wrocław, maj 2007 Zbigniew Huzar

1. Elementarne pojęcia logiczne

1.1. Czym jest logika?

Słowo *logika*[1] bywa używane przez filozofów, matematyków i w mowie potocznej w licznych znaczeniach i kontekstach. Długotrwała tradycja terminologiczna określa logikę jako analizę języka pod kątem jego wykorzystania do:

- definiowania,
- klasyfikowania,
- wnioskowania.

Celem takiej analizy jest podanie reguł posługiwania się językiem, aby był on skuteczny.

Logika pojmowana jako narzędzie poprawnego myślenia, czyli wnioskowania lub rozumowania, była już przedmiotem zainteresowania starożytnych[2] [Kotarbiński 1985], [Murawski 1995]. Drugie jej narodziny przypadają na wiek XIX. Traktowana jako pomocniczy dział matematyki, wyodrębniła się na początku XX wieku w samodzielną dyscyplinę matematyki. Obecnie zakres pojęcia logiki jest szeroki i obejmuje trzy odrębne dziedziny [Bocheński 1992]:

- logikę formalną,
- metodologię,
- filozofię logiki.

Przedmiotem zainteresowania *logiki formalnej* są *wypowiedzi* w danym języku, a dokładniej to, czy są one prawdziwe czy fałszywe. Daną wypowiedź można oceniać albo jako prawdziwą, albo jako fałszywą, gdyż żadna wypowiedź nie może być jednocześnie prawdziwa i fałszywa. *Prawda* i *fałsz*, jako własności wypowiedzi, są zatem podstawowymi pojęciami logiki.

Pojęcie prawdy, chociaż używane powszechnie, nie jest łatwe do określenia. Klasyczne rozumienie prawdy opiera się na związku pomiędzy wypowiedzią a rzeczywisto-

[1] Słowo *logika* pojawia się po raz pierwszy w tytule dzieła Demokryta (460–371 p.n.e.).

[2] Problematyka logiczna była rozważana przez Sokratesa (469–399 p.n.e.) i Platona (427–347 p.n.e.), ale za pierwszego twórcę systemu logiki uważa się Arystotelesa (384–322 p.n.e.).

ścią, do której dana wypowiedź się odnosi. Ten sens oddają słowa wypowiedziane blisko dwa tysiące lat temu przez Sekstusa Empiryka [Turski 1988]:

> *O każdym bowiem zdaniu rozstrzyga się, że jest prawdziwe albo że jest fałszywe ze względu na jego odniesienie do rzeczy, o której zostało orzeczone. Jeżeli bowiem okazuje się ono zgodne z rzeczą, o której zostało orzeczone, wydaje się prawdziwe, jeżeli niezgodne – fałszywe.*

Zadaniem logiki formalnej jest ustalanie prawdziwości wypowiedzi. Pierwszym zadaniem jest ustalanie prawdziwości wypowiedzi złożonych na podstawie prawdziwości wypowiedzi, które stanowią ich składowe. Szczególnym rodzajem są takie wypowiedzi złożone, które są zawsze prawdziwe, niezależnie od prawdziwości swoich wypowiedzi składowych. Wypowiedzi takie nazywa się prawami logicznymi. Głównym zadaniem logiki jest jednak wnioskowanie, czyli badanie tego, co na podstawie danego zestawu prawdziwych wypowiedzi – przesłanek – można sądzić o prawdziwości innych wypowiedzi. Chodzi tu o wnioskowanie niezawodne, to znaczy takie, które na podstawie prawdziwych przesłanek gwarantuje zawsze wyprowadzenie prawdziwych wniosków. Przedmiotem logiki są różnego rodzaju schematy niezawodnego wnioskowania, ich formułowanie, porządkowanie i uzasadnianie.

Metodologia zajmuje się stosowaniem logiki do różnych dziedzin [Bocheński 1992], [Wójcicki 1982]. W praktyce okazuje się, że te same prawa logiczne mogą być stosowane w różny sposób. Inną rzeczą są schematy wnioskowania, a inną przeprowadzanie wnioskowania na podstawie tych schematów. Znany na przykład podział wnioskowania na metody dedukcyjne i indukcyjne nie polega na użyciu różnych praw logiki, lecz na różnym użyciu tych samych praw. Celem metodologii – nie wnikając w szczegóły – są ogólne sposoby zdobywania i formułowania wiedzy prawdziwej albo przynajmniej dobrze uzasadnionej.

Filozofia logiki obejmuje analizę podstawowych pojęć logiki [Bocheński 1992]. Próbuje odpowiadać na przykład na pytania: *Co to jest prawda? Co to jest prawo logiczne? Skąd wiadomo, że jest ono prawdziwe?*

Książka obejmuje tylko *logikę formalną*, nazywaną inaczej *logiką matematyczną* lub *logiką symboliczną*. Logika formalna wprowadza język symbolicznego zapisu wypowiedzi i określa jak można takim symbolicznym zapisom przypisywać pewne znaczenie, czyli – w jaki sposób można określać ich *semantykę*? Zakres wypowiedzi, które można zapisywać w języku logiki formalnej, nie obejmuje oczywiście wszystkich wypowiedzi, które można sformułować w języku naturalnym. Język logiki formalnej jest natomiast całkowicie wystarczający do przedstawiania i analizy wypowiedzi dowolnych działów matematyki. Nic w tym dziwnego, gdyż narodziny współczesnej logiki wiążą się właśnie z potrzebą precyzyjnego sformułowania i analizy zagadnień z zakresu podstaw matematyki, które pojawiły się pod koniec XIX i na początku XX wieku. Dlatego logikę formalną określa się niekiedy jako *metamatematykę*, czyli jako naukę dostarczającą języka do opisu wszystkich pozostałych działów matematyki.

Celem logiki formalnej jest ujęcie procesu rozumowania, albo wnioskowania, w postaci *przekształcania napisów* reprezentujących wypowiedzi. Chodzi o to, aby na podstawie pewnych napisów, reprezentujących wypowiedzi uznane za prawdziwe, uzyskiwać prawdziwe wnioski – nowe napisy, reprezentujące nowe, prawdziwe wypowiedzi. Inaczej: chodzi o to, aby przekształcania napisów reprezentowały niezawodne schematy wnioskowania.

Przekształcanie napisów opiera logika formalna na *systemie dedukcyjnym*, czyli na ustalonym zbiorze reguł mechanicznego przekształcania tekstów. Pewne napisy przyjmuje się za poprawne z założenia. Traktuje się je jako *aksjomaty* systemu dedukcyjnego. Inne napisy przyjmuje się za poprawne tylko wtedy, gdy daje się je wyprowadzić z aksjomatów przez stosowanie ustalonych *reguł* systemu dedukcyjnego. Reguła jest mechanicznym sposobem przekształcania jednych napisów w inne napisy. Napisy wyprowadzone w wyniku stosowania przyjętych reguł powinny być poprawne nie tylko w sensie zgodności z przyjętymi regułami przekształcania, ale również powinny być poprawne w sensie semantycznym, to znaczy powinny być wypowiedziami prawdziwymi.

Logik formalnych jest wiele [Marciszewski 1987, 1988]. Różnią się one klasą obiektów, do których odnoszą się wypowiedzi, rodzajami wypowiedzi (np. wypowiedzi oznajmujące, przypuszczające, pytające, nakazujące) oraz stosowanymi systemami dedukcyjnymi – systemami wnioskowania. Szczególną rolę – zarówno ze względu na historię, a także zastosowania – pełni *logika klasyczna*. Logika klasyczna jest jądrem wszystkich innych logik formalnych, w tym również różnych specjalistycznych logik stosowanych w informatyce.

1.2. Język logiki formalnej

Jak wspomniano, przedmiotem logiki formalnej są *wypowiedzi* w danym języku, a dokładniej to: czy są prawdziwe czy fałszywe.

Nie wszystkie wypowiedzi mogą być jednak oceniane jako prawdziwe albo fałszywe. Nie sposób tak ocenić wypowiedzi rozkazującej czy pytającej, można tak oceniać co najwyżej wypowiedzi oznajmujące, ale nawet co do nich mogą powstawać wątpliwości. Na przykład, czy wypowiedź:

W 2100 roku bardzo popularną formą wypoczynku będą wakacje na Marsie.

jest prawdziwa czy fałszywa? Trudno to osądzić, przynajmniej w obecnej dobie. Z powodu braku wiedzy historycznej nie można natomiast stwierdzić, czy prawdziwa jest wypowiedź:

Król Bolesław Chrobry urodził się w poniedziałek.

Wypowiedzi, którym można przypisać prawdziwość albo fałszywość, będą nazywane *zdaniami*. Zdania mogą być proste, na przykład:

Książka leży na stole.
W programie koncertu jest symfonia Mahlera.

Warto zwrócić uwagę, że tego rodzaju zdania są formułowane w pewnym kontekście sytuacyjnym i tylko w tym kontekście można rozstrzygać, czy są prawdziwe czy fałszywe. W języku naturalnym spotyka się też wypowiedzi, których prawdziwość, nawet po ustaleniu kontekstu, może być trudna do określenia. Rozpatrzmy zdania:

On jest dosyć wysokim mężczyzną.
Samochód jechał dosyć wolno.

Powodem trudności w pierwszym zdaniu jest rozumienie zwrotu *dosyć wysoki*. Czy jest dosyć wysoki mężczyzna, który ma 180 cm wzrostu, czy dopiero taki, który ma 185 cm? Podobnie w drugim zdaniu problem stwarza rozumienie zwrotu *jechać dosyć wolno*.

Ze zdań prostych można budować zdania złożone, na przykład:

Pójdę do kina ***lub*** *pójdę do teatru.*
Jeżeli *wykonawcą koncertu będą filharmonicy berlińscy,* ***to*** *zwalą się tłumy.*
W 1939 roku Hitler napadł na Czechosłowację ***i*** *– w roku następnym – na Polskę.*

Zdania złożone powstają przez połączenie zdań prostych za pomocą spójników logicznych. Spójnikami logicznymi (zdaniowymi) są na przykład słowa i zwroty: *nie*, *lub*, *i* (*oraz*), *jeżeli ..., to ..., ... wtedy i tylko wtedy, gdy ...* .

W języku naturalnym zwroty te mają ustalone znaczenie. Poniżej przedstawia się prostą formalizację uściślającą ich znaczenie. Formalizacja spójników logicznym polega na:

- nadaniu im pewnej symbolicznej notacji,
- przypisaniu im znaczenia w terminach tablic prawdziwościowych.

Zdania będą oznaczane symbolami p, q, r, ... Spójniki logiczne będą oznaczane następująco:

- Spójnik *nie* – nazywany *negacją* – jest oznaczany symbolem $\neg$. Negację zdania p zapisujemy: $\neg p$.
- Spójnik *i* (*oraz*) – nazywany *koniunkcją* – jest oznaczany symbolem $\wedge$. Koniunkcję zdań p, q zapisujemy: $p \wedge q$.
- Spójnik *lub* – nazywany *dysjunkcją* lub *alternatywą* – jest oznaczany symbolem $\vee$. Dysjunkcję (alternatywę) zdań p, q zapisujemy: $p \vee q$.
- Spójnik *jeżeli ..., to ...* – nazywany *implikacją* – jest oznaczany symbolem $\Rightarrow$. Implikację zdań p, q zapisujemy: $p \Rightarrow q$.

  Implikację $p \Rightarrow q$ można także czytać w inny sposób, na przykład:

  p jest warunkiem dostatecznym do tego, że q.
  q pod warunkiem, że p.
  q wtedy, gdy p.
  q jest warunkiem koniecznym do tego, że p.

- Spójnik *wtedy i tylko wtedy, gdy* – nazywany *równoważnością* – jest oznaczany symbolem $\Leftrightarrow$. Równoważność zdań p, q zapisujemy: $p \Leftrightarrow q$.

  Równoważność $p \Leftrightarrow q$ można także czytać w inny sposób, na przykład:

  p jest warunkiem koniecznym i wystarczającym do tego, że q.
  p dokładnie wtedy, gdy p.
  p jest równoważne p.

Uwaga

Czasem negacją, koniunkcją itd. nazywa się zdania złożone, które powstają z innych zdań przez łączenie spójnikami negacji, koniunkcji itd.

Zapisując zdania złożone w postaci symbolicznej, będziemy używać nawiasów, grupując w odpowiedni sposób zdania składowe. Nawiasy będą opuszczane, gdy przyjmie się następującą kolejność stosowania (wiązania) spójników (od najsilniejszego do najsłabszego):

$$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow.$$

Zamiast na przykład

$$((\neg p) \wedge q) \vee (r \wedge s)) \Rightarrow t$$

można pisać

$$\neg p \wedge q \vee r \wedge s \Rightarrow t$$

Zakłada się ponadto, że występujące obok siebie spójniki $\wedge$, $\vee$ łączą w lewo, a występujące obok siebie spójniki $\Rightarrow$, $\Leftrightarrow$ łączą w prawo. Na przykład:

$p \wedge q \wedge r$ znaczy $(p \wedge q) \wedge r$,
$p \Rightarrow q \Rightarrow r$ znaczy $p \Rightarrow (q \Rightarrow r)$.

Prawdziwość zdania złożonego zależy tylko od prawdziwości jego zdań składowych i od tego, jakim spójnikiem są one połączone. Taką własność nazywa się *ekstensjonalnością*.

Tablica 1.1

p	$\neg p$
P	F
F	P

p	q	$p \wedge q$
F	F	F
P	F	F
F	P	F
P	P	P

p	q	$p \vee q$
F	F	F
P	F	P
F	P	P
P	P	P

p	q	$p \Rightarrow q$
F	F	P
P	F	F
F	P	P
P	P	P

p	q	$p \Leftrightarrow q$
F	F	P
P	F	F
F	P	F
P	P	P

Rolę spójników logicznych daje się prosto wyrazić za pomocą tablic prawdziwościowych – tablica 1.1. Tablica prawdziwościowa jest tabelarycznym zestawieniem wszystkich wartościowań zdań składowych oraz odpowiadających im wartościowa-

niom zdania złożonego połączonego danym spójnikiem logicznym. W celu zmniejszenia rozmiarów tablicy zamiast *prawda* lub *fałsz* pisze się symbole **P** oraz **F**. Tablica uściśla znaczenie, które się wiąże ze spójnikami logicznymi w języku naturalnym.

Własność ekstensjonalności, czyli abstrahowanie od wewnętrznych treści zdań składowych przy ocenie prawdziwości zdań złożonych, może powodować kolizję z potocznym rozumieniem prawdziwości zdań. Typowym przykładem są zdania połączone spójnikiem implikacji. Zdanie

Jeżeli księżyc ma kształt sześcianu, to dzisiaj mamy dzień rektorski.

uznalibyśmy za bezsensowne. Formalnie jest to zdanie poprawnie zbudowane, a ponadto jest to zdanie prawdziwe. Chociaż zdanie *dzisiaj mamy dzień rektorski* nie musi być zdaniem prawdziwym, ale fałszywość zdania *księżyc ma kształt sześcianu* pociąga prawdziwość całej wypowiedzi. Pojęcie sensowności, do którego często się odwołujemy w języku naturalnym, nie ma bezpośredniego odpowiednika w języku logiki klasycznej. Wynika to z tego, że język logiki klasycznej jest znacznie uboższy od języka naturalnego – jest tylko pewnym jego przybliżeniem.

Zdania są wypowiedziami, którym – w danym kontekście wypowiedzi – jednoznacznie przypisuje się prawdziwość lub fałsz. Znane są też inne rodzaje wypowiedzi, którym prawdziwość lub fałsz można przypisać dopiero po dodatkowych uściśleniach dotyczących elementów wypowiedzi. Na przykład o prawdziwości zdania

Mężczyzna jest wyższy od kobiety.

można jednoznacznie się wypowiedzieć dopiero wtedy, gdy wiadomo, o którego mężczyznę i o którą kobietę chodzi. Mężczyzna i kobieta stanowią tu argumenty wypowiedzi. Wskazując na konkretnego mężczyznę i na konkretną kobietę, można stwierdzać o prawdziwości lub fałszu tego zdania.

Zdania tego rodzaju nazywa się *funkcjami zdaniowymi* albo *formami zdaniowymi*. Można je traktować jako pewien sposób wyrażania *własności* elementów pewnego zbioru. Zdania takie będą zapisywane $P(a)$, gdzie a jest argumentem wypowiedzi.

Funkcji zdaniowych często się używa w powiązaniu z charakterystycznymi zwrotami, na przykład:

Możliwe, że zachodzi $P(a)$.
Dla każdego elementu a ze zbioru A zachodzi $P(a)$.

Pierwszy ze zwrotów to rodzaj zwrotu *modalnego*. Taki zwrot występuje na przykład w zdaniach:

Możliwe, że Piotr wypożyczył już potrzebną mu książkę.
Możliwe, że prezes spóźni się na spotkanie.
Możliwe, że w 2100 roku bardzo popularną formą wypoczynku będą wakacje na Marsie.

Drugi ze zwrotów to rodzaj zwrotu *kwantyfikacyjnego*. Przykłady wypowiedzi z tym zwrotem:

Każdy student otrzymuje indeks.
Każdy dorosły ponosi pełną odpowiedzialność za swoje czyny.

Dalej zajmiemy się przede wszystkim zwrotami kwantyfikacyjnymi. Będą rozważane tylko dwa zwroty:

Dla każdego elementu a zachodzi P(a).
Istnieje element a, dla którego zachodzi P(a).

Pierwszy zwrot jest nazywany *kwantyfikacją ogólną* albo *uniwersalną*, a drugi – *kwantyfikacją szczegółową* albo *egzystencjalną*.

Istnieją jeszcze inne rodzaje zwrotów kwantyfikacyjnych, które nie będą rozważane, na przykład:

Dla większości elementów a ze zbioru A zachodzi P(a).
Dla nieskończenie wielu elementów a ze zbioru A zachodzi P(a).

Jeżeli symbolem $P(a)$ oznaczyć funkcję zdaniową, której dla ustalonego elementu a ze zbioru A można w jednoznaczny sposób przyporządkować prawdę albo fałsz, to wypowiedź z kwantyfikatorem ogólnym dla $P(a)$ zapisuje się symbolicznie w postaci

$$\forall a \in A \bullet P(a),$$

a wypowiedź z kwantyfikatorem szczegółowym w postaci

$$\exists a \in A \bullet P(a).$$

Wypowiedzi z kwantyfikatorami można czytać w różny sposób.

Zapis z kwantyfikatorem ogólnym $\forall a \in A \bullet P(a)$ można czytać:

Dla każdego $a \in A$ [zachodzi] $P(a)$.
Dla dowolnego $a \in A$ [zachodzi] $P(a)$.
Dla wszystkich $a \in A$ [zachodzi] $P(a)$.
Wszystkie $a \in A$ mają własność $P(a)$.
$P(a)$ dla wszystkich $a \in A$.

Zapis z kwantyfikatorem szczegółowym $\exists a \in A \bullet P(a)$ można czytać:

Dla pewnego $a \in A$ [zachodzi] $P(a)$.
Dla jakiegoś $a \in A$ [zachodzi] $P(a)$.
Jakieś $a \in A$ ma własność $P(a)$.
$P(a)$ dla pewnego $a \in A$.

Zapis [zachodzi] oznacza tu, że słowo 'zachodzi' występuje opcjonalnie – można je czytać albo pomijać.

Uwaga

Oprócz wprowadzonej, używa się również innych notacji na wypowiedzi z kwantyfikatorami, na przykład:

$\forall a \in A.P(a)$ $\forall a \in A{:}P(a)$ $(\forall a \in A)P(a)$ $\bigwedge_{a \in A} P(a)$

$\exists a \in A.P(a)$ $\exists a \in A{:}P(a)$ $(\exists a \in A)P(a)$ $\bigvee_{a \in A} P(a)$

1.3. Wnioskowanie

Logika formalna zajmuje się schematami wnioskowania, które pozwalają na to, aby na podstawie prawdziwości jednych wypowiedzi – *przesłanek* – wnioskować o prawdziwości innych wypowiedzi – *wniosków*. Historycznie najstarsze schematy wnioskowania, nazywane *sylogizmami*, pochodzą od Arystotelesa.

Przykładem wnioskowania opartego na jednym z sylogizmów jest następujące rozumowanie:

Wszyscy bogowie greccy są zazdrośni.
Zeus jest greckim bogiem.
Zatem: *Zeus jest zazdrosny.*

Dwa pierwsze zdania są tu przesłankami, a ostatnie – wnioskiem (konkluzją).

Ogólnie *schemat wnioskowania* można przedstawić w postaci „ułamka", w którego „liczniku" będą zapisywane przesłanki, a w „mianowniku" będą zapisywane wnioski. Rozpatrzmy kilka schematów wnioskowań, które można odnieść do wielu codziennych sytuacji.

Znanym schematem jest *modus ponendo ponens*, mający postać

$$\frac{p \Rightarrow q \qquad p}{q}$$

gdzie: p oraz q oznaczają dowolne wypowiedzi.

Na podstawie takiego schematu wnioskuje się na przykład:

***Jeżeli** dzisiaj jest niedziela, **to** jutro jest poniedziałek*
Dzisiaj jest niedziela.
Jutro jest poniedziałek.

Schemat ten jest niezawodny, co oznacza, że jeżeli przesłanki są prawdziwe, to także prawdziwy jest wyprowadzony na ich podstawie wniosek.

Inny przykład również niezawodnego schematu wnioskowania to *modus ponendo tollens*

$$\frac{\textit{Albo } p, \textit{ albo } q \qquad p}{\neg q}$$

Zwrot ***albo*** ..., ***albo*** ... nie był wcześniej omówiony. Zgodnie z potocznym rozumieniem zdanie złożone postaci ***albo*** *p*, ***albo*** *q* jest prawdziwe tylko wtedy, gdy jest prawdziwe dokładnie jedno ze zdań składowych *p*, *q*.

Przykład wnioskowania:

Albo *pójdę do kina*, ***albo*** *pójdę do teatru.*
Pójdę do kina.

Nie *pójdę do teatru.*

Jeszcze inny przykład niezawodnego schematu wnioskowania to *modus tollendo ponens*:

$$\frac{p \vee q \qquad \neg p}{q}$$

Według tego schematu wnioskuje się w przykładzie:

Pójdę do kina ***lub*** *pójdę do teatru.*
Nie *pójdę do kina.*

Pójdę do teatru.

Często korzysta się ze schematów wnioskowania nazywanych *sylogizmem warunkowym*. Przykładem jest schemat:

$$\frac{p \Rightarrow q \qquad q \Rightarrow r}{p \Rightarrow r}$$

Schematy wnioskowania, którymi zajmuje się logika formalna, są w pewien sposób ograniczone. Nie biorą pod uwagę treści, lecz tylko prawdziwość zdań, dlatego mechaniczne stosowanie przedstawionych schematów wnioskowania, jeżeli się nie wnika w treść zdań, może prowadzić do absurdalnych wniosków. Jako przykład posłuży następujące rozumowanie: Niech będą dane dwie wypowiedzi:

Jeżeli *Cezar pozostanie w domu,* ***to*** *Cezar nie zostanie zabity przez spiskowców.*

oraz

Jeżeli *Cezar nie zostanie zabity przez spiskowców,* ***to*** *Cezar wygłosi przemówienie w Senacie.*

Wprowadźmy oznaczenia. Niech:

p oznacza: *Cezar pozostanie w domu.*
q oznacza: *Cezar nie zostanie zabity przez spiskowców.*
r oznacza: *Cezar wygłosi przemówienie w Senacie.*

Z zastosowaniem tych oznaczeń wnioskowanie oparte na schemacie sylogizmu warunkowego przebiega następująco: Na podstawie przesłanek $p \Rightarrow q$ oraz $q \Rightarrow r$ otrzymuje się wniosek $p \Rightarrow r$, czyli:

***Jeżeli** Cezar pozostanie w domu, **to** Cezar wygłosi przemówienie w Senacie.*

Wniosek ten jest całkowicie sprzeczny ze zdrowym rozsądkiem. Wynika to z tego, że w zastosowanym schemacie wnioskowania uwzględnia się tylko prawdziwość przesłanek, a nie uwzględnia się treściowego powiązania przesłanek i konkluzji: pozostawanie w domu w pewnym okresie wyklucza przebywanie w Senacie w tym samym okresie i, tym samym, wygłoszenie tam przemówienia.

To, że zdanie jest wynikiem zastosowania pewnego schematu wnioskowania do innych zdań-przesłanek nazywa się *konsekwencją dowodową* albo *konsekwencją składniową*. W rozpatrywanym przykładzie wyraża się to zapisem

$$\{p \Rightarrow q, q \Rightarrow r\} \vdash p \Rightarrow r$$

Ogólnie, jeżeli $\{p_1, ..., p_n\}$ jest pewnym zbiorem zdań-przesłanek, a q jest zdaniem, które wyprowadzono z tego zbioru na podstawie jedno- lub wielokrotnego stosowania pewnych schematów wnioskowania, to zapisuje się to w postaci

$$\{p_1, ..., p_n\} \vdash q$$

Symbol $\vdash$ nazywa się symbolem *konsekwencji składniowej*. Powyższy zapis czyta się: q jest konsekwencją składniową zbioru zdań $\{p_1, ..., p_n\}$.

Rozpatrzmy teraz rozumowanie, które nie opiera się na przedstawionych schematach wnioskowania. Niech dany będzie przykład:

***Jeżeli** znany pianista da recital, **to** przyjdą tłumy, **gdy** ceny biletów nie będą zbyt wygórowane.*
***Jeżeli** znany pianista da recital, **to** ceny biletów nie będą zbyt wygórowane.*
Zatem: ***Jeżeli** znany pianista da recital, **to** przyjdą tłumy.*

W pierwszym z powyższych zdań występuje zwrot ***gdy***. Zgodnie z potocznym rozumieniem, zdanie złożone postaci *p* ***gdy*** *q* jest równoważne zdaniu ***jeżeli*** *q*, ***to*** *p*.

Czy jeżeli przesłanki w przykładzie – dwa pierwsze zadania – są prawdziwe, to czy prawdziwa jest również konkluzja – trzecie zdanie? Wprowadźmy oznaczenia. Niech:

p oznacza: *Znany pianista da recital.*
q oznacza: *Przyjdą tłumy.*
r oznacza: *Ceny biletów będą zbyt wygórowane.*

Po zastosowaniu tych oznaczeń nasze wnioskowanie ma postać:

$p \Rightarrow (\neg r \Rightarrow q)$
$p \Rightarrow \neg r$
zatem: $p \Rightarrow q$

Można przytoczyć dwa sposoby uzasadnienia poprawności wnioskowania. Pierwszy sposób można zilustrować tablicą prawdziwościową (tablica 1.2). Podobnie jak w poprzedniej tablicy, zamiast *prawda* i *fałsz* pisze się **P** i **F**.

Tablica 1.2

	p	q	r	$\neg r$	$\neg r \Rightarrow q$	$p \Rightarrow \neg r$	$p \Rightarrow (\neg r \Rightarrow q))$	$p \Rightarrow q$
1	F	F	F	P	F	P	P	P
2	F	F	P	F	P	P	P	P
3	F	P	F	P	P	P	P	P
4	F	P	P	F	P	P	P	P
5	P	F	F	P	F	P	F	F
6	P	F	P	F	P	F	P	F
7	P	P	F	P	P	P	P	P
8	P	P	P	F	P	F	P	P

Sposób uzasadniania jest tu następujący: wnioskowanie ma być niezawodne, to znaczy konkluzja ma być prawdziwa zawsze wtedy, gdy prawdziwe są przesłanki. Wystarczy rozpatrzyć wszystkie wartościowania zdań prostych p, q, r, przy których prawdziwe są zdania złożone stanowiące przesłanki, i sprawdzić, czy przy tych wartościowaniach prawdziwe jest również zdanie stanowiące konkluzję. Przypadki takich wartościowań reprezentują wiersze 1, 2, 3, 4 i 7 w tablicy 1.2. Analiza tych przypadków potwierdza poprawność wyprowadzonego wniosku.

Drugi sposób opiera się na następującym rozumowaniu nie wprost: jeżeli założyć, że nasz wniosek jest poprawny, to czy jest możliwe, aby jednocześnie były prawdziwe przesłanki i negacja konkluzji? Inaczej – czy zdanie

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

może być prawdziwe dla dowolnych wartościowań zdań prostych p, q, r? Okazuje się, co pokazuje tablica 1.3, że przy wszystkich wartościowaniach zdanie to jest fałszywe. Nie może być tak, że jednocześnie są prawdziwe przesłanki i negacja konkluzji. Nie jest zatem możliwe, aby zdania stanowiące przesłanki mogły być niezgodne ze zdaniem stanowiącym konkluzję.

Oba sposoby nie polegały na tekstowym przekształcaniu przesłanek, ale opierały się na analizie znaczenia zdań, dokładniej – na analizie ich prawdziwości. Oba sposoby potwierdziły, że konkluzja jest *konsekwencją logiczną*, albo *konsekwencją semantyczną*, zbioru przesłanek. Fakt ten, w odniesieniu do przykładu, zapisuje się w postaci

$$\{p \Rightarrow (\neg r \Rightarrow q), p \Rightarrow \neg r\} \models p \Rightarrow q$$

Ogólnie, jeżeli $\{p_1, ..., p_n\}$ jest pewnym zbiorem zdań-przesłanek, a q jest jego logiczną konsekwencją, zapisuje się to w postaci

$$\{p_1, ..., p_n\} \models q$$

Symbol $\models$ nazywa się symbolem *konsekwencji semantycznej*. Zapis ten czyta się: q jest konsekwencją semantyczną zbioru zdań $\{p_1, ..., p_n\}$.

Tablica 1.3

	p	q	r	$\neg r$	$\neg r \Rightarrow q$	$p \Rightarrow (\neg r \Rightarrow q)$	$p \Rightarrow \neg r$	$p \Rightarrow q$	$\neg(p \Rightarrow q)$	$(p \Rightarrow (\neg r \Rightarrow q)) \wedge (p \Rightarrow \neg r) \wedge \neg(p \Rightarrow q)$
1	F	F	F	P	F	P	P	P	F	F
2	F	F	P	F	P	P	P	P	F	F
3	F	P	F	P	P	P	P	P	F	F
4	F	P	P	F	P	P	P	P	F	F
5	P	F	F	P	F	F	P	F	P	F
6	P	F	P	F	P	P	F	F	P	F
7	P	P	F	P	P	P	P	P	F	F
8	P	P	P	F	P	P	F	P	F	F

Mając pojęcia konsekwencji składniowej i konsekwencji semantycznej, można sprecyzować niezawodność schematów wnioskowania. Schemat wnioskowania jest niezawodny (albo poprawny), gdy dla dowolnego zbioru zdań $\{p_1, ..., p_n\}$ oraz zdania q, jeżeli

$$\{p_1, ..., p_n\} \vdash q$$

to

$$\{p_1, ..., p_n\} \models q$$

Inaczej: schemat wnioskowania jest niezawodny, gdy dla dowolnego zbioru przesłanek konsekwencja składniowa pociąga konsekwencję semantyczną.

Schemat wnioskowania, który nie jest niezawodny, jest praktycznie bezużyteczny. Wszystkie przedstawiane wcześniej schematy są schematami niezawodnymi (poprawnymi), zawodnym natomiast schematem wnioskowania jest na przykład schemat postaci

$$\frac{p \Rightarrow q}{q}$$

Aby to stwierdzić, wystarczy rozpatrzyć wartościowanie, w którym obie wypowiedzi p oraz q są fałszywe. Przy takim wartościowaniu przesłanka schematu $p \Rightarrow q$ jest prawdziwa, ale wniosek q jest fałszywy.

1.4. Indukcja matematyczna

Zasada indukcji matematycznej jest jednym z bardziej użytecznych schematów wnioskowania. Bezpośrednio odnosi się ona do badania własności wyrażanych w terminach liczb naturalnych, czyli do własności postaci $P(n)$, gdzie $n \in Nat$. Zasada indukcji opiera się na prostej obserwacji, że cały zbiór liczb naturalnych można uporządkować, zaczynając od 0, a następnie można przechodzić do kolejnych liczb przez dodawanie 1. Z obserwacji tej wynika, że udowodnienie, iż pewna własność $P(n)$ zachodzi dla każdej liczby naturalnej n wymaga pokazania, że zachodzi ona dla $n = 0$, oraz że zachodzi $P(n + 1)$, gdy zachodzi $P(n)$. Czy zachodzi $P(0)$ wymaga zatem bezpośredniego zbadania, natomiast $P(1)$ zachodzi, ponieważ zachodzi $P(0)$, podobnie $P(2)$ zachodzi, ponieważ zachodzi $P(1)$ itd.

Twierdzenie 1.1 (*Zasada indukcji matematycznej*)

Niech $P(n)$ będzie pewną własnością, która odnosi się do liczby naturalnej n. Aby pokazać, że własność $P(n)$ zachodzi dla każdej liczby naturalnej $n \in Nat$, wystarczy pokazać, że:

krok początkowy: P zachodzi dla $n = 0$, czyli zachodzi $P(0)$,
krok indukcyjny: jeżeli zachodzi $P(n)$, to również zachodzi $P(n + 1)$.

Dowodzenie zgodnie z zasadą indukcji składa się z dwóch kroków. Krok początkowy wymaga zbadania zachodzenia własności P dla $n = 0$. Drugi krok wymaga udowodnienia implikacji: jeżeli $P(n)$, to $P(n + 1)$. Założenie $P(n)$ w tej implikacji nazywa się *hipotezą indukcyjną*.

Przykład 1.1

Niech $P(n)$ oznacza własność, że $2^n > n^2$. Własność ta nie zachodzi dla wszystkich liczb naturalnych, ale zachodzi dla $n \geq 5$.
Łatwo sprawdzić, że zachodzi $P(5)$, gdyż $2^5 > 5^2$, ale nie zachodzi $P(4)$, gdyż nieprawda, że $2^4 > 4^2$, i podobnie nie zachodzi $P(0)$, $P(1)$, $P(2)$, $P(3)$.
Jeżeli zachodzi $P(n)$ oraz $n \geq 5$, to zachodzi również $P(n + 1)$.
Istotnie, jeżeli $2^n > n^2$, to – że $2^{n+1} > (n+1)^2$ – wynika z następującego wnioskowania:

$$
\begin{aligned}
2^{n+1} &= 2 \times 2^n \\
&> 2n^2 && \text{– na mocy hipotezy indukcyjnej, że } 2^n > n^2 \\
&= n^2 + n^2 \\
&\geq n^2 + 5n && \text{– na mocy założenia, że } n \geq 5 \\
&= n^2 + 2n + 3n \\
&> n^2 + 2n + 1 && \text{– własność trywialna: } 3n > 1 \text{ dla } n \geq 1 \\
&= (n + 1)^2
\end{aligned}
$$

Czasem własności, o których się sądzi, że można je łatwo udowodnić metodą indukcji, mogą się okazać niemożliwe do bezpośredniego wykorzystania zasady indukcji. Ilustruje to przykład.

Przykład 1.2

Należy pokazać, że suma n początkowych liczb nieparzystych jest kwadratem pewnej liczby naturalnej, to znaczy że dla każdej liczby naturalnej n istnieje taka liczba naturalna k, że

$$\sum_{i=0}^{n-1}(2i+1)=k^2$$

Dla $n = 0$ własność $P(0)$ zachodzi trywialnie dla $k = 1$. Załóżmy teraz, że istnieje $k>1$ takie, że zachodzi powyższy wzór, wówczas

$$\sum_{i=0}^{n}(2i+1)=\sum_{i=0}^{n-1}(2i+1)+(2n+1)=k^2+2n+1$$

Niestety, nie ma gwarancji, że wyrażenie $k^2 + 2n + 1$ jest kwadratem pewnej liczby naturalnej i tym samym nie można dowodu kontynuować. Może to sugerować, że indukcja jest zbyt słabym schematem dla dowodzenia tego typu własności. Tak jednak nie jest. Należy zauważyć, że gdyby w rozważanym wyrażeniu przyjąć, że $k = n$, wówczas zachodziłaby równość

$$k^2 + 2n + 1 = n^2 + 2n + 1 = (n + 1)^2$$

Obserwacja ta sugeruje rozważenie mocniejszej własności, mianowicie

$$\sum_{i=0}^{n-1}(2i+1)=n^2$$

Jej udowodnienie metodą indukcji jest już proste i pozostawia się je Czytelnikowi.

Udowodnienie pewnych własności wymaga silniejszej formy indukcji. Mianowicie, w indukcyjnym kroku, aby udowodnić $P(n + 1)$ wymaga się założenia, że nie tylko zachodzi $P(n)$, ale również, że zachodzą $P(1)$, ..., $P(n - 1)$.

Twierdzenie 1.2 (*Zasada indukcji matematycznej – silna forma*)

Niech $P(n)$ będzie pewną własnością, która odnosi się do liczby naturalnej n. Aby pokazać, że własność $P(n)$ zachodzi dla każdej liczby naturalnej $n \in Nat$, wystarczy pokazać, że:

krok początkowy: P zachodzi dla $n = 0$, czyli zachodzi $P(0)$, oraz
krok indukcyjny: zachodzi $P(n + 1)$, gdy zachodzi $P(k)$ dla każdego $k = 0, ..., n$.

Przykład 1.3

Należy pokazać, że każda liczba naturalna $n \geq 2$ jest iloczynem liczb pierwszych. Własność ta zachodzi oczywiście dla $n = 2$. Dalej załóżmy, że własność ta zachodzi dla 2, 3, ..., n. Na tej podstawie należy pokazać, że zachodzi ona również dla $n + 1$. Jeżeli $n + 1$ jest liczbą pierwszą, to własność jest oczywiście prawdziwa. W przeciwnym razie, gdy $n + 1$ nie jest liczbą pierwszą, oznacza to, że $n + 1$ może być wyrażone jako iloczyn km dwóch liczb, gdzie $2 \leq k \leq n$ oraz $2 \leq m \leq n$. Na mocy hipotezy indukcyjnej liczby k oraz m są iloczynami liczb pierwszych, zatem $n + 1$ wyraża się również jako iloczyn liczb pierwszych.

W przykładzie nie korzysta się bezpośrednio z hipotezy indukcyjnej dla n, ale dla pewnych liczb k, m mniejszych od $n + 1$. Ogólnie może zachodzić potrzeba wykorzystania wielu takich liczb.

1.5. Logika w informatyce

Logika klasyczna znajduje w informatyce szerokie zastosowanie. W pierwszej kolejności dostarcza ona języka do przedstawiania i badania własności modeli informatycznych, w tym systemów komputerowych i języków programowania. Szczególną rolę odegrała logika w formowaniu pojęcia algorytmu i obliczalności [Arbib 1968], [Davis, Hersh 1994], [Mostowski, Pawlak 1970], [Penrose 1996].

Oprócz logiki klasycznej są wykorzystywane logiki specjalne, przeznaczone na przykład do specyfikacji oprogramowania, a także jako języki programowania.

Ważną rolę w informatyce odgrywają różnego rodzaju logiki nieklasyczne, między innymi w systemach eksperckich (doradczych). Zadaniem takich systemów jest wspomaganie człowieka podczas podejmowania decyzji, na przykład postawienie przez lekarza diagnozy o stanie zdrowia pacjenta na podstawie wyników badań. Proces podejmowania decyzji opiera się w takich przypadkach na informacji niepewnej lub niepełnej, a wnioski z przeprowadzonego wnioskowania nie muszą być niezawodne.

Oto wybrane działy informatyki, w których logika znajduje bezpośrednie zastosowanie:

- Specyfikacja i weryfikacja programów – np. [Bicaregui, Fitzgerald, Lindsay 1994], [Dembiński, Małuszyński 1981], [Shepard 1995].

  Formuły logiczne służą do wyrażenia tego, co program ma obliczać, czyli do wyrażania specyfikacji programu. Stwierdzenie czy dany program oblicza to, co powinien, czyli czy spełnia zadaną specyfikację, polega na odpowiednim manipulowaniu na tekście formuł specyfikacji i na tekście programu. Inaczej: stwierdzanie poprawności programu względem danej specyfikacji polega na przeprowadzeniu dowodu w odpowiedniej logice programów.

- Przetwarzanie rozproszone, współbieżność i sterowanie, systemy komunikacji w czasie rzeczywistym – np. [Apt, Olderog 1991], [Huzar 1989].

  Odpowiednie formy logiki zostały opracowane w celu wyrażania i wnioskowania o zjawiskach, które występują w przestrzeni i trwają w czasie. Są one wykorzystywane na przykład do wnioskowania o współpracy pomiędzy mobilnymi równoległymi procesami.

- Zarządzanie bazami wiedzy – np. [Bolc, Borodziewicz, Wójcik 1991], [Tyugu 1989].

  Zadaniem odpowiednich logik jest umożliwienie udzielenia odpowiedzi na pytania kierowane do bazy wiedzy. Udzielanie odpowiedzi na pytania sprowadza się do zbadania, czy jest ono konsekwencją semantyczną nagromadzonej wiedzy.

- Projektowanie układów logicznych – np. [Harrison 1973].

  Projektowanie układów elektronicznych komputerów, na przykład układów scalonych, spowodowało powstanie specjalistycznych logik, między innymi logik wielowartościowych i progowych.

- Systemy ekspertowe, planowanie i sztuczna inteligencja – np. [Bolc, Borodziewicz, Wójcik 1991], [Banerji 1990], [Bubnicki 1990], [Huzar, Kurzyński, Sas 1994].

  Dział ten wykształcił nową grupę logik, których istotą jest prowadzenie wnioskowania w warunkach informacji niepełnej lub niepewnej.

- Przetwarzanie języka naturalnego (lingwistyka informatyczna) – np. [Carnap 1990], [Marciszewski 1987].

  W celu automatycznej analizy tekstu, czy też automatycznego przekładu z jednego języka na inny, powstały różne logiki służące przede wszystkim do wyrażania znaczenia tekstu.

- Programowanie logiczne – np. [Kowalski 1989], [Wójcik 1991].

  Język logiki może być traktowany bezpośrednio jako język programowania. To, co w podejściu klasycznym jest logiczną specyfikacją programu – przy zachowaniu pewnych ograniczeń – może być interpretowane jako wykonywalny program.

Ćwiczenia

1. Która z wypowiedzi jest zdaniem, a która funkcją zdaniową:
   a) *Księżyc jest zrobiony z żółtego sera.*
   b) *On faktycznie jest wysokim mężczyzną.*
   c) *Słońce krąży dookoła Ziemi.*
   d) *W ciągu wieków składniki te uformowały rafy.*
   e) *Niech żyje przyjaźń między narodami!*

f) *Dwa jest liczbą parzystą.*
g) *Która drużyna zdobędzie mistrzostwo kraju w piłce nożnej?*
h) *Oczekuje się, że w przyszłym roku obroty na giełdzie znacznie wzrosną.*
i) $x^2 - 4 = 0$.
j) *Długi honorowe należy spłacać w ciągu 24 godzin.*
k) *Mężczyzna jest wyższy od kobiety.*
l) *Należy raczej zapobiegać niż leczyć.*
m) *Kalifornię co roku nawiedza trzęsienie ziemi o sile 7 stopni w skali Richtera.*
n) *Król Jagiełło był raczej wysokim mężczyzną.*

2. Jaką wartość logiczną mają zdania:
   a) *8 jest liczbą nieparzystą lub 6 jest liczbą parzystą.*
   b) *8 jest liczbą nieparzystą oraz 6 jest liczbą parzystą.*
   c) *Jeżeli 8 jest liczbą nieparzystą, to 6 jest liczbą parzystą.*
   d) *Jeżeli 8 jest liczbą nieparzystą oraz 6 jest liczbą parzystą, to 6 jest większe od 8.*

3. Które ze zdań jest negacją danego zdania:
   a) *Wynikiem obliczeń jest albo 2, albo 3.*
      (i) *Wynikiem nie jest ani 2, ani 3.*
      (ii) *Wynikiem nie jest 2 lub nie jest 3.*
      (iii) *Wynikiem nie jest 2 i nie jest 3.*
   b) *Ogórek jest zieloną rośliną nasienną.*
      (i) *Ogórek nie jest zielony, ale jest rośliną nasienną.*
      (ii) *Ogórek nie jest zielony lub nie jest rośliną nasienną.*
      (iii) *Ogórek nie jest zielony i nie jest rośliną nasienną.*

4. Wskaż poprzednik i następnik implikacji w zdaniach:
   a) *Pomyślny wzrost roślin jest uwarunkowany prawidłowym nawadnianiem.*
   b) *W przypadku modyfikacji programu pojawią się w nim błędy.*
   c) *Błędy w programie pojawią się tylko w przypadku jego modyfikacji.*
   d) *Oszczędność energii jest związana z dobrą izolacją ścian i szczelnością okien.*

5. W podanych zdaniach złożonych rozpoznaj zdania proste i łączące je spójniki:
   a) *Edmund Hillary i Tenzing Norgay są pierwszymi zdobywcami Mont Everestu.*
   b) *Indochiny leżą w strefie tropikalnej i mają gorące lata, ale zimy w części północnej są chłodne.*
   c) *Niezależnie od tego, jak wysoko skaczesz, księżyca nie osiągniesz, chyba że polecisz tam rakietą.*

6. Przedstaw tablice prawdziwościowe wyrażające znaczenie występujących w języku polskim, następujących zwrotów:
   a) *co najwyżej jedno z dwojga,*
   b) *dokładnie jedno z dwojga,*

c) *o ile ...,*
d) *ani ..., ani,*
e) *pod warunkiem, że ...,*
f) *chyba że,*
g) *choćby nawet,*
h) *zawsze wtedy, gdy.*

7. Za pomocą dowolnych dwóch spośród spójników: negacji, koniunkcji, alternatywy i implikacji, zdefiniuj znaczenie wyrażeń podanych w zadaniu 6.

8. Rozpatrz następujące wnioskowanie oparte na sylogizmie warunkowym:

Jeżeli dzisiaj jest wtorek, to jutro jest środa.
Jeżeli dzisiaj jest środa, to jutro jest czwartek.
Zatem: *Jeżeli dzisiaj jest wtorek, to jutro jest czwartek.*

Wyjaśnić przyczyny paradoksalnego wniosku.

9. Oto fragment raportu policji sporządzonego przez młodego aspiranta:

Świadek nie był zastraszony lub też, jeśli Henry popełnił samobójstwo, to testament odnaleziono. Jeśli świadek był zastraszony, to Henry nie popełnił samobójstwa. Jeśli testament odnaleziono, to Henry popełnił samobójstwo. Jeśli Henry nie popełnił samobójstwa, to testament odnaleziono.

Co komendant policji może wywnioskować z powyższego raportu (poza oczywistym faktem, że należy zwolnić aspiranta)? Odpowiedz na pytania:

Czy świadek był zastraszony?
Czy Henry popełnił samobójstwo?
Czy testament odnaleziono?

10. Pośród członków pewnego klubu lingwistycznego każdy uczy się francuskiego, niemieckiego lub hiszpańskiego. Wiadomo, że 20 uczy się francuskiego, 12 francuskiego i hiszpańskiego, 16 niemieckiego, 16 hiszpańskiego, 4 francuskiego i niemieckiego, 7 niemieckiego i hiszpańskiego, 3 wszystkich trzech języków. Ilu członków liczy klub? Ilu z nich uczy się dokładnie dwóch języków?

11. Oto przykłady wnioskowań przez indukcję:

a) Pokażę, że wszystkie liczby naturalne są parzyste. Oczywiście 0 jest liczbą parzystą. Niech n będzie dowolną liczbą naturalną i załóżmy, że dla wszystkich $k < n$, k jest parzyste. Niech n_1 i n_2 będzie dowolnym rozbiciem liczby n na sumę liczb mniejszych (tzn. $n = n_1 + n_2$). Ponieważ n_1 oraz n_2 są mniejsze od n, zatem n_1 i n_2 są parzyste, a więc n jest parzyste jako suma dwóch liczb parzystych.

b) Pokażę, że wszystkie dodatnie liczby naturalne są nieparzyste. Oczywiście 1 jest liczbą nieparzystą. Niech n będzie dowolną liczbą naturalną i załóżmy, że dla wszystkich $k < n$, k jest nieparzyste. Niech 1, n_1 *i* n_2 będzie dowolnym rozbiciem liczby n na sumę trzech liczb mniejszych (tzn. $n = n_1 + n_2 + 1$). Ponieważ

n_1 oraz n_2 są mniejsze od n, zatem n_1 i n_2 są nieparzyste, a więc n jest parzyste jako suma dwóch liczb nieparzystych i liczby 1.

c) Pokażę, że wszystkie proste na płaszczyźnie są równoległe. Rozważmy jednoelementowy zbiór prostych na płaszczyźnie. Oczywiście wszystkie proste należące do tego zbioru są do siebie równoległe. Załóżmy, że w każdym n-elementowym zbiorze prostych wszystkie proste są do siebie równoległe. Rozważmy teraz $(n + 1)$-elementowy zbiór prostych. Ustalmy w nim jedną prostą p. Na mocy założenia indukcyjnego wszystkie pozostałe n prostych są do siebie równoległe. Ustalmy teraz inną prostą q. Na mocy założenia indukcyjnego wszystkie pozostałe n prostych są również do siebie równoległe. Ponieważ relacja równoległości prostych jest przechodnia, wszystkie $n + 1$ proste są równoległe. Na mocy zasady indukcji matematycznej każdy zbiór prostych na płaszczyźnie zawiera wyłącznie proste równoległe. Dotyczy to zbioru wszystkich prostych na płaszczyźnie.

Które z tych rozumowań jest poprawne? Wskaż błędy popełnione w błędnym rozumowaniu.

12. Rozważyć uogólnienie problemu przedstawionego w przykładzie 1. Dla jakich liczb naturalnych n, k zachodzi nierówność: $2^n > n^k$?

13. O własności $P(n)$ wiadomo, że jest prawdziwe $P(1)$, a ponadto dla każdej liczby naturalnej n zachodzi implikacja $P(n) \Rightarrow P(n + 10)$. Czy wynika stąd, że prawdziwe są implikacje:
    a) $P(32) \Rightarrow P(62)$,
    b) $P(33) \Rightarrow P(61)$,
    c) $P(34) \Rightarrow P(63)$,
    d) $P(31) \Rightarrow P(64)$.

14. O własności $P(n)$ wiadomo, że jest prawdziwe $P(1)$, natomiast $P(100)$ jest fałszywe, a ponadto dla każdej liczby naturalnej n zachodzi implikacja $P(n) \Rightarrow P(n + 2)$. Czy wynika stąd, że:
    a) $P(101)$ jest prawdziwe,
    b) $P(300)$ jest prawdziwe,
    c) $P(50)$ jest fałszywe,
    d) $P(200)$ jest fałszywe.

15. Przeanalizuj prawdziwość zdania: *To, co mówię w tej chwili, jest kłamstwem.*

16. Oto rozmowa czterech krasnoludków A, B, C oraz D, z których każdy zawsze mówi prawdę albo zawsze kłamie:
    A mówi do B: *Jesteś kłamcą.*
    C mówi do A: *Ty sam jesteś kłamcą.*
    D mówi do C: *Oni obaj są kłamcami. I ty także jesteś kłamcą.*

    Który z nich mówi prawdę?

2. Elementarne pojęcia mnogościowe

2.1. Zbiór i element zbioru

Podstawą wszelkiej komunikacji pomiędzy ludźmi, a także pomiędzy ludźmi i komputerami, jest język, który używa wspólnie ustalonych symboli i jednoznacznie skojarzonych z nimi pojęć. Symbole służą do reprezentacji pojęć. Mogą one mieć różną postać – mogą to być dźwięki, obrazy, znaki graficzne. Każdy symbol powinien reprezentować pojęcie, które jest jednakowo rozumiane przez obiekty (podmioty) uczestniczące w komunikacji. Wprowadzenie języka wymaga zdefiniowania odpowiedniego zestawu symboli oraz zdefiniowania przypisywanego im znaczenia. Z wprowadzaniem nowego języka wiąże się pewien problem: definicje elementów języka wymagają opisu, wyrażanego w pewnym innym języku. Oznacza to, że przed wprowadzeniem pewnego języka należy dysponować innym językiem służącym do opisu nowego języka. W celu odróżnienia tych języków, język definiowany nazywa się *językiem przedmiotowym*, krótko językiem, a język służący do opisu języka przedmiotowego nazywa się *metajęzykiem*.

Uwaga

Pojęcie metajęzyka funkcjonuje w wielu sytuacjach. Na przykład podczas nauki języka obcego, załóżmy angielskiego, język ten opisujemy i wyjaśniamy za pomocą języka polskiego. Każdy metajęzyk ma swój metajęzyk – można pisać po niemiecku o kimś, kto pisze po polsku o angielskim. Istnieje niekończąca się hierarchia metajęzyków.

Elementy języka formalnego (symbolicznego) zawierają pojęcia odnoszące się do dwóch obszarów – obszaru teorii mnogości i logiki. Elementy tego języka wyjaśnia się w języku naturalnym. Język naturalny odgrywa tu rolę metajęzyka. Z języka naturalnego wykorzystuje się oczywiście tylko te pojęcia, co do których nie ma wątpliwości interpretacyjnych. Sytuacja jest podobna do tej, z którą spotyka się, studiując na przykład encyklopedię. Encyklopedia służy do wyjaśniania pewnych pojęć-haseł i czyni to za pomocą innych pojęć-haseł, o których się zakłada, że powinny być powszechnie znane i jednoznacznie rozumiane. Czasem wprawdzie jest tak, że w wyjaśnianiu pewnych haseł występują inne hasła, ale w odniesieniu do encyklopedii jako całości przyjmuje się, że istnieje pewien zestaw pojęć pierwotnych, których encyklo-

pedia używa, ale ich nie wyjaśnia i które się uważa za powszechnie zrozumiałe. Postępowanie takie nie jest całkowicie ścisłe i czasem może być źródłem niejednoznaczności lub nawet sprzeczności, ale praktycznie – w większości przypadków – pozwala na wprowadzanie i wyjaśnianie potrzebnych pojęć.

Obszary teorii mnogości i logiki silnie się przenikają. Formułując pojęcia należące do obszaru teorii mnogości, korzysta się z pojęć logicznych, ale też i odwrotnie – formułowanie własności logicznych wymaga odwołania się do pojęć mnogościowych. Można się zatem spotkać z dwoma podejściami do opisu logiki klasycznej. Pierwsze podejście polega na przyjęciu pewnych elementów teorii mnogości i wyprowadzaniu na ich podstawie pojęć logicznych. Drugie podejście, odwrotnie, polega na przyjęciu podstawowych pojęć logicznych i na wyprowadzaniu na ich podstawie pojęć mnogościowych. W książce przyjęto pierwsze podejście – przed pełnym opisem pojęć logicznych wyjaśnia się elementarne pojęcia z zakresu teorii mnogości.

Do podstawowych pojęć mnogościowych zalicza się:

- pojęcie *zbioru*,
- pojęcie *elementu* zbioru,
- pojęcie *należenia* bądź *nienależenia* elementu do zbioru.

Pojęcie zbioru – intuicyjnie zrozumiałe – okazało się bardzo trudne do precyzyjnego zdefiniowania. Poprzestaniemy tu na intuicyjnym albo naiwnym rozumieniu zbioru, tak jak czynił to w XIX wieku Cantor[3] – twórca teorii mnogości – który zbiór określał jako

> *ujęcie w całość określonych, dobrze wyróżnionych obiektów, zwanych elementami zbioru.*

W określeniu tym nie wskazuje się, czym mogą być elementy zbioru. Podane określenie zbioru nie jest precyzyjne, gdyż przy próbie odpowiedzi na pewne pytania mogą się pojawić sprzeczności. Próby uściślenia pojęcia zbioru prowadziły do powstania sformalizowanej teorii mnogości.

Należy podkreślić, że w podanym określeniu kładzie się akcent na rozróżnialność bytów stanowiących elementy zbioru. Nie określa się natomiast jak osiągać tę rozróżnialność, czy na przykład przez jednoznaczną identyfikację bytów, czy przez określenie unikalnych ich własności. Oznacza to jednak, że mając dwa byty, potrafi się stwierdzić, czy są one identyczne czy różne.

Jeżeli symbolem A oznacza się pewien zbiór oraz symbolem a oznacza się pewien element, to zapis $a \in A$ czyta się: *a jest elementem zbioru A*, natomiast zapis $a \notin A$ czyta się: *a nie jest elementem zbioru A*.

Jeżeli $a \in A$ oraz $b \in A$, to zapisuje się to skrótowo: $a, b \in A$.

[3] Georg Cantor (1845–1918).

Na ogół zbiory będziemy oznaczać napisami zaczynającymi się wielkimi literami lub pojedynczymi literami greckimi, a elementy zbiorów odpowiednimi małymi literami, z ewentualnymi indeksami.

Pewne zbiory przyjmuje się jako znane. Będą to zbiory: liczb naturalnych *Nat*, liczb całkowitych *Całkowite*, liczb wymiernych – *Wymierne*, liczb rzeczywistych – *Rzeczywiste*.

Szczególnym zbiorem jest *zbiór pusty* – zbiór, który nie ma żadnego elementu. Będzie on oznaczany symbolem $\emptyset$.

W celu wyeliminowania pewnej klasy paradoksów (zob. dalej – paradoks Russella), które mogą powstać podczas definiowania zbiorów, zakłada się, że żaden zbiór nie może być swoim elementem, to znaczy dla dowolnego zbioru A zachodzi: $A \notin A$.

2.2. Definiowanie zbiorów

Zbiory można definiować w różny sposób. Przedstawia się trzy sposoby definiowania zbiorów:

- enumeracyjny,
- rekursywny,
- ekstensjonalny.

Najprostszym sposobem definiowania zbioru jest jawne wskazanie wszystkich jego elementów. Sposób ten nazywa się *enumeracją* lub *wyliczeniem* elementów zbioru. Schemat takiej definicji ma postać

$$A =_{\text{def}} \{a_1, a_2, ..., a_n\}$$

Zapis ten czytamy: *A jest nazwą zbioru, którego elementami są* a_1, a_2, ..., a_n. Symbol $=_{\text{def}}$ czytamy: *równy z definicji*.

Przedstawiony wyżej schemat definicji zbioru zawiera dwa elementy:

- wprowadza symbol A jako *nazwę* zbioru,
- określa *znaczenie* (inaczej *interpretację*), które przypisujemy temu symbolowi; jest nim zestaw elementów a_1, a_2, ..., a_n, które należą do zbioru A. Elementy te, oddzielone przecinkami, tworzą skończony ciąg. Występujące tu trzy kropki są tylko zaznaczeniem, że liczba tych elementów może być dowolna, ale skończona. Definicja konkretnego zbioru musi oczywiście wymienić jawnie wszystkie jego elementy.

W związku z rozróżnieniem pojęcia symbolu oraz pojęcia znaczenia symbolu należy zwrócić uwagę na nazwę zbioru. Możliwe są dwa spojrzenia na nazwę.

W pierwszym spojrzeniu nazwę traktuje się tylko jako symbol – nazwa nie wyraża żadnego znaczenia, jest tylko znakiem lub ciągiem znaków z ustalonego repertuaru

znaków. Taką rolę ma symbol A występujący po lewej stronie w podanej poprzednio definicji.

W drugim spojrzeniu nazwę traktuje się jako zbiór, którego elementy są wymienione w nawiasach. Aby na przykład odpowiedzieć na pytanie czy $a \in A$, należy widzieć A jako zestaw konkretnych elementów.

Często dalej używanym zbiorem będzie zbiór wartości logicznych

$$Logiczne =_{def} \{prawda, fałsz\}$$

Rozpatrzmy dalsze przykłady enumeracyjnej definicji zbiorów:

$Kreski =_{def} \{|, -\}$
$Strzałki =_{def} \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$
$DniTygodnia =_{def} \{poniedziałek, wtorek, środa, czwartek, piątek, sobota, niedziela\}$
$LiteryMałe =_{def} \{a, b, ..., z\}$
$LiteryDuże =_{def} (A, B, ..., Z\}$

Elementami pierwszego i drugiego zbioru są symbole graficzne, elementami pozostałych zbiorów są litery lub napisy. W definicjach dwóch ostatnich zbiorów występują takie same kropki, ale z uwagi na kontekst, w którym występują, potrafimy nadać im odpowiednie różne znaczenia.

Bezpośrednio z definicji zbiorów wynika, że na przykład:

$| \in Kreski$
$\rightarrow, \uparrow \in Strzałki$

Używane pojęcie zbioru nie narzuca ograniczeń na to, czym mogą być jego elementy. W szczególności elementami zbioru mogą być inne zbiory. Rozpatrzmy przykłady zbiorów:

$\{a\}$
$\{\{a\}\}$
$\{\{a, b\}, \{a\}\}$
$\{\{\{a\}\}, \{a\}, a\}$

Są to zbiory *anonimowe*, to znaczy niemające nazw. Pierwszy zbiór składa się tylko z jednego elementu a. Drugi zbiór składa się również z jednego elementu, ale elementem tym jest zbiór jednoelementowy $\{a\}$. Trzeci zbiór ma dwa elementy, którymi są zbiory $\{a, b\}$ oraz $\{a\}$. Ostatni zbiór ma trzy elementy, z których każdy ma różną strukturę – pierwszy jest zbiorem postaci $\{\{a\}\}$, drugi jest zbiorem postaci $\{a\}$, a trzeci jest pojedynczym elementem a.

Enumeracyjne definiowanie zbioru nie jest możliwe, gdy zbiór zawiera nieskończenie wiele elementów. W tym przypadku można stosować podejście *rekursywne*. Rekursywna definicja zbioru składa się z dwóch części:

- *części bazowej*, w której jawnie wskazuje się na pewne obiekty jako elementy definiowanego zbioru,
- *części rekursywnej*, w której wskazuje się na nowe obiekty jako elementy definiowanego zbioru, przez odpowiednie odwołanie się do tych obiektów, o których już wiadomo, że należą do definiowanego zbioru (inaczej: część rekursywna określa jak za pomocą obiektów, o których wcześniej wiemy, że są elementami zbioru, można wyrazić inne obiekty, które są również elementami definiowanego zbioru).

Szczególnie ważnym i potrzebnym zbiorem nieskończonym jest zbiór liczb naturalnych *Nat*. Można zdefiniować go rekursywnie następująco:

- $0 \in Nat$
- jeżeli $n \in Nat$, to $n + 1 \in Nat$.

Elementy zbioru w części rekursywnej definicji są wyrażane przez napisy n oraz $n + 1$. Napisy te reprezentują pewne liczby, przy czym to, jakie są to liczby, zależy od tego, jaką liczbę przypisze się symbolowi n. Stosując część rekursywną po raz pierwszy, symbolowi n przypisuje się 0 i na tej podstawie wnioskuje się, że 1 jest również elementem zbioru *Nat*. Stosując część rekursywną po raz drugi, symbolowi n przypisze się liczbę 1 itd.

W podanej definicji zakłada się, że wiadomo jest, czym jest liczba i co oznacza dodanie jedynki do liczby. Bez rozumienia tych pojęć nie można zrozumieć, czym jest zbiór *Nat*. Pojęcia te należą do metajęzyka, którego używamy do zdefiniowania zbioru liczb naturalnych. Inna, formalna definicja liczb naturalnych, która nie odwołuje się do pojęcia liczby i dodawania, jest podana dalej.

Uwaga

Podana definicja zbioru liczb naturalnych przyjmuje, że liczba 0 jest najmniejszą liczbą naturalną. Spotyka się też definicje, które przyjmują, że najmniejszą liczbą naturalną jest 1. Konwencja ta wynika z historii powstawania liczb naturalnych, kiedy – do odkrycia zera – za liczby naturalne uważano tylko 1, 2, 3 itd.

Podobnie można zdefiniować zbiór dodatnich liczb parzystych:

- $2 \in ParzysteDodatnie$
- jeżeli $n \in ParzysteDodatnie$, to $n + 2 \in ParzysteDodatnie$.

Ponownie należy zwrócić uwagę, że w części rekursywnej definicji zbioru użyto konstrukcji $n + 2$, która należy do metajęzyka służącego do definiowania zbioru, i o której zakładamy, że jest dla Czytelnika jednoznacznie zrozumiała.

Inny przykład rekursywnej definicji pewnego zbioru liczb *PewneLiczby* jest następujący:

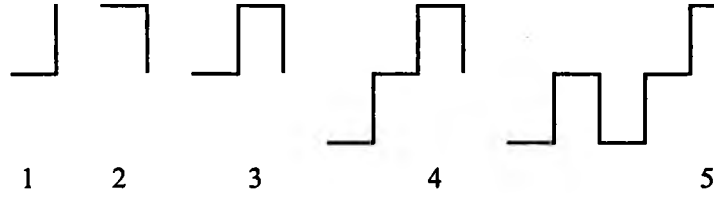
- $5, 7 \in PewneLiczby$
- jeżeli $n, m \in PewneLiczby$, to $n + m \in PewneLiczby$

Część bazowa określa, że elementami zbioru *PewneLiczby* są liczby 5 i 7. Analizując część rekursywną, łatwo się przekonać, że elementami tego zbioru będą także liczby 10, 12, 14, 15, 17, 20 itd.

Rekursywna definicja zbioru *LinieŁamane*, którego elementami są symbole graficzne – linie łamane, złożone z elementów zbioru *Kreski* – ma następującą postać:

- $|, - \in LinieŁamane$
- jeżeli $a, b \in LinieŁamane$, to linia powstająca z połączenia a oraz b w taki sposób, że jeden z końców a był połączony z jednym końcem b tak, aby poza miejscem połączenia a oraz b nie miały innych punktów wspólnych, należy również do zbioru *LinieŁamane*.

Łatwo się przekonać, że elementami zbioru *LinieŁamane* będą m.in. następujące linie:



Rys. 2.1. Elementy zbioru *LinieŁamane*

Linie o numerach 1 i 2 powstają przez różne powiązania elementów zbioru *Kreski*, linia 3 jest wynikiem połączenia linii 1 i 2, linia 4 – linii 1 i 3, a linia 5 – linii 3 i 4.

Rekursywna definicja zbioru ma charakter konstruktywny, to znaczy określa jak można skonstruować nowe elementy zbioru z innych elementów, o których już wiemy, że są elementami definiowanego zbioru. Inaczej można powiedzieć, że definicja rekursywna wyznacza pewien *algorytm* konstrukcji elementów zbioru. Algorytm jest w tym momencie rozumiany nieformalnie jako ciąg pewnych kroków obliczeniowych prowadzących do rozwiązania danego problemu. Z tego względu rekursywny sposób definiowania zbiorów jest bardzo często wykorzystywany w informatyce. Rekursywne podejście pozwala wprawdzie na definiowanie zbiorów nieskończonych, ale nie dowolnych zbiorów, lecz tylko zbiorów przeliczalnych, tzn. takich, których wszystkie elementy można zestawić w jeden ciąg (pojęcie przeliczalności zbioru jest zdefiniowane w dalszej części książki). Oczywiście, w skończonej liczbie kroków można wyznaczyć tylko skończoną liczbę elementów zbioru.

Najogólniejszy sposób definiowania zbiorów opiera się na podejściu *ekstensjonalnym*. Podejście to polega na definiowaniu zbioru przez określenie własności jego elementów. Schemat definicji zbioru ma postać

$$A =_{\text{def}} \{a \mid P(a)\}$$

Zapis ten czytamy: *do zbioru o nazwie A należą wszystkie te i tylko te elementy a, które mają własność P(a)*, czyli takie elementy, dla których wypowiedź $P(a)$ jest prawdziwa. $P(a)$ jest funkcją zdaniową, dlatego też ten sposób definiowania zbiorów nazywa się także *definiowaniem przez funkcję zdaniową*. Formalna postać funkcji zdaniowych będzie precyzyjnie określona w dalszej części książki.

Uwaga

Definicja ekstensjonalna nie określa skąd brać te elementy, które mają własność $P(a)$. Ogólne pytanie o to, które byty należy rozważać przy takim definiowaniu zbioru, wiąże się ze znanym problemem filozoficznym, określanym jako problem *powszechników* albo *uniwersaliów*.

Rozpatrzmy poprzedni przykład zbioru dodatnich liczb parzystych

$$ParzysteDodatnie =_{def} \{x \mid (x \textit{ jest liczbą naturalną}) \wedge (x > 0) \wedge (x \textit{ jest podzielne przez } 2)\}$$

Własność

$$(x \textit{ jest liczbą naturalną}) \wedge (x > 0) \wedge (x \textit{ jest podzielne przez } 2)$$

ma postać wypowiedzi złożonej. Poszczególne jej człony należą do metajęzyka – języka arytmetyki. Aby rozumieć sens całej wypowiedzi, należy rozumieć jej części składowe:

x *jest liczbą naturalną*, czyli $x \in Nat$

$x > 0$

x *jest liczbą naturalną podzielną przez* 2

oraz łączący je spójnik logiczny $\wedge$. Pierwsza z wypowiedzi wymaga rozumienia przynależności elementu do zbioru, a pozostałe wymagają elementarnej wiedzy z zakresu arytmetyki. Znaczenie spójnika $\wedge$ zostało wyjaśnione w poprzednim rozdziale.

Czasem, gdy definiujemy nowy zbiór A, wygodne jest odniesienie do innego, wcześniej ustalonego zbioru B. Piszemy wtedy

$$A =_{def} \{x \in B \mid P(x)\},$$

co jest skrótem od

$$\{x \mid x \in B \wedge P(x)\}.$$

Możemy więc napisać

$$ParzysteDodatnie =_{def} \{x \in Nat \mid (x>0) \wedge (x \textit{ jest podzielne przez } 2)\}$$

Uwaga

Czasem, definiując zbiór, zamiast symbolu $=_{def}$ używa się również innych oznaczeń, na przykład $\overset{def}{=}$, $\hat{=}$, a nawet $=$. Ostatnim symbolem należy się posługiwać

ostrożnie, gdyż jego znaczeniem podstawowym jest stwierdzanie równości (identyczności) elementów należących do pewnego zbioru.

Podejście ekstensjonalne do definiowania zbiorów jest wygodne i uniwersalne, ale nieostrożny sposób formułowania własności może prowadzić do absurdu. Znany przykład takiego absurdu jest nazywany *paradoksem Russella*[4]. Russel wykorzystał w skrajnej postaci rozumowanie, stosowane w początkowym okresie rozwoju teorii mnogości, mianowicie: niech Z będzie zbiorem zdefiniowanym następująco:

$$Z =_{\text{def}} \{X \mid X \notin X\}$$

to znaczy Z jest zbiorem – rodziną zbiorów – którego elementami są wszystkie zbiory X, mające tę własność, że nie są swoimi elementami. Odpowiedzmy teraz na pytanie: czy $Z \in Z$? Jeżeli Z jest swoim elementem, czyli $Z \in Z$, to oznacza, że ma taką samą własność jak wszystkie elementy zbioru Z, czyli $Z \notin Z$. Jeżeli natomiast Z nie jest swoim elementem, czyli $Z \notin Z$, to z definicji należy do rodziny zbiorów Z, czyli $Z \in Z$. W obu przypadkach zachodzi sprzeczność.

Paradoks ten uzasadnia dlaczego na początku rozdziału wprowadzono ograniczenie, że dla dowolnego zbioru A zachodzi $A \notin A$.

Warto zwrócić uwagę, że przypuszczenie, iż zbiór może być swoim elementem, wcale nie jest absurdalne. Rozważmy bowiem zbiór Z, którego elementami są zbiory nieskończone, to znaczy zbiory o nieskończenie wielu elementach. Z pewnością istnieje nieskończenie wiele zbiorów nieskończonych, a zatem zbiór Z jest nieskończony, czyli jest swoim elementem!

Rozpatrzmy jeszcze jeden przykład. W większości praktycznie spotykanych przypadków mamy do czynienia ze zbiorami, które nie są swoimi elementami – nazwijmy je zbiorami zwyczajnymi, w odróżnieniu od pozostałych zbiorów, które nazwiemy niezwyczajnymi. Utwórzmy teraz zbiór Z, którego elementami są wszystkie zbiory zwyczajne. Zapytajmy: czy zbiór Z jest zbiorem zwyczajnym czy niezwyczajnym? Jeśli Z jest zbiorem zwyczajnym, to wchodzi w skład swoich elementów, lecz wówczas – zgodnie z określeniem – jest on zbiorem niezwyczajnym. Jeśli natomiast Z jest zbiorem niezwyczajnym, to – zgodnie z określeniem niezwyczajności – powinien być swoim własnym elementem, a przecież elementami zbioru Z są tylko zbiory zwyczajne. Ponownie, jak w poprzednim przykładzie, w obu przypadkach zachodzi sprzeczność.

Uwaga

Potrzeba wyeliminowania sprzeczności wynikających ze zbyt swobodnego definiowania bardzo „obszernych", prowadzących do antynomii, zbiorów doprowadziła do aksjomatycznego ujęcia teorii zbiorów (zob. podrozdział 4.1). Niektóre z takich koncepcji wiązały się z wprowadzeniem pojęcia klasy. Zasadnicza idea

[4] Bertrand Russell (1872–1970).

polegała na odróżnieniu klasy od zbioru: zbiory mogą być elementami innych zbiorów, klasy zaś nie, dlatego – zamiast operować niejasnym pojęciem zbiór wszystkich zbiorów – mówi się o klasie wszystkich zbiorów.

Pojęcie klasy w wyżej przedstawionym znaczeniu należy odróżniać od, mającego zupełnie inne znaczenie, pojęcia klasy używanego w informatyce – w programowaniu i projektowaniu systemów informatycznych.

Przykład 2.1

Rozpatrzmy przykłady zbiorów używanych w językach programowania. W zasadzie wszystkie takie zbiory są zbiorami skończonymi. Wyróżnia się między innymi predefiniowane zbiory wartości związane z typami danych.

Zbiór wartości logicznych

$$Boolean =_{def} \{false, true\}$$

Zbiór całkowitoliczbowy

$$Integer =_{def} \{-N, ..., 0, ..., N\},$$

gdzie N jest liczbą naturalną określoną przez daną implementację języka.

Zbiór liczb rzeczywistych

$$Real =_{def} \{-N^*\delta, ..., 0, ..., N^*\delta\}$$

gdzie N jest liczbą naturalną, a δ jest liczbą wymierną określoną przez daną implementację języka; jest to tzw. *stałoprzecinkowa* reprezentacja liczb (w reprezentacji *zmiennoprzecinkowej* kolejne liczby są oddalone od siebie o zmienną różnicę). Warto podkreślić, że wbrew temu, co sugeruje nazwa, zbiór ten zawiera skończoną ilość liczb wymiernych.

Zbiór napisów

$$String =_{def} \{s \mid s \text{ jest skończonym ciągiem znaków ustalonego repertuaru znaków}\}$$

W praktycznej implementacji typu napisowego długość takich ciągów jest ograniczona konkretną liczbą. Przykładem takiego repertuaru znaków są na przykład znaki kodów stosowane w reprezentacji maszynowej, na przykład jednobajtowy kod ASCII (*American Standard Code for Information Interchange*) czy dwubajtowe kody *Unicode* lub BMP (*Basic Multilingual Plane*).

Definiowany przez programistę zbiór wyliczeniowy to na przykład

$$DniTygodnia =_{def} \{pon, wt, sr, czw, pt, sob, nd\}$$

gdzie *pon*, *wt*, ..., *nd* są pewnymi ustalonymi napisami. Napisy te – w odróżnieniu od napisów, które należą do zbioru *String* – są nierozkładalne, to znaczy ich fragmenty nie są elementami zbioru *DniTygodnia*.

Specyficznym dla wielu języków, nie tylko języków programowania, jest zbiór identyfikatorów. Zbiór ten będzie dalej często wykorzystywany i oznaczymy go symbolem *Ident*. Może on być definiowany na przykład tak

$$Ident =_{def} \{s \mid s \text{ jest niepustym ciągiem składającym się z liter lub cyfr, którego pierwszym elementem jest litera}\}$$

Należy zwrócić uwagę, że w treści własność definiujących zbiory występują pojęcia, o których się zakłada, że są pojęciami zrozumiałymi – są to pojęcia metajęzyka, w którym opisujemy dane własności. Na przykład w definicji zbiorów *String* oraz *Ident* takim pojęciem jest ciąg, a w definicji zbioru *DniTygodnia* takim pojęciem jest napis.

2.3. Podzbiory, równość zbiorów, zbiory potęgowe

Mówimy, że A jest *podzbiorem* zbioru B, co oznaczamy $A \subseteq B$, wtedy i tylko wtedy, gdy dla dowolnego elementu a: jeżeli $a \in A$, to także $a \in B$. Symbol $\subseteq$ nazywa się symbolem *zawierania* lub symbolem *inkluzji*. Podaną definicję zawierania zbiorów można również wyrazić formalnie

$$A \subseteq B \Leftrightarrow (\forall a \bullet a \in A \Rightarrow a \in B)$$

Z definicją wiąże się następujący komentarz: Jest to definicja w postaci *normalnej*. Składa się ona z dwóch części przedzielonych symbolem równoważności $\Leftrightarrow$, który czytamy: *wtedy i tylko wtedy*. Część po lewej stronie symbolu równoważności jest wyrażeniem zawierającym pojęcie definiowane – *definiendum*, a część po prawej stronie zawiera pojęcie definiujące – *definiens*. Poprawność definicji wymaga, aby po prawej stronie nie występowało pojęcie definiowane, gdyż byłby to przypadek „błędnego koła". Oczywiście, aby rozumieć sens definicji, pojęcia występujące w części definiującej muszą być znane. Podana definicja spełnia przedstawione wymogi, gdyż w wyrażeniu definiującym po prawej stronie nie występuje pojęcie podzbioru, a pojęcia należenia elementu do zbioru, spójnika implikacji i kwantyfikatora ogólnego były wyjaśnione wcześniej. Definicja normalna pozwala przełożyć każdy zwrot językowy zawierający wyrażenie definiowane na zwrot niezawierający tego wyrażenia. Większość definicji podawanych w książce ma postać definicji normalnej.

Łatwo zauważyć, że zachodzą własności:

$$\varnothing \subseteq A$$
$$A \subseteq A$$
$$(A \subseteq B \wedge B \subseteq C) \Rightarrow (A \subseteq C)$$

Używa się też symbolu inkluzji właściwej $\subset$. Zapis $A \subset B$ czytamy: *zbiór A zawiera się właściwie w zbiorze B*. Oznacza to, że A zawiera się w B, czyli $A \subseteq B$, oraz zbiór B zawiera przynajmniej jeden element, który nie należy do zbioru A. Formalnie

$$A \subset B \Leftrightarrow (A \subseteq B) \wedge (\exists\, a \bullet a \notin A \wedge a \in B)$$

Dwa zbiory A i B są *identyczne* albo *równe*, co oznacza się

$$A = B$$

wtedy i tylko wtedy, gdy mają dokładnie te same elementy, czyli gdy $A \subseteq B$ oraz $B \subseteq A$. Formalnie

$$A = B \Leftrightarrow (A \subseteq B) \wedge (B \subseteq A)$$

Symbol = jest tu symbolem *równości* lub *identyczności* zbiorów.

W zbiorze nie odróżnia się kolejności ani powtórzeń elementów. Na przykład zbiory:

$A =_{def} \{1, 2, 3\}$
$B =_{def} \{1, 3, 2\}$
$C =_{def} \{1, 2, 3, 2\}$

są identyczne, czyli $A = B = C$.

Jeżeli A jest zbiorem, to przez 2^A oznacza się zbiór, którego elementami są wszystkie podzbiory zbioru A. Zbiór 2^A jest nazywany *zbiorem potęgowym* zbioru A. Zbiór potęgowy jest więc rodziną zbiorów.

Uwaga

Zbiór potęgowy zbioru A oznacza się również przez $\mathbb{P}(A)$ albo $\mathcal{P}(A)$.

Przykład 2.2

Dla zbioru $A =_{def} \{a, b, c\}$ jego zbiór potęgowym 2^A jest równy zbiorowi

$$\{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$$

Postać oznaczenia zbioru potęgowego 2^A wynika z następującej własności: Jeżeli A jest zbiorem skończonym, to przez $card(A)$ oznaczamy liczbę jego elementów. Łatwo pokazać, że dla dowolnego skończonego zbioru A zachodzi

$$card(2^A) = 2^{card(A)}$$

Należy zwrócić uwagę na to, że użyty powyżej symbol równości = odnosi się do równości liczb całkowitych, podczas gdy użyty wcześniej ten sam symbol odnosił się do równości zbiorów. Symbol równości w różnych kontekstach może być używany do porównywania obiektów należących do różnych kategorii.

Uwaga

Na określenie liczności elementów skończonego zbioru A używa się również innych oznaczeń, na przykład: $\#(A)$, $|A|$.

W przypadku zbiorów nieskończonych nie można mówić o liczbie ich elementów. Można natomiast porównywać dwa zbiory pod względem równoliczności. Pojęcie równoliczności zbiorów jest zdefiniowane dalej, po wprowadzeniu pojęcia funkcji.

2.4. Operacje na zbiorach

Mając dane pewne zbiory, można z nich budować nowe zbiory. Na zbiorach wykonuje się operacje (działania), których wynikiem są nowe zbiory. Podstawowymi operacjami są: suma, przekrój, różnica i różnica symetryczna dwóch zbiorów. Działania te są zdefiniowane przez podanie własności zbiorów wynikowych.

Suma zbiorów

$$A \cup B =_{\text{def}} \{a \mid a \in A \vee a \in B\}$$

Przekrój zbiorów

$$A \cap B =_{\text{def}} \{a \mid a \in A \wedge a \in B\}$$

Różnica zbiorów

$$A \setminus B =_{\text{def}} \{a \mid a \in A \wedge a \notin B\}$$

Uwaga

Innym oznaczeniem różnicy zbiorów jest $A - B$.

Przykład 2.3

Rozpatrzmy zbiory:

$A =_{\text{def}} \{\{a, b\}, c\}$
$B =_{\text{def}} \{c, d\}$
$C =_{\text{def}} \{\{a, \{a\}\}, a\}$
$D =_{\text{def}} \{a, \{a\}\}$

Łatwo sprawdzić, że:

$A \cup B = \{\{a, b\}, c, d\}$ $\quad A \cap B = \{c\}$ $\quad A \setminus B = \{\{a, b\}\}$

$C \cup D = \{\{a, \{a\}\}, \{a\}, a\}$ $\quad C \cap D = \{a\}$ $\quad C \setminus D = \{\{a, \{a\}\}\}$

Gdy interesujące nas zbiory są podzbiorami pewnego wyróżnionego zbioru, nazywanego *zbiorem uniwersum*, używa się operacji dopełnienia zbioru. Jeżeli U jest uniwersum oraz A jest pewnym jego podzbiorem, to przez A' oznaczamy operację dopełnienia zbioru A, którą definiujemy jako

$$A' =_{\text{def}} U \setminus A$$

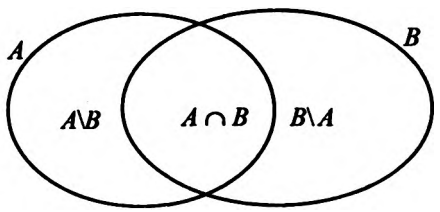
Dwa zbiory A, B nazywa się zbiorami *rozłącznymi*, jeżeli ich przekrój jest pusty, czyli gdy

$$A \cap B = \varnothing$$

Często stosowanym sposobem ilustracji operacji mnogościowych są *wykresy Venna*[5]. Zakłada się w nich, że uniwersum jest zbiór punktów na płaszczyźnie, a rozważanymi

[5] John Venn (1834–1923).

zbiorami są dowolne obszary na płaszczyźnie. Przykład takiego wykresu przedstawiono na rysunku 2.2. Dwa przecinające się owale reprezentują zbiory A oraz B. Poszczególne podobszary oznaczają odpowiednio podzbiory $A \setminus B$, $A \cap B$ oraz $B \setminus A$.



Rys. 2.2. Związki pomiędzy zbiorami

Wprowadzone operacje mają różne własności. Łatwo sprawdzić, że zachodzą następujące własności, nazywane też prawami mnogościowymi:

1. *Prawa przemienności*

$$A \cup B = B \cup A$$
$$A \cap B = B \cap A$$

2. *Prawa łączności*

$$(A \cup B) \cup C = A \cup (B \cup C)$$
$$(A \cap B) \cap C = A \cap (B \cap C)$$

3. *Prawa rozdzielności*

$$(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$$
$$(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$$

4. *Prawa de Morgana*

$$(A \cap B)' = A' \cup B'$$
$$(A \cup B)' = A' \cap B'$$

5. *Prawa dla zbioru pustego*

$$A \cap \emptyset = \emptyset$$
$$A \cup \emptyset = A$$
$$A \setminus \emptyset = A$$
$$\emptyset \setminus A = \emptyset$$
$$\emptyset' = U$$

6. *Prawa dla zbioru uniwersum*

$$A \cap U = A$$
$$A \cup U = U$$
$$A \setminus U = \emptyset$$
$$U' = \emptyset$$

Przykładowo pokażemy jak uzasadnić jedną z tych własności – pierwsze z praw de Morgana.

Własność

Zachodzi następująca równość zbiorów

$$(A \cap B)' = A' \cup B'$$

Dowód

Z definicji równości zbiorów wynika, że należy pokazać dwie inkluzje:

$$(A \cap B)' \subseteq A' \cup B'$$
$$A' \cup B' \subseteq (A \cap B)'$$

Z kolei, z definicji podzbioru wynika, że należy pokazać dwie implikacje:

$$x \in (A \cap B)' \Rightarrow x \in A' \cup B'$$
$$x \in A' \cup B' \Rightarrow x \in (A \cap B)'$$

albo – co oznacza to samo – równoważność

$$x \in (A \cap B)' \Leftrightarrow x \in A' \cup B'$$

W naszym przypadku pokażemy właśnie ostatnią równoważność. Dowód ma postać ciągu zdań połączonych spójnikami równoważności. Po prawej stronie wiersza, w którym występuje równoważność, po symbolu dwóch kresek, jest podany odpowiedni komentarz uzasadniający.

$x \in (A \cap B)' \Leftrightarrow$	-- z definicji operacji dopełnienia
$x \in U \wedge x \notin (A \cap B) \Leftrightarrow$	-- z definicji operatora nienależenia do zbioru
$x \in U \wedge \neg(x \in (A \cap B)) \Leftrightarrow$	-- z definicji iloczynu zbiorów
$x \in U \wedge \neg(x \in A \wedge x \in B) \Leftrightarrow$	-- z prawa de Morgana dla rachunku zdań
$x \in U \wedge (\neg x \in A \vee \neg x \in B) \Leftrightarrow$	-- z prawa operatora nienależenia do zbioru
$x \in U \wedge (x \notin A \vee x \notin B) \Leftrightarrow$	-- z prawa rozdzielności koniunkcji względem dysjunkcji
$x \in U \wedge x \notin A \vee x \in U \wedge x \notin B \Leftrightarrow$	-- z definicji operacji dopełnienia
$x \in A' \vee x \in B' \Leftrightarrow$	-- z definicji sumy zbiorów
$x \in A' \cup B'$	■

Uwaga

Przedstawimy kilka komentarzy związanych z dowodem. Pojęcie dowodu będzie omawiane dalej. Schemat przedstawionego tu dowodu ma postać ciągu równoważności

$$p_1 \Leftrightarrow p_1 \ldots \Leftrightarrow p_n$$

Poszczególne równoważności zachodzą na mocy definicji lub wynikają z pewnych reguł wnioskowania.

Równoważność ma własność przechodniości, co oznacza, że można stosować regułę wnioskowania:

$$\frac{p \Leftrightarrow q \quad q \Leftrightarrow r}{p \Leftrightarrow r}$$

gdzie: p, q, r są zdaniami. Na podstawie tej reguły wnioskowania można zatem stwierdzić, że

$$p_1 \Leftrightarrow p_n$$

W treści dowodu, poza powołaniem się na odpowiednie definicje występujących pojęć, powołaliśmy się na dwie własności rachunku zdań: prawa de Morgana i rozdzielność koniunkcji względem dysjunkcji. Własności rachunku zdań będą omówione w dalszej części książki, ale – dla czytelności – przedstawimy je również tutaj. Prawa de Morgana dla rachunku zdań mają postać:

$$\neg (p \wedge q) \Leftrightarrow \neg p \vee \neg q$$
$$\neg (p \vee q) \Leftrightarrow \neg p \wedge \neg q$$

Prawa rozdzielności mają natomiast postać:

$$p \wedge (q \vee r) \Leftrightarrow p \wedge q \vee p \wedge r$$
$$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$$

Wymienione równoważności noszą miano praw logicznych, co oznacza, że równoważności te są prawdziwe niezależnie od wartościowań zmiennych p, q, r. Można się o tym przekonać, budując i sprawdzając odpowiednie tablice prawdziwościowe (zob. rozdz. 1.).

Zdefiniowane dla dwóch zbiorów operacje sumy i przekroju uogólnia się na dowolne rodziny zbiorów. Niech I będzie dowolnym zbiorem, nazywanym *zbiorem indeksów*, oraz niech $\{A_i \mid i \in I\}$ będzie indeksowaną rodziną zbiorów, wtedy:

$$\bigcup_{i \in I} A_i =_{\text{def}} \{a \mid \exists\, i \in I \bullet\ a \in A_i\}$$

$$\bigcap_{i \in I} A_i =_{\text{def}} \{a \mid \forall i \in I \bullet\ a \in A_i\}$$

są *uogólnioną sumą* i *uogólnionym przekrojem* zbiorów. Uogólnioną sumę i przekrój rodziny zbiorów $\{A_i \mid i \in I\}$ będziemy też zapisywać w postaci:

$$\bigcup \{A_i \mid i \in I\}$$

$$\bigcap \{A_i \mid i \in I\}$$

Przykład 2.4

Niech $A_i =_{def} \{1, 2, ..., i\}$ będzie rodziną zbiorów, gdzie $i \in ParzysteDodatnie$. Łatwo sprawdzić, że:

$$\bigcap_{i \in ParzysteDodatnie} A_i = \{1, 2\}$$

$$\bigcup_{i \in ParzysteDodatnie} A_i = Nat \setminus \{0\}$$

Ćwiczenia

1. Przeanalizować jednoznaczność podanych niżej określeń zbiorów. Jakie związki zachodzą pomiędzy tymi zbiorami?
   a) zbiór ludzi żyjących na jednym kontynencie,
   b) zbiór narodów europejskich,
   c) zbiór zbiorów ludzi posługujących się tym samym językiem,
   d) zbiór obywateli polskich,
   e) zbiór Polaków.
2. Wskazać elementy następujących zbiorów:
   a) $\{a\}$
   b) $\{\{a\}\}$
   c) $\{\{a, b\}, \{a\}\}$
   d) $\{\{\{a\}\}, \{a\}, a\}$
   e) $\{x \in Nat \mid x^2 < 7\}$
   f) $\{x \in Wymierne \mid x^2 = 2\}$
   g) $\{x \in Wymierne \mid (x + 1)^2 < 0\}$
3. Niech A, B, C, D będą parami rozłącznymi, niepustymi zbiorami. Jakie warunki powinny spełniać te zbiory, aby zachodziły następujące równości:
   a) $\{B, C\} = \{B, C, D\}$
   b) $\{\{A, B\}, C\} = \{\{A\}, C\}$
   c) $\{\{A, B\}, \{D\}\} = \{\{A\}\}$
   d) $\{\{A, \emptyset\}, B\} = \{\{\emptyset\}\}$
4. Wykazać, że równość zbiorów $\{\{A\}, \{A, B\}\} = \{\{C\}, \{C, D\}\}$ zachodzi wtedy i tylko wtedy, gdy $A = C$ oraz $B = D$.
5. Obliczyć $A \cap B$, $A \cup B$, $A \setminus B$, $B \setminus A$ dla następujących zbiorów A i B:
   a) $A = \{\{a, b\}, c\}$ $B = \{c, d\}$
   b) $A = \{\{a, \{a\}\}, a\}$ $B = \{a, \{a\}\}$

6. Sprawdzić i uzasadnić, które spośród niżej podanych równości zachodzą bądź nie zachodzą dla dowolnych zbiorów A, B, C, D:

   a) $(A \cup B) \setminus C = (A \setminus C) \cup (B \setminus C)$
   b) $(A \setminus B) \cap (C \setminus D) = (A \cap C) \setminus (B \cup D)$
   c) $(A \cup B) \cap B = B$
   d) $(A \cap B) \cup (A \setminus B) = A$
   e) $(A \setminus B) = A \setminus (A \cap B)$
   f) $(A \setminus B) \cup B = A$

7. Niech U będzie pewnym ustalonym zbiorem, zwanym uniwersum. Jeżeli $A \subseteq U$, to $A' =_{def} U \setminus A$ nazywa się *dopełnieniem* zbioru A. Pokazać, że dla podzbiorów z uniwersum U zachodzą prawa de Morgana:

   a) $(A \cup B)' = A' \cap B'$
   b) $(A \cap B)' = A' \cup B'$

8. Dane są podzbiory A, B, C pewnego uniwersum U. Ile co najwyżej różnych zbiorów można otrzymać ze zbiorów A, B, C za pomocą operacji $\cup, \cap, \setminus$?

9. Różnica symetryczna zbiorów jest zdefiniowana następująco:

   $$A \div B =_{def} A/B \cup B/A$$

   Pokazać, że zachodzą następujące równości:

   a) $A \div B = B \div A$
   b) $(A \div B) \div C = A \div (B \div C)$
   c) $A \cap (B \div C) = (A \cap B) \div (A \cap C)$
   d) $A \cup (B \div C) = (A \div B) \div (A \cap B)$

10. Zdefiniować operacje $\cup, \cap, \setminus$ przez:

    a) $\div, \cap$
    b) $\div, \cup$
    c) $\div, /$

11. Ile elementów ma najmniejsza, niepusta rodzina zbiorów A z pewnego uniwersum U taka, że:

    a) jeżeli $A \in A$ i $B \in A$, to $A \cup B \in A$,
    b) jeżeli $A \in A$ i $B \in A$, to $A \cap B \in A$.

12. Udowodnić, że:

    a) $2^{A \cap B} = 2^A \cap 2^B$
    b) $2^{A \cup B} = \{A_1 \cup B_1 \mid A_1 \in 2^A \wedge B_1 \in 2^B\}$

13. Niech $card(A)$ oznacza liczbę elementów zbioru skończonego A. Pokazać, że dla skończonych zbiorów A oraz B zachodzi:
    a) $card(2^A) = 2^{card(A)}$
    b) $card(A \cup B) = card(A) + card(B) - card(A \cap B)$

14. Dla dowolnych zbiorów skończonych A, B i C znaleźć wzory określające:
    a) $card(A \cup B \cup C)$
    b) $card(2^{A \cup B})$
    c) $card(A \setminus B)$

15. Dowieść, że dla dowolnej rodziny zbiorów $A_1, A_2, ..., A_n$, dla $n \in Nat$, zachodzi równość

$$A_1 \cup A_2 \cup ... \cup A_n =$$
$$(A_1 \setminus A_2) \cup (A_2 \setminus A_3) \cup ... \cup (A_{n-1} \setminus A_n) \cup (A_n \setminus A_1) \cup (A_1 \cap A_2 \cap ... \cap A_n)$$

16. Niech $A_1, A_2, ..., A_n$, dla $n > 0$, będą podzbiorami zbioru U. Przez A_i^1 oznaczmy zbiór A_i, a przez A_i^0 oznaczmy dopełnienie tego zbioru, czyli A_i'. Każdy iloczyn postaci:

$$A_1^{i_1} \cap ... \cap A_n^{i_n}$$

gdzie $i_j \in \{0, 1\}$ dla $j = 1, ..., n$ nazywa się składową.

a) Pokazać, że różnych składowych jest co najwyżej 2^n.
b) Pokazać, że różne składowe są rozłączne.
c) Znaleźć sumę wszystkich składowych.
d) Udowodnić, że zbiór A_i jest równy sumie tych składowych, w których występuje czynnik postaci A_i^1.

17. Zbadać i udowodnić, które z podanych niżej związków zachodzą dla rodzin zbiorów $\{A_i \mid i \in I\}$, $\{B_i \mid i \in I\}$ oraz $\{C_{i,j} \mid i \in I, j \in J\}$:
    a) $\bigcup_{i \in I} A_i \cup \bigcup_{i \in I} B_i = \bigcup_{i \in I} (A_i \cup B_i)$,
    b) $\bigcup_{i \in I} (A_i \cap B_i) \subseteq \bigcup_{i \in I} A_i \cap \bigcup_{i \in I} B_i$,
    c) $\bigcap_{i \in I} A_i \cup \bigcap_{i \in I} B_i \subseteq \bigcup_{i \in I} (A_i \cup B_i)$,
    d) $\bigcup_{i \in I} \bigcap_{j \in J} C_{i,j} \subseteq \bigcap_{i \in I} \bigcup_{j \in J} C_{i,j}$.

18. Rodzinę $\{A_n \mid n \in Nat\}$ nazywa się *zstępującą* rodziną zbiorów, gdy $A_{n+1} \subseteq A_n$ dla $n \in Nat$. Udowodnić, że jeśli $\{A_n \mid n \in Nat\}$ oraz $\{B_n \mid n \in Nat\}$ są rodzinami zstępującymi, to:

$$\bigcap_{i \in Nat} (A_i \cup B_i) = \left(\bigcap_{i \in Nat} A_i \right) \cup \left(\bigcap_{i \in Nat} B_i \right).$$

Podać przykłady rodzin zbiorów $\{A_n \mid n \in Nat\}$ oraz $\{B_n \mid n \in Nat\}$, dla których powyższa równość nie zachodzi.

19. Rodzinę $\{A_n \mid n \in Nat\}$ nazywa się *wstępującą* rodziną zbiorów, gdy $A_n \subseteq A_{n+1}$, dla $n \in Nat$. Udowodnić, że jeśli $\{A_n \mid n \in Nat\}$ oraz $\{B_n \mid n \in Nat\}$ są rodzinami wstępującymi, to

$$\bigcup_{i \in Nat}(A_i \cap B_i) = \left(\bigcup_{i \in Nat} A_i\right) \cap \left(\bigcup_{i \in Nat} B_i\right).$$

Podać przykłady rodzin zbiorów $\{A_n \mid n \in Nat\}$ oraz $\{B_n \mid n \in Nat\}$, dla których powyższa równość nie zachodzi.

20. Funkcja f określona dla liczb rzeczywistych i o wartościach rzeczywistych jest ciągła, jeśli

$$\forall x_0 \in Rzeczywiste \bullet \forall \varepsilon > 0 \bullet \exists \delta > 0 \bullet \forall x \in Rzeczywiste \bullet |x - x_0| < \delta \Rightarrow |f(x) - f(x_0)| < \varepsilon$$

Nie używając znaku negacji, zapisać formułę: „funkcja nie jest ciągła".

21. Liczba g jest granicą w punkcie x_0 funkcji f określonej dla liczb rzeczywistych i o wartościach rzeczywistych, jeśli

$$\forall \varepsilon > 0 \bullet \exists \delta > 0 \bullet \forall x \in Rzeczywiste \bullet 0 < |x - x_0| < \delta \Rightarrow |f(x) - g| < \varepsilon$$

Nie używając znaku negacji, zapisać formułę: „liczba g nie jest granicą funkcji w punkcie x_0".

3. Relacje i funkcje

3.1. Produkty kartezjańskie

Podczas grupowania pewnych elementów w zbiory kolejność ich wyliczenia nie jest istotna. W sytuacji, gdy kolejność jest istotna, elementy się grupuje, używając pojęcia *par uporządkowanych* i *krotek*. Jeżeli $a \in A$ oraz $b \in B$ są dwoma elementami, niekoniecznie różnymi, to zapis

$<a, b>$

oznacza parę uporządkowaną, której komponentami są a oraz b. Uporządkowanie oznacza, że para $<a, b>$ nie jest tym samym, co para $<b, a>$. Dwie pary:

$<a, b>$ oraz $<c, d>$

są *identyczne*, co pisze się $<a, b> = <c, d>$, wtedy i tylko wtedy, gdy:

$a = c$ oraz $b = d$.

Występujący powyżej symbol = ma dwa znaczenia. Gdy pisze się $<a, b> = <c, d>$, oznacza to identyczność dwóch par. Gdy natomiast pisze się $a = b$, oznacza to identyczność dwóch elementów. W obu przypadkach porównuje się ze sobą obiekty różnych kategorii.

Uwaga

Para $<a, b>$ będzie też zapisywana w postaci (a, b).

Symbol identyczności, najczęściej reprezentowany symbolem = lub – rzadziej – symbolem ≈, zasługuje na wyróżnienie, z uwagi na częste użycie w różnych kontekstach. Konteksty te należy odróżniać, a w konkretnym kontekście właściwie rozumieć znaczenie identyczności. Ogólnie, symbol, który może mieć różne znaczenia, nazywa się *symbolem przeciążonym*.

Symbol identyczności, niezależnie od tego, jakie kategorie obiektów porównuje, ma pewne stałe własności. Są to własności *zwrotności*, *symetrii* i *przechodniości*. Niech a, b, c będą obiektami tego samego zbioru. Własność zwrotności oznacza, że dany obiekt jest identyczny ze sobą samym, czyli $a = a$. Własność symetrii oznacza, że jeżeli a jest identyczne z b, to b jest identyczne z a, czyli jeżeli $a = b$, to także $b = a$. Własność przechodniości oznacza, że jeżeli a jest identyczne z b oraz

b jest identyczne z c, to a jest identyczne z c, czyli jeżeli $a = b$ oraz $b = c$, to $a = c$. Symbolicznie własności te można przedstawić w postaci:

$\forall a \in A \bullet a = a$
$\forall a, b \in A \bullet (a = b) \Rightarrow (b = a)$
$\forall a, b, c \in A \bullet (a = b) \wedge (b = c) \Rightarrow (a = c)$

Parę uporządkowaną $<a, b>$ można też wyrazić jako zbiór postaci $\{a, \{a, b\}\}$. Równość zbiorów $\{a, \{a, b\}\}$ oraz $\{c, \{c, d\}\}$ odpowiada wówczas równości odpowiadających im par $<a, b>$ oraz $<c, d>$. W szczególności widać, że dwie pary $<a, b>$ oraz $<b, a>$ są różne, gdyż odpowiadające im zbiory $\{a, \{a, b\}\}$ oraz $\{b, \{a, b\}\}$ nie są identyczne. Posługiwanie się parą uporządkowaną $<a, b>$ zamiast zbiorem $\{a, \{a, b\}\}$ jest wygodniejsze i dlatego dalej będzie używana tylko taka notacja.

Przykład 3.1

Para uporządkowana postaci $<x, y>$, gdzie $x, y \in$ *LiczbyRzeczywiste*, może być interpretowana jako punkt na płaszczyźnie, o współrzędnych x, y.

Pary mają też interpretacje w programowaniu. Pary:

<nazwisko, Bach >
<nazwisko, Kant>,

gdzie: *nazwisko* $\in$ *Ident* oraz *Bach,Kant* $\in$ *Nazwiska* są przykładami danych prostych (wartościami pojedynczych pól rekordów). Podobnie, innymi przykładami danych prostych są pary:

<urodziny, XVII >
<urodziny, XVIII >

gdzie *urodziny* $\in$ *Ident* oraz *XVII, XVIII* $\in$ *LiczbyRzymskie*.

Pary postaci:

<<nazwisko, Bach >, <urodziny, XVII >>
<<nazwisko, Kant>, <urodziny, XVIII >>

reprezentują dane złożone (wartości rekordów złożonych z dwóch pól).

Ogólnie, dla dowolnej liczby naturalnej n definiuje się tzw. *n-krotki*. Jeżeli $a_1, ..., a_n$ są elementami, niekoniecznie różnymi, to

$<a_1, ..., a_n>$

jest *n-krotką*, a $a_1, ..., a_n$ są jej komponentami. Wyróżnia się więc: 0-krotkę $<>$, 1-krotkę $<a>$, 2-krotkę lub parę $<a, b>$, 3-krotkę lub trójkę $<a, b, c>$ itd.

Dwie krotki:

$<a_1, ..., a_n>$ oraz $<b_1, ..., b_m>$

są identyczne wtedy i tylko wtedy, gdy $n = m$ oraz $a_i = b_i$ dla każdego $i = 1, ..., n$.

Krotki:

<1, 1, 1>
<<1, 1>, 1>
<1, <1, 1>>

są więc różne. Pierwsza z nich jest trójką, a pozostałe są parami, w których jedna ze składowych jest również parą.

Produkt (*iloczyn*) *kartezjański* zbiorów A, B jest zbiorem par:

$$A \times B =_{def} \{<a, b> \mid a \in A \wedge b \in B\}.$$

Zauważmy, że jeżeli zbiory A i B są niepuste oraz $A \neq B$, to $A \times B \neq B \times A$.

Ogólnie – *n-krotny produkt kartezjański zbiorów* $A_1, ..., A_n$, dla $n > 1$, jest zbiorem

$$A_1 \times ... \times A_n =_{def} \{<a_1, ..., a_n> \mid a_i \in A_i \text{ dla } i = 1, ..., n\}$$

Zamiast pisać $A \times ... \times A$, gdzie A powtarza się n razy, dla $n \in Nat$, pisze się A^n. Z definicji:

$$A^0 =_{def} \{<>\}$$

$$A^1 =_{def} A.$$

Uogólnionym produktem kartezjańskim na zbiorze A nazywa się zbiór

$$\bigcup_{n \in Nat} A^n = A^0 \cup A^1 \cup A^2 \cup A^3 \cup ...$$

3.2. Relacje

Określona na zbiorach A oraz B *relacja binarna* R jest podzbiorem produktu kartezjańskiego $A \times B$, czyli $R \subseteq A \times B$.

Jeżeli $A = B$, to mówi się o relacji binarnej na A.

Jeżeli para $<a, b>$ jest elementem relacji binarnej R, to pisze się $<a, b> \in R$. Czasem używa się równoważnego zapisu aRb.

Jeżeli $R_1, R_2 \subseteq A \times B$, to równość relacji $R_1 = R_2$ jest równością zbiorów par reprezentowanych przez R_1 oraz R_2.

Ponownie warto zwrócić uwagę na nową rolę symbolu równości =. Tym razem symbol ten oznacza równość relacji, podczas gdy wcześniej oznaczał równość elementów w obrębie zbioru, równość zbiorów oraz równość krotek.

Zbiór wszystkich relacji binarnych określonych na zbiorach A i B będzie oznaczany przez $2^{A \times B}$, tzn.

$2^{A\times B} =_{\text{def}} \{R \mid R \subseteq A\times B\}$

Przy wprowadzonych oznaczeniach zapisy:

$R \subseteq A\times B$ oraz $R \in 2^{A\times B}$

są równoważne.

Uwaga

Na określenie zbioru relacji na produkcie kartezjańskim $A \times B$ używa się również innych oznaczeń, na przykład: $A \leftrightarrow B$ lub $\mathrm{P}(A\times B)$.

Zapis postaci

$R \subseteq A\times B$

nazywa się *sygnaturą relacji*. Symbol R jest *nazwą* relacji, a wyrażenie $A \times B$, gdzie A oraz B są nazwami zbiorów, jest *typem* relacji.

Jeżeli R jest relacją binarną na $A\times B$, to jej *dziedziną* jest zbiór

$dom(R) =_{\text{def}} \{a \in A \mid \exists\, b \in B \bullet \langle a, b\rangle \in R\}$

a jej *przeciwdziedziną* jest zbiór

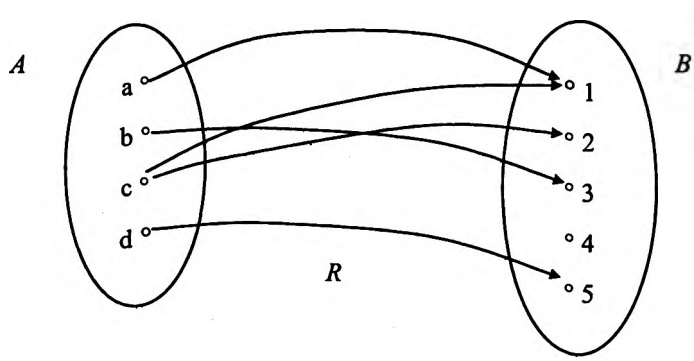
$ran(R) =_{\text{def}} \{b \in B \mid \exists\, a \in A \bullet \langle a, b\rangle \in R\}$

Relacja binarna $R \subseteq A\times B$ ma swoją *relację odwrotną* $R^{-1} \subseteq B\times A$, zdefiniowaną następująco:

$R^{-1} =_{\text{def}} \{\langle b, a\rangle \mid \langle a, b\rangle \in R\}$

Łatwo zauważyć, że $(R^{-1})^{-1} = R$.

Wprowadzone pojęcia można zilustrować graficznie. Rozpatrzmy przykład relacji binarnej zdefiniowanej na zbiorach $A =_{\text{def}} \{a, b, c, d\}$ oraz $B =_{\text{def}} \{1, 2, 3, 4, 5\}$, przedstawiony na rysunku 3.1.



Rys. 3.1. Graficzna ilustracja relacji

Łuki prowadzące od elementów zbioru A do elementów zbioru B reprezentują pojedyncze pary – elementy relacji R. Z rysunku wynika, że

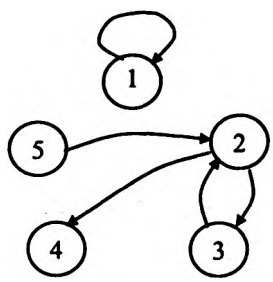
$$R = \{<a, 1>, <b, 3>, <c, 1>, <c, 2>, <d, 5>\}$$

Ponadto $dom(R) = A$ oraz $ran(R) = \{1, 2, 3, 5\} \subset B$. Przedstawienie na rysunku, zgodnie z tą samą konwencją, relacji odwrotnej R^{-1} polegałoby na odwróceniu kierunku strzałek.

Gdy ma się do czynienia z relacjami binarnymi określonymi na jednym zbiorze, czyli relacjami o sygnaturze $R \subseteq A^2$, bardzo przejrzystym sposobem reprezentacji graficznej są grafy. Formalnie grafy są definiowane dalej, tutaj ograniczamy się do przykładu. Niech $A =_{def} \{1, 2, 3, 4, 5\}$ oraz

$$R =_{def} \{<1, 1>, <3, 2>, <2, 3>, <2, 4>, <5, 2>\}$$

Graf reprezentujący relację R jest pokazany na rysunku 3.2.



Rys. 3.2. Graficzna ilustracja relacji R

Wierzchołki grafu reprezentują elementy zbioru A, a łuki grafu reprezentują elementy relacji w taki sposób, że para $<a, b> \in R$ jest reprezentowana przez łuk wychodzący z wierzchołka a i prowadzący do wierzchołka b.

Relacje mają różne zastosowanie w informatyce. Tablice w bazach danych są typowym przykładem relacji.

Przykład 3.2

W pewnej bazie danych dane są dwie tablice:

Tablica 3.1

Nazwisko	Wiek urodzin
Bach	*XVII*
Frege	*XVIII*
Leibnitz	*XVII*
Tarski	*XX*

Tablica 3.2

Nazwisko	Zawód
Bach	*Muzyk*
Frege	*Logik*
Leibnitz	*Filozof*
Tarski	*Matematyk*

Każda z tablic reprezentuje pewną relację. Pierwsza jest relacją typu *Nazwiska* × *Liczby Rzymskie*, druga zaś jest typu *Nazwiska* × *Zawody*. Relacje te można przedstawić w postaci mnogościowej przez wyliczenie odpowiednich par:

$\{<Bach, XVII>, <Frege, XVIII>, <Leibnitz, XVII>, <Tarski, XX>\}$

$\{<Bach, Muzyk>, <Frege, Logik>, <Leibnitz, Filozof>, <Tarski, Matematyk>\}$

Pojęcie relacji binarnej uogólnia się na *relację n-krotną R* jako dowolny podzbiór *n*-krotnego, produktu kartezjańskiego

$R \subseteq A_1 \times ... \times A_n$ dla $n \in Nat \setminus \{0, 1\}$

3.3. Operacje na relacjach

Relacje są zbiorami, można zatem na nich wykonywać wszystkie wcześniej zdefiniowane operacje mnogościowe. Jeżeli na przykład R, $Q \subseteq A \times B$, to zbiory $R \cup Q$, $R \cap Q$, $R \setminus Q$ są również relacjami na produkcie kartezjańskim $A \times B$.

Wprowadza się też specyficzne operacje mnogościowe. Operacje takie występują na przykład w systemach zarządzania bazami danych.

Przykład 3.3

Rozpatruje się operację *złączenia* dwóch relacji $R \subseteq A \times B$ oraz $Q \subseteq A \times C$. Operacja ta, oznaczana tu przez $R \oplus Q$, jest zdefiniowana następująco:

Jeśli $dom(R) = dom(Q)$, to $R \oplus Q \subseteq A \times B \times C$ jest relacją:

$$R \oplus Q =_{def} \{<a, b, c> \mid <a, b> \in R \wedge <a, c> \in Q\}$$

Jeżeli za R oraz Q weźmie się relacje zdefiniowane przez tablicę 3.1 i tablicę 3.2 w poprzednim przykładzie, to widać, że $dom(R) = dom(Q)$, a wynikową relację $R \oplus Q$ przedstawia tablica 3.3.

Tablica 3.3

Nazwisko	Wiek urodzin	Zawód
Bach	*XVII*	*Muzyk*
Frege	*XVIII*	*Logik*
Leibnitz	*XVII*	*Filozof*
Tarski	*XX*	*Matematyk*

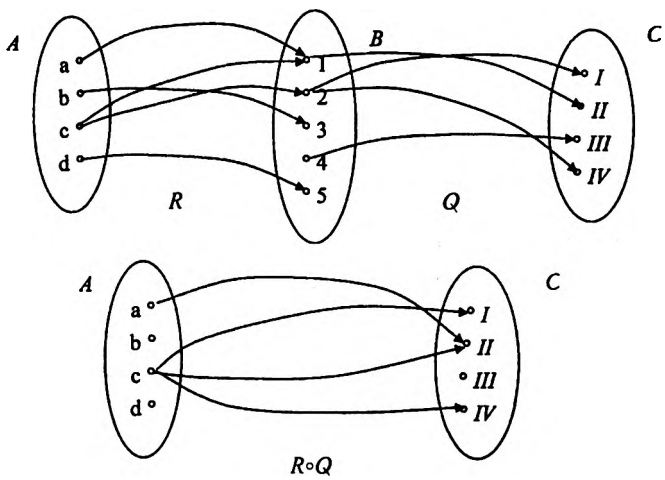
Nową operacją jest *złożenie* (*superpozycja*) dwóch relacji $R \subseteq A \times B$ oraz $Q \subseteq B \times C$. Jest to nowa relacja, zapisywana w postaci $R \circ Q$, zdefiniowana następująco:

$$R \circ Q =_{def} \{<a, c> \mid \exists\, b \in B \bullet <a, b> \in R \wedge <b, c> \in Q\}$$

Graficzną ilustracją złożenia dwóch relacji $R \subseteq A \times B$ oraz $Q \subseteq B \times C$, gdzie

$$A =_{def} \{a, b, c, d\},\ B =_{def} \{1, 2, 3, 4, 5\},\ C =_{def} \{I, II, III, IV\}$$

jest rysunek 3.3.



Rys. 3.3. Graficzna ilustracja złożenia relacji

Górna część rysunku przedstawia relacje R oraz Q, a dolna część – złożenie $R \circ Q$.

Łatwo sprawdzić, że złożenie relacji jest operacją łączną, to znaczy

$$(R \circ Q) \circ S = R \circ (Q \circ S)$$

ale nie jest operacją przemienną, to znaczy

$$R \circ Q \neq Q \circ R$$

Dla dowolnej liczby naturalnej n, *n-krotnym złożeniem* relacji binarej $R \subseteq A^2$ jest relacja R^n, zdefiniowana *indukcyjnie* w sposób następujący:

$$R^0 =_{def} \{<a, a> \mid a \in A\}$$

$$R^{n+1} =_{def} R^n \circ R \text{ dla } n \in Nat$$

Relację R^0 nazywa się *relacją identycznościową* lub *tożsamościową* na A.

Jeżeli $R \subseteq A \times B$, to *obrazem zbioru* $A_1 \subseteq A$ wyznaczonym przez relację R jest zbiór

$$R(A_1) =_{def} \{b \in B \mid \exists\, a \in A_1 \bullet \langle a, b \rangle \in R\}$$

Łatwo pokazać, że dla $A_1, A_2 \subseteq A$ zachodzą własności:

$$R(A_1 \cup A_2) = R(A_1) \cup R(A_2)$$
$$R(A_1 \cap A_2) \subseteq R(A_1) \cap R(A_2)$$
$$R(dom(R)) = ran(R)$$

Jeżeli $R \subseteq A \times B$, to *przeciwobrazem zbioru* $B_1 \subseteq B$ wyznaczonym przez relację R jest zbiór $R^{-1}(B_1)$.

3.4. Podstawowe rodzaje relacji binarnych

Relacje binarne na A, czyli relacje $R \subseteq A^2$, mogą się charakteryzować różnymi własnościami. Wśród podstawowych własności wyróżnia się między innymi własności *zwrotności*, *przeciwzwrotności*, *symetrii*, *przeciwsymetrii*, *antysymetrii*, *przechodniości* i *spójności*. Własności te są definiowane następująco:

zwrotność
dla dowolnego $a \in A$ zachodzi: $\langle a, a \rangle \in R$
– symbolicznie: $\forall a \in A \bullet \langle a, a \rangle \in R$

przeciwzwrotność
dla dowolnego $a \in A$ zachodzi: $\langle a, a \rangle \notin R$
– symbolicznie: $\forall a \in A \bullet \langle a, a \rangle \notin R$

symetria
dla dowolnych $a, b \in A$ zachodzi:
jeżeli $\langle a, b \rangle \in R$, to również $\langle b, a \rangle \in R$
– symbolicznie: $\forall a, b \in A \bullet \langle a, b \rangle \in R \Rightarrow \langle b, a \rangle \in R$

przeciwsymetria
dla dowolnych $a, b \in A$ zachodzi:
jeżeli $\langle a, b \rangle \in R$, to $\langle b, a \rangle \notin R$
– symbolicznie: $\forall a, b \in A \bullet \langle a, b \rangle \in R \Rightarrow \langle b, a \rangle \notin R$

antysymetria
dla dowolnych $a, b \in$ A zachodzi:
jeżeli $\langle a, b \rangle \in R$ oraz $\langle b, a \rangle \in R$, to $a = b$
– symbolicznie: $\forall a, b \in A \bullet \langle a, b \rangle \in R \wedge \langle b, a \rangle \in R \Rightarrow a = b$

przechodniość
dla dowolnych $a, b, c \in A$ zachodzi:
jeżeli $\langle a, b \rangle \in R$ oraz $\langle b, c \rangle \in R$, to $\langle a, c \rangle \in R$
– symbolicznie: $\forall a, b, c \in A \bullet \langle a, b \rangle \in R \wedge \langle b, c \rangle \in R \Rightarrow \langle a, c \rangle \in R$

spójność
dla dowolnych $a, b \in A$ zachodzi:
jeżeli $a \neq b$, to $\langle a, b \rangle \in R$ lub $\langle b, a \rangle \in R$
– symbolicznie: $\forall a, b \in A \bullet a \neq b \Rightarrow \langle a, b \rangle \in R \vee \langle b, a \rangle \in R$

Niekiedy mamy do czynienia z sytuacjami, gdy zachodzi potrzeba takiego rozszerzenia relacji, aby uzyskały pewne z przedstawionych własności. Niech R będzie dowolną relacją binarną na A. *Zwrotnym domknięciem* relacji R jest relacja zdefiniowana jako

$$R \cup R^0$$

gdzie R^0 jest *relacją identycznościową* na A.

Symetrycznym domknięciem relacji R jest relacja zdefiniowana jako

$$R \cup \{<b, a> \mid <a, b> \in R\} \text{ albo } R \cup R^{-1}$$

Przechodnim domknięciem relacji R jest relacja oznaczana symbolem R^+, zdefiniowana jako

$$R^+ =_{\text{def}} \bigcup_{n \in Nat \setminus \{0\}} R^n$$

Zwrotne, przechodnie (*tranzytywne*) *domknięcie* relacji R na zbiorze A jest to relacja, oznaczana symbolem R^*, zdefiniowana następująco:

$$R^* =_{\text{def}} \bigcup_{n \in Nat} R^n$$

Podane wyżej definicje domknięcia relacji oznaczają, że po domknięciu relacja ma odpowiednio własność zwrotności, symetrii i przechodniości. Domknięcie relacji uogólnia się ze względu na dowolną własność w sposób następujący: Niech P będzie pewną własnością oraz R pewną relacją binarną. Mówimy, że relacja R_P jest *domknięciem relacji R względem własności P* wtedy i tylko wtedy, gdy:

1. relacja R_P ma własność P,
2. $R \subseteq R_P$,
3. nie istnieje inna relacja Q, która ma własność P taką, że $Q \subseteq R_P$.

3.5. Relacja równoważności

Bardzo ważna jest relacja *równoważności*, będąca dowolną relacją binarną, która jest zwrotna, symetryczna i przechodnia.

Dla relacji równoważności R określonej na zbiorze A definiuje się zbiory nazywane *klasami abstrakcji*. Dla dowolnego elementu a zbioru A definiuje się mianowicie zbiór

$$\{b \in A \mid <a, b> \in R\}$$

Zbiór taki oznacza się przez $[a]_R$ i nazywa się *klasą abstrakcji generowaną przez element a względem relacji R*.

Twierdzenie 3.1

Zbiór klas abstrakcji ma następujące własności:

1. $\bigcup_{a \in A} [a]_R = A$
2. $<a, b> \in R$ wtedy i tylko wtedy, gdy $[a]_R = [b]_R$
3. jeżeli $[a]_R \neq [b]_R$, to $[a]_R \cap [b]_R = \emptyset$

Dowód

Własność 1

Z definicji równości zbiorów wynika, że należy pokazać:

a) $\bigcup_{a \in A} [a]_R \subseteq A$

b) $A \subseteq \bigcup_{a \in A} [a]_R$

Przypadek a)

Niech $x \in \bigcup_{a \in A} [a]_R$. Należy pokazać, że $x \in A$.

Z definicji uogólnionej sumy zbiorów wynika, że istnieje takie $a \in A$, że $x \in [a]_R$.

Z definicji klasy abstrakcji wynika, że $[a]_R \subseteq A$.

Z obu tych faktów, na podstawie definicji podzbioru, wynika, że $x \in A$, czyli pokazaliśmy, że $x \in \bigcup_{a \in A} [a]_R \Rightarrow x \in A$.

Przypadek b)

Niech $x \in A$. Należy pokazać, że $x \in \bigcup_{a \in A} [a]_R$.

Z definicji klasy abstrakcji i własności zwrotności relacji R wynika, że $x \in [x]_R$.

Z tego faktu zatem, na mocy definicji uogólnionej sumy zbiorów, wynika, że $x \in \bigcup_{a \in A} [a]_R$

Własność 2

Z definicji równoważności wynika, że należy pokazać dwie implikacje:

a) jeżeli $<a, b> \in R$, to $[a]_R = [b]_R$,

b) jeżeli $[a]_R = [b]_R$, to $<a, b> \in R$.

Przypadek a)

Niech $<a, b> \in R$. Należy pokazać, że $[a]_R = [b]_R$.

Pokażemy, że jeśli $<a, b> \in R$, to $[a]_R \subseteq [b]_R$ oraz, że jeśli $<a, b> \in R$, to $[b]_R \subseteq [a]_R$.

Z założenia, że $<a, b> \in R$ i z definicji klasy abstrakcji wynika, że $b \in [a]$.

Niech $x \in [a]_R$. Z definicji klasy abstrakcji wynika, że $<a, x> \in R$, co z własności symetrii relacji równoważności pociąga, że $<x, a> \in R$.

Z własności przechodniości relacji równoważności, na podstawie stwierdzeń, że $<x, a> \in R$ oraz $<a, b> \in R$ wynika, że $<x, b> \in R$. Ponownie, z własności symetrii relacji R oraz z definicji klasy abstrakcji wynika, że $x \in [b]_R$.

Pokazaliśmy, że $x \in [b]_R \Rightarrow x \in [b]_R$, czyli $[a]_R \subseteq [b]_R$.

Wykazanie, że jeśli $<a, b> \in R$, to $[b]_R \subseteq [a]_R$ przebiega analogicznie do pokazanego wyżej.

Przypadek b)

Niech $[a]_R = [b]_R$. Należy pokazać, że $<a, b> \in R$.

Z definicji klasy abstrakcji i zwrotności relacji wynika, że $a \in [a]_R$ i $b \in [b]_R$. Na podstawie równości zbiorów $[a]_R = [b]_R$ stwierdzamy, że $a, b \in [a]_R$, stąd – na podstawie definicji klasy abstrakcji – wynika, że $<a, b> \in R$.

Własność 3

Należy pokazać, że jeżeli $[a]_R \neq [b]_R$, to $[a]_R \cap [b]_R = \emptyset$.

Dowód przeprowadzimy metodą nie wprost. Metodę tę stosuje się do twierdzeń, które mają postać implikacyjną. W rozważanym przypadku założeniem – poprzednikiem implikacji – jest $[a]_R \neq [b]_R$), a tezą – następnikiem implikacji – jest $[a]_R \cap [b]_R = \emptyset$. Dowód nie wprost polega na przyjęciu założenia, nazywanego założeniem dowodu nie wprost, które jest negacją tezy. Następnie należy pokazać, że z tego założenia wynika sprzeczność z założeniem twierdzenia.

W rozpatrywanym przypadku należy pokazać, że

jeżeli $[a]_R \cap [b]_R \neq \emptyset$, to $[a]_R = [b]_R$.

Jeżeli $[a]_R \cap [b]_R \neq \emptyset$, to oznacza, że istnieje element $x \in [a]_R \cap [b]_R$. Z definicji przekroju zbiorów wynika, że $x \in [a]_R$ i $x \in [b]_R$, co na podstawie definicji klasy abstrakcji oznacza, że $<a, x> \in R$ oraz $<b, x> \in R$. Z własności symetrii i przechodniości relacji R wynika, że $<a, b> \in R$, stąd – na mocy udowodnionej wyżej własności 2 – wynika, że $[a]_R \cap [b]_R = \emptyset$, co oznacza sprzeczność z założeniem twierdzenia. ■

Z udowodnionych własności wynika, że jeżeli zbiorze A jest zdefiniowana relacja równoważności, to relacja ta wyznacza podział zbioru A na rozłączne podzbiory (klasy abstrakcji). Podział taki nazywa się też *partycją*. Zbiór, którego elementami są wszystkie klasy abstrakcji, nazywa się *zbiorem ilorazowym* zbioru A względem relacji R i oznacza się A/R, czyli

$$A/R =_{def} \{[a]_R \mid a \in A\}$$

Przykład 3.4

Niech $A =_{def} \{1, 2, 3, 4, 5\}$ oraz relacja $R \subseteq A^2$ jest zdefiniowana następująco:

$R =_{def} \{<1, 1>, <2, 2>, <3, 3>, <4, 4>, <5, 5>, <3, 2>, <2, 3>, <2, 5>, <5, 2>, <3, 5>, <5, 3>\}$

Łatwo sprawdzić, że relacja jest zwrotna, symetryczna i przechodnia, czyli jest relacją równoważności. Wyznaczone przez poszczególne elementy zbioru A klasy równoważności są następujące:

$[1]_R = \{1\}$
$[2]_R = [3]_R = [5]_R = \{2, 3, 5\}$
$[4]_R = \{4\}$

Zbiór ilorazowy A/R wyznaczony przez relację R ma postać

$A/R = \{\{1\}, \{2, 3, 5\}, \{4\}\}$

3.6. Relacje porządku

Oprócz relacji równoważności ważną grupę stanowią *relacje porządku*. Wyróżnia się:

- relację *quasi-porządkującą*, gdy jest zwrotna i przechodnia,
- relację *częściowo porządkującą w ścisłym sensie*, gdy jest antysymetryczna i przechodnia,
- relację *częściowo porządkującą*, gdy jest zwrotna, antysymetryczna i przechodnia,
- relację *liniowego porządku w ścisłym sensie*, gdy jest antysymetryczna, przechodnia i spójna,
- relację *liniowego porządku* (czasem też krótko: *relację porządku*), gdy jest zwrotna, antysymetryczna, przechodnia i spójna, czyli, gdy jest relacją częściowego porządku i relacją spójną.

Zbiór, na którym jest określona pewna relacja porządku częściowego, nazywa się *zbiorem częściowo uporządkowanym*, a zbiór, na którym określono relację porządku liniowego – *zbiorem liniowo* (albo *całkowicie*) *uporządkowanym*.

Przykład 3.5

Rozważmy rodzinę wszystkich podzbiorów dowolnego zbioru U, czyli zbiór potęgowy 2^U. Zbiór potęgowy 2^U jest zbiorem częściowo uporządkowanym przez relację $R \subseteq 2^U \times 2^U$, która jest określona następująco:

$$R = \{<A, B> \in 2^U \times 2^U \mid A \subseteq B\}$$

Relacja R jest relacją częściowego porządku. Istotnie, R jest relacją zwrotną, gdyż dla dowolnego $A \in 2^U$ zachodzi $<A, A> \in R$, dlatego, że $A \subseteq A$. R jest relacją antysymetryczną, gdyż – jeżeli $<A, B> \in R$ oraz $<B, A> \in R$ – co oznacza, że $A \subseteq B$ oraz $B \subseteq A$, to $A = B$. R jest też relacją przechodnią, gdyż z faktu, że $<A, B> \in R$ oraz $<B, C> \in R$, co oznacza, że $A \subseteq B$ oraz $B \subseteq C$, wynika, że $<A, C> \in R$, co oznacza, że $A \subseteq C$. Relacja R nie jest natomiast relacją liniowego porządku, gdyż nie jest spójna.

Przykład 3.6

Relacją liniowego porządku jest relacja $\leq$ określona na różnych zbiorach liczbowych, na przykład *Nat*, *Całkowite*, *Wymierne* lub *Rzeczywiste*. Używając tej relacji, posługujemy się notacją $a \leq b$, zamiast $<a, b> \in \leq$. Zachodzą oczywiście własności:

$$\forall a \in Rzeczywiste \bullet a \leq b$$

$$\forall a, b \in Rzeczywiste \bullet a \leq b \wedge a \leq b \Rightarrow a = b$$

$$\forall a, b, c \in Rzeczywiste \bullet a \leq b \wedge b \leq c \Rightarrow a \leq c$$

$$\forall a, b \in Rzeczywiste \bullet a \neq b \Rightarrow a \leq b \vee a \leq b$$

Nie jest natomiast relacją liniowego porządku relacja $<$, gdyż nie spełnia wymogu zwrotności, to znaczy nie jest prawdą, że $a < a$ dla dowolnego $a \in Rzeczywiste$.

Zestaw relacji częściowego porządku $R_1, ..., R_n$, zdefiniowanych odpowiednio na zbiorach $A_1, ..., A_n$, można wykorzystać do zdefiniowania nowej relacji częściowego porządku R, zdefiniowanej na produkcie kartezjańskim $A_1 \times ... \times A_n$. Przykładem jest *leksykograficzne złożenie* relacji $R_1, ..., R_n$, określone jako relacja R, nazywana *relacją porządku leksykograficznego* (alfabetycznego), zdefiniowana na $A_1 \times ... \times A_n$ w sposób następujący:

$$< a_1, ..., a_n > R < b_1, ..., b_n >$$

wtedy i tylko wtedy, gdy istnieje $i \in \{1, ..., n\}$ takie, że dla każdego $j < i$ zachodzi $a_j = b_j$ oraz $a_i \, R_i \, b_i$.

Niektóre własności relacji częściowego porządku można wyrazić za pomocą tzw. elementów wyróżnionych. Niech $\preccurlyeq$ będzie relacją częściowego porządku na zbiorze A oraz niech $B \subseteq A$.

Uwaga

Symbol $\preccurlyeq$ jest często stosowany na oznaczenie relacji częściowego porządku ze względu na graficzne podobieństwo do symbolu $\leq$ przy jednoczesnym podkreśleniu, że dziedziną relacji nie muszą być tylko zbiory liczbowe.

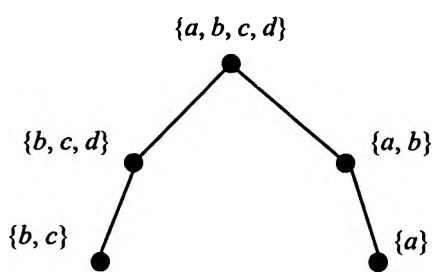
Mówimy, że $a \in A$ jest:

- *elementem najmniejszym* w zbiorze B, jeśli $\forall b \in B \bullet a \preccurlyeq b$,
- *elementem największym* w zbiorze B, jeśli $\forall b \in B \bullet b \preccurlyeq a$,
- *elementem minimalnym* w zbiorze B, jeśli $\forall b \in B \bullet \neg(b \preccurlyeq a)$,
- *elementem maksymalnym* w zbiorze B, jeśli $\forall b \in B \bullet \neg(a \preccurlyeq b)$,
- *ograniczeniem dolnym* zbioru B, jeśli $\forall b \in B \bullet a \preccurlyeq b$ (zauważmy, że a nie musi należeć do zbioru B, element najmniejszy zbioru B jest zatem jego ograniczeniem dolnym),
- *ograniczeniem górnym* zbioru B, jeśli $\forall b \in B \bullet b \preccurlyeq a$ (zauważmy, że a nie musi należeć do zbioru B, element największy zbioru B jest zatem jego ograniczeniem górnym),
- *kresem dolnym* (*infimum*) zbioru B, jeśli a jest elementem największym w zbiorze wszystkich ograniczeń dolnych zbioru B,
- *kresem górnym* (*supremum*) zbioru B, jeśli a jest elementem najmniejszym w zbiorze wszystkich ograniczeń górnych zbioru B.

Przykład 3.7

Rozpatrzmy zbiór potęgowy A zbioru $\{a, b, c, d\}$, czyli $A = 2^{\{a, b, c, d\}}$, z uporządkowaniem częściowym, określonym przez inkluzję. Niech $B = \{\{a\}, \{a, b\}, \{b, c\}, \{b, c, d\}, \{a, b, c, d\}\}$. W zbiorze B są dwa elementy minimalne $\{a\}$ i $\{b, c\}$ oraz jeden element największy $\{a, b, c, d\}$.

Ilustracją związków pomiędzy elementami zbioru częściowo uporządkowanego jest diagram (graf – zob. p. 3.10) Haasego na rysunku 3.4. Wierzchołki diagramu reprezentują elementy zbioru, a krawędzie diagramu łączą ze sobą te wierzchołki x, y, tu elementy zbioru B, pomiędzy którymi zachodzi związek $x \preccurlyeq y$ oraz nie istnieje taki element z, że $x \preccurlyeq z$ oraz $z \preccurlyeq y$.



Rys. 3.4. Diagram Haasego dla zbioru B

W zbiorze B nie ma elementu najmniejszego, element $\{a, b, c, d\}$ jest natomiast jego kresem górnym.

Pomiędzy elementami wyróżnionymi zachodzą następujące związki:

Twierdzenie 3.2

Niech $\preccurlyeq$ będzie relacją częściowego porządku na zbiorze A oraz niech $B \subseteq A$, wtedy:

1. W zbiorze B istnieje co najwyżej jeden element największy i co najwyżej jeden element najmniejszy.
2. Zbiór B ma co najwyżej jeden kres górny i jeden kres dolny.
3. Jeśli b jest największym elementem w zbiorze B, to jest on jedynym elementem maksymalnym w zbiorze B oraz jego kresem górnym.
4. Jeśli b jest najmniejszym elementem w zbiorze B, to jest on jedynym elementem minimalnym w zbiorze B oraz jego kresem dolnym.

Dowód pozostawiamy jako ćwiczenie.

3.7. Funkcje

Niech A oraz B będą dwoma dowolnymi zbiorami. *Funkcją* albo *odwzorowaniem* z A w B nazywa się taką relację binarną $f \subseteq A \times B$, że dla każdego elementu $a \in A$ istnieje co najwyżej jeden element $b \in B$ taki, że $<a, b> \in f$.

Inne, równoważne sformułowanie tej samej własności:

dla każdego elementu $a \in A$, jeżeli $<a, b> \in f$ oraz $<a, c> \in f$, to $b = c$

W symbolicznym zapisie własność ta przyjmuje postać

$$\forall a \in A \bullet <a, b> \in f \wedge <a, c> \in f \Rightarrow b = c$$

W podanej definicji funkcji f zawiera się możliwość, że dla danego $a \in A$ nie istnieje taki element $b \in B$, że $<a, b> \in f$. Oznacza to, że dla $a \in A$ funkcja f jest nieokreślona. Fakt, że para $<a, b> \in f$ jest elementem funkcji f, będzie również zapisywany w postaci

$$f(a) = b.$$

Element a nazywa się *argumentem* funkcji f, a b nazywa się *wartością* funkcji f dla argumentu a.

Napis

$$f : A \rightarrow B$$

nazywa się *sygnaturą* funkcji; symbol f jest *nazwą* funkcji, a wyrażenie $A \rightarrow B$, gdzie A oraz B są nazwami zbiorów, jest *typem* funkcji.

Ogólnie, funkcja może mieć n argumentów ($n \in Nat$). Sygnatura takiej funkcji ma postać

$$f: A_1 \times ... \times A_n \to B$$

W skrajnym przypadku, gdy funkcja ma zero argumentów – nazywa się ją funkcją *zeroargumentową* lub *stałą*, a jej sygnaturę zapisujemy

$$f: \to B$$

Dla funkcji o sygnaturze $f: A_1 \times ... \times A_n \to B$, fakt, że $<a_1, ..., a_n, b> \in f$, zapisuje się również w postaci

$$f(a_1, ..., a_n) = b.$$

Zapis wartości funkcji w postaci

$$f(a_1, ..., a_n)$$

jest zapisem w tak zwanej konwencji *prefiksowej* lub *przedrostkowej*. Inną konwencją, która nie będzie używana, jest notacja *przyrostkowa* (*postfiksowa*), nazywana też *odwrotną notacją polską*, na cześć polskiego logika Łukasiewicza[6], który ją wprowadził. W tej notacji zapisowi wartości funkcji dla argumentów $a_1, ..., a_n$ odpowiada zapis

$$(a_1, ..., a_n)f$$

Notacja ta jest stosowana w algorytmicznym obliczaniu wartości wyrażeń stanowiących złożenie wielu funkcji.

W przypadku funkcji dwuargumentowych, oprócz podanych notacji, powszechnie stosuje się notację *wrostkową* (*infiksową*). Wartość funkcji $f(a_1, a_2)$, dla argumentów a_1, a_2, zapisuje się w postaci

$$a_1 f a_2$$

Deklarację użycia notacji infiksowej można zaznaczyć w sygnaturze, pisząc

$$\_f\_ : A_1 \times A_2 \to B$$

Podkreślenia po obu stronach symbolu f wskazują na miejsca umieszczania jej argumentów.

Niech $f: A \to B$. Tak jak poprzednio, przez $dom(f)$ i $ran(f)$ oznacza się odpowiednio *dziedzinę* i *przeciwdziedzinę* funkcji f.

Jeżeli $dom(f) = A$, to f nazywa się funkcją *całkowicie określoną*, albo – krótko – *całkowitą*. Zbiór wszystkich funkcji całkowicie określonych z A do B oznacza się $A \to B$. Zbiór A nazywa się zbiorem *źródłowym*, a B – zbiorem *docelowym* funkcji. Zbiór wszystkich funkcji całkowitych z A w B oznacza się też przez B^A.

[6] Jan Łukasiewicz (1878–1956).

W przypadku, gdy $dom(f) \subset A$, funkcję f nazywa się *częściowo określoną*, albo – krótko – *częściową*. Funkcja częściowa f jest nieokreślona dla elementów nienależących do jej dziedziny, czyli do zbioru $A \backslash dom(f)$. Elementowi $a \in A \backslash dom(f)$ nie odpowiada żaden element ze zbioru B. Fakt ten zapisuje się niekiedy, pisząc $f(a) = \perp$, gdzie symbol $\perp$ oznacza *niezdefiniowane*.

Wykorzystując symbol $\perp$, zbiór wszystkich funkcji z A do B oznacza się $(B \cup \perp)^A$.

Jeżeli $ran(f) = B$, to funkcję f nazywa się *surjekcją* albo funkcją „*na*".

Jeżeli dla dwóch różnych argumentów a_1, a_2 funkcja f przyjmuje różne wartości $f(a_1)$, $f(a_2)$, to nazywa się ją funkcją *różnowartościową* albo *injekcją*.

Funkcję f, która jest całkowicie określona, jest surjekcją oraz injekcją, nazywa się funkcją *wzajemnie jednoznaczną* albo *bijekcją*.

Bijekcję, która jest funkcją o tej samej dziedzinie i przeciwdziedzinie, czyli o sygnaturze $f: A \rightarrow A$, nazywa się *permutacją*.

Funkcję f nazywa się funkcją *skończoną*, gdy dziedzina funkcji $dom(f)$ jest zbiorem skończonym.

Jeżeli relacja odwrotna f^{-1} dla funkcji $f: A \rightarrow B$ jest funkcją, to nazywa się ją *funkcją odwrotną* funkcji f.

Przykład 3.8

Niech $A =_{def} \{1, 2, 3, 4, 5\}$ oraz $B =_{def} \{1, 2, 3, 4\}$.

Relacja $R \subseteq A \times B$, zdefiniowana jako zbiór par:

$\{<1, 1>, <2, 3>, <1, 4>, <3, 3>, <4, 4>, <2, 4>, <5, 1>\}$

nie jest funkcją.

Funkcja $f: A \rightarrow B$, zdefiniowana jako zbiór par:

$\{<1, 1>, <3, 5>, <4, 3>, <5, 1>\}$

jest funkcją częściową, gdyż $dom(f) = \{1, 3, 4, 5\} \subset A$.

Funkcja $g: A \rightarrow B$, zdefiniowana jako zbiór par:

$\{<1, 1>, <2, 3>, <3, 4>, <4, 3>, <5, 2>\}$

jest funkcją „*na*", gdyż $ran(f) = B$.

Funkcja $h: A \rightarrow A$, zdefiniowana jako zbiór par:

$\{<1, 1>, <2, 3>, <3, 4>, <4, 3>, <5, 1>\}$

nie ma funkcji odwrotnej.

Szczególną formą enumeracyjnej definicji funkcji jest tablica lub krotka. Na przykład wyżej zdefiniowaną funkcję f można przedstawić w postaci tablicy

$f =$

1	2	3	4	5
1	*	5	3	1

lub krotki

$h = <1, *, 5, 3, 1>$,

gdzie symbol * oznacza, że dla danego argumentu funkcja jest niezdefiniowana.

Przedstawienie funkcji w postaci krotki wymaga dodatkowo, aby dziedzina funkcji była zbiorem liniowo uporządkowanym. Tak jest oczywiście w przypadku dziedziny funkcji f, gdzie porządek w zbiorze A jest wyznaczony przez relację $\leq$.

Funkcje mogą być określane na dowolnych zbiorach. Elementami takich zbiorów mogą być złożone twory, na przykład inne funkcje. W takich przypadkach funkcje nazywa się *funkcjonałami*.

Przykład 3.9

Rozpatrzmy zbiór *FUN*, którego elementami są jednoargumentowe funkcje określone na zbiorze liczb rzeczywistych i o wartościach w zbiorze liczb rzeczywistych *Rzeczywiste*. Z definicji jest to zbiór, którego elementami są funkcje typu

Rzeczywiste → *Rzeczywiste*

Rozpatrzmy operator różniczkowania funkcji *Diff*. Jest to funkcja o sygnaturze

Diff: *FUN* → *FUN*

albo, w postaci rozwiniętej, o sygnaturze

Diff: (*Rzeczywiste* → *Rzeczywiste*) → (*Rzeczywiste* → *Rzeczywiste*)

Operator *Diff* jest funkcją częściową, gdyż istnieją funkcje, które nie mają pochodnej w żadnym punkcie. Istnieją też funkcje całkowicie określone, które mają punkty nieciągłości (są nieróżniczkowalne w tych punktach). Dla takich funkcji operator różniczkowania wyznacza funkcje częściowo określone.

W rozważanych wyżej przykładach funkcje były definiowane enumeracyjnie. Często spotykanym sposobem jest definiowanie funkcji przez wyrażenia funkcyjne. Definicja ma postać *równości*, na przykład

$f(x, y, z) = x * y + 10*z$

Jest to *równość*, po lewej stronie której występuje symbol funkcji z listą zmiennych (argumentów), a po prawej stronie występuje *wyrażenie funkcyjne* (*term*).

W przykładowym wyrażeniu funkcje +, * są znanymi dwuargumentowymi funkcjami arytmetycznymi, 10 jest funkcją zeroargumentową, czyli stałą, a x, y są zmiennymi. Wyrażenie funkcyjne jest więc złożeniem pewnych funkcji. Ogólnie jest ono definiowane następująco:

- stała i zmienna są wyrażeniami funkcyjnymi,
- jeżeli h jest n-argumentową funkcją oraz $g_1, ..., g_n$ są wyrażeniami funkcyjnymi, to $h(g_1, ..., g_n)$ jest wyrażeniem funkcyjnym.

Występujący po lewej stronie symbol funkcji nie może wystąpić po prawej stronie równości. Jedynymi zmiennymi, które mogą występować po prawej stronie równości, są tylko te, które występują po lewej stronie.

Funkcje są pewnymi zbiorami i mogą być definiowane rekursywnie oraz przez określenie własności. Definicję rekursywną, czyli algorytmiczną, funkcji nazywa się też definicją *intensjonalną*, a definicję przez określenie własności – definicją *ekstensjonalną*.

Przykład 3.10

Funkcja *Silnia* jest typu $Nat \rightarrow Nat$. Jej definicja rekursywna ma postać:

$$Silnia(0) = 1$$
$$Silnia(n) = n * Silnia(n-1) \qquad \text{dla } n > 0$$

Definicja składa się z dwóch równości. Po lewej stronie równości występuje symbol definiowanej funkcji wraz z odpowiednimi wartościami argumentu, a po prawej stronie występują wyrażenia funkcyjne. Wyrażenie funkcyjne w pierwszej równości jest stałą (funkcją zeroargumentową), a w drugiej – jest złożeniem funkcji trzech funkcji: odejmowania –, mnożenia * oraz definiowanej funkcji *Silnia*. Pierwsza równość definiuje wartość funkcji dla argumentu o wartości 0, druga – definiuje wartość funkcji dla pozostałych wartości argumentu. Zastosowanie drugiej równości do obliczenia wartości funkcji dla argumentu n wymaga uprzedniego obliczenia wartości funkcji dla argumentu $n - 1$.

W podobny sposób jest zdefiniowana rekursywnie funkcja $M : Nat \rightarrow Nat$:

$$M(1) = 2$$
$$M(2) = 2$$
$$M(n) = 2 * M(n-1) + M(n-2) \qquad \text{dla } n > 2$$

Znacznie bardziej złożony jest sposób definicji funkcji Ackermanna o sygnaturze Ack: $Nat \times Nat \rightarrow Nat$

$$Ack(x, y) = y + 1 \qquad \text{gdy } x = 0$$
$$Ack(x, y) = Ack(x - 1, 1) \qquad \text{gdy } y = 0$$
$$Ack(x, y) = Ack(x - 1), Ack(x, y - 1)$$

Przykład 3.11

Niech, jak poprzednio, *FUN* oznacza zbiór, którego elementami są jednoargumentowe funkcje określone na zbiorze liczb rzeczywistych i o wartościach w zbiorze liczb rzeczywistych, czyli funkcje typu *Rzeczywiste* → *Rzeczywiste*. Dla dowolnej funkcji f: *Rzeczywiste* → *Rzeczywiste* rozważa się równanie postaci $f(x) = 0$. Równanie to może nie mieć pierwiastków rzeczywistych, może też mieć ich nieskończenie wiele. Przez *Pierwiastki*(f) oznacza się wartość funkcji, która dla danej funkcji f *wyznacza* podzbiór liczb rzeczywistych P, które są pierwiastkami równania $f(x) = 0$. Funkcja *Pierwiastki* jest typu $FUN \rightarrow 2^{Rzeczywiste}$ Funkcję tę można zdefiniować ekstensjonalnie w sposób następujący:

$$Pierwiastki = \{<f, P> \in FUN \times 2^{Rzeczywiste} \mid x \in P \Leftrightarrow f(x) = 0\}$$

Wprawdzie funkcja *Pierwiastki* jest zdefiniowana jednoznacznie, jednak z definicji tej nie wynika jak dla konkretnej funkcji f określić zbiór jej pierwiastków. Wiadomo, że znajdywanie pierwiastków rzeczywistych równania $f(x) = 0$ jest zadaniem rozwiązywalnym efektywnie tylko dla pewnych klas funkcji.

Przykład 3.12

Rozważmy funkcję *WartośćWielomianu*, która oblicza wartość dowolnego wielomianu dla zadanego argumentu. Jest to funkcja o sygnaturze

$$WartośćWielomianu : Wielomiany \times Rzeczywiste \rightarrow Rzeczywiste$$

Wielomian n-tego stopnia

$$a_n * x^n + ... + a_1 * x + a_0$$

gdzie $n \in Nat$, jest jednoznacznie określony przez zestaw swoich $n + 1$ współczynników a_n, ..., a_1, $a_0 \in Rzeczywiste$. Zbiór *Wielomiany* może zatem być zdefiniowany jako

$$Wielomiany =_{def} \bigcup_{n \in Nat} Rzeczywiste^n$$

stąd, dla dowolnego $<a_n, ..., a_1, a_0> \in Wielomiany$ oraz $x \in Rzeczywiste$

$$WartośćWielomianu(<a_n, ..., a_1, a_0>, x) = a_n * x^n + ... + a_1 * x + a_0$$

Obliczenie tak zdefiniowanej wartości funkcji *WartośćWielomianu* sprowadza się, w oczywisty sposób, do prostego algorytmu obliczeń.

3.8. Operacje na funkcjach

Na funkcjach można wykonywać różne operacje, definiując w ten sposób nowe funkcje. Funkcje są relacjami, w szczególności można wykonywać na nich operacje mno-

gościowe, ale należy zauważyć, że wynikiem takich operacji nie zawsze jest funkcja. Jeżeli na przykład dane są dwie funkcje f, g: $A \to B$, to ich mnogościowa suma $f \cup g$ może nie być funkcją, natomiast przekrój funkcji $f \cap g$ oraz ich różnica $f \setminus g$ są zawsze funkcjami.

Operacja *superpozycji* albo *składania sekwencyjnego* funkcji jest zdefiniowana tak samo jak dla relacji. Jeżeli dane są dwie funkcje:

$$f: A \to B \text{ oraz } g: B \to C$$

to złożeniem sekwencyjnym albo superpozycją funkcji f z funkcją g, oznaczanym przez $f \circ g$, jest funkcja typu $A \to C$, określona następująco:

$$(f \circ g)(a) =_{\text{def}} g(f(a))$$

pod warunkiem, że $f(a)$ oraz $g(f(a))$ są określone.

Inne operacje specyficzne dla funkcji to operacje:

- *warunkowego wyboru*,
- *modyfikacji funkcji przez podstawienie*,
- *obcięcia*.

Niech będą dane dwie funkcje $f, g : A \to B$ oraz trzecia funkcja $h : A \to Logiczne$, gdzie $Logiczne =_{\text{def}} \{prawda, fałsz\}$. *Warunkowym wyborem* funkcji f, g, h, oznaczanym

$$h \to f, g$$

nazywa się funkcję typu $A \to B$, która jest określona następująco:

$$(h \to f,\ g)(a) = \begin{cases} f(a) \text{ gdy } & h(a) = prawda \\ g(a) \text{ gdy } & h(a) = fałsz \end{cases}$$

Przykład 3.13

Niech

$f = \{<1, 2>, <2, 3>, <3, 4>\}$
$g = \{<1, 3>, <2, 3>, <5, 5>\}$
$h = \{<1, prawda>, <2, fałsz>, <3, prawda>, <4, fałsz>, <5, prawda>\}$

wówczas

$h \to f, g = \{\{<1, 2>, <2, 3>, <3, 4>\}$

Niech $f : A \to B$ będzie funkcją oraz niech $a \in A$, $b \in B$. *Modyfikacją funkcji f przez podstawienie* wartości b dla argumentu o wartości a jest funkcja typu $A \to B$, oznaczana symbolicznie $f[a := b]$, zdefiniowana w sposób następujący:

$$f[a := b](x) =_{\text{def}} (x = a) \to b, f(x)$$

W definicji tej wykorzystano poprzednio wprowadzony operator warunkowego wyboru funkcji. Wyrażenie $x = a$ przedstawia funkcję, która przyjmuje wartość logiczną *prawda* wtedy i tylko wtedy, gdy argument x przyjmuje wartość a.

Przykład 3.14

Niech:

$f = \{<1, 2>, <2, 3>, <3, 4>\}$

$g = \{<2, 3>, <5, 5>\}$

wówczas:

$f[1 := 5] = \{<1, 5>, <2, 3>, <3, 4>\}$

$g[1 := 5] = \{<1, 5>, <2, 3>, <5, 5>\}$

Niech $C \subset A$. *Obcięciem* funkcji $f: A \to B$ do podzbioru C zbioru źródłowego A będzie się nazywać funkcję $f|_C : C \to B$, określoną następująco:

dla dowolnego $a \in C : f|_C(a) =_{def} f(a)$.

Łatwo sprawdzić, że równoważną definicją obcięcia funkcji jest

$$f|_C =_{def} f \cap (C \times B)$$

Niech $f: A \to B$ oraz niech $A_1 \subseteq A$, $B_1 \subseteq B$. *Obrazem* zbioru A_1 dla funkcji f nazywa się zbiór

$$f(A_1) =_{def} \{b \in B \mid \exists\, a \in A_1 \bullet f(a) = b\}$$

Przeciwobrazem zbioru B_1 dla funkcji f nazywa się zbiór

$$f^{-1}(B_1) =_{def} \{a \in A \mid \exists\, b \in B_1 \bullet f(a) = b\}$$

Przykład 3.15

Jeżeli

$f = \{<1, 2>, <2, 3>, <3, 4>, <4, 5>\}$

to:

$f|_{\{1, 2\}} = \{<1, 2>, <2, 3>\}$

$f(\{1, 2\}) = \{2, 3\}$

$f^{-1}(\{3, 4\}) = \{2, 3\}$

3.9. Funkcje a relacje

Pomiędzy relacjami a funkcjami zachodzą pewne związki. Każda funkcja jest – z definicji – relacją. Odwrotnie tak nie jest, ale każdej relacji $R \subseteq A \times B$ można przyporządkować przynajmniej jedną taką funkcję $f_R : A \rightarrow B$, że dla każdego $a \in dom(R)$

$$f_R(a) =_{def} b$$

gdzie $b \in B$ jest takim elementem, że $<a, b> \in R$, czyli że

$$f_R \subseteq R \text{ oraz } dom(f_R) = dom(R).$$

Funkcję f_R nazywa się funkcją *zgodną* z relacją R.

Przykład 3.16

Niech

$$R = \{<1, 2>, <1, 3>, <2, 5>, <2, 3>, <3, 4>, <4, 5>\}$$

wówczas:

$$\{<1, 2>, <2, 3>, <3, 4>, <4, 5>\}$$
$$\{<1, 3>, <2, 3>, <3, 4>, <4, 5>\}$$

są funkcjami zgodnymi z R.

Związek zgodności zachodzi pomiędzy programem a jego specyfikacją. Specyfikację programu wyraża się jako pewną relację *Spec*, program zaś jest pewną funkcją *Prog*. Program spełnia specyfikację, gdy pomiędzy specyfikacją *Spec* i programem *Prog* zachodzi związek zgodności, czyli $dom(Prog) = dom(Spec)$ oraz $Prog \subseteq Spec$.

Przykład 3.17

Przypuśćmy, że potrzebna jest funkcja obliczająca pierwiastek kwadratowy z liczby rzeczywistej x, z dokładnością 1. Niech y będzie wartością tej funkcji dla danego x. Związek między x oraz y jest określony zależnością

$$y^2 \leq x \leq (y + 1)^2$$

Formuła ta definiuje relację

$$Spec_{sqrt} =_{def} \{<x, y> \in Rzeczywiste^2 \mid y^2 \leq x \leq (y + 1)^2\}$$

która może być *specyfikacją* programu.

Implementacją dla tej relacji jest dowolna funkcja (realizowana przez program)

$$Impl_{sqrt} : Rzeczywiste \rightarrow Rzeczywiste$$

która spełnia warunki:

$$dom(Impl_{sqrt}) = dom(Spec_{sqrt}) \text{ oraz } Impl_{sqrt} \subseteq Spec_{sqrt}.$$

Czytelnikowi proponuje się samodzielne przedstawienie graficznej ilustracji relacji $Spec_{sqrt}$ oraz funkcji $Impl_{sqrt}$ na płaszczyźnie współrzędnych x, y.

Dowolną relację można przedstawić za pomocą jej funkcji charakterystycznej. Jeżeli dana jest relacja $R \subseteq A_1 \times ... \times A_n$, to jej *funkcją charakterystyczną* jest funkcja

$$f_R : A_1 \times ... \times A_n \rightarrow Logiczne$$

zdefiniowana następująco:

$f_R(a_1, ..., a_n) = prawda$ wtedy i tylko wtedy, gdy $<a_1, ..., a_n> \in R$.

Funkcja charakterystyczna dla danej relacji R jest wyznaczona jednoznacznie. Odwrotnie – dana funkcja charakterystyczna wyznacza jednoznacznie pewną relację.

3.10. Grafy a relacje

Liczne zastosowania w informatyce mają grafy. Rozpatrzmy najpierw klasę *grafów skierowanych*. Mają one dwie równoważne definicje. W zależności od potrzeb wykorzystuje się jedną z nich.

Pierwsza definicja określa graf skierowany G jako parę

$$G = <V, A>$$

gdzie:

V jest zbiorem *wierzchołków* grafu,

A jest zbiorem *łuków* grafu, określonym jako relacja binarna na zbiorze wierzchołków $A \subseteq V \times V$.

Interpretacja relacji A jest następująca: para $<v_1, v_2> \in A$ reprezentuje łuk grafu prowadzący od wierzchołka v_1 do wierzchołka v_2.

Druga definicja określa graf skierowany G jako parę

$$G = <V, S>$$

gdzie:

V jest zbiorem *wierzchołków* grafu,

S jest funkcją, zwaną *funkcją następników*, określoną na zbiorze wierzchołków, której wartościami są podzbiory wierzchołków S: $V \rightarrow 2^V$.

Interpretacja funkcji S jest następująca: $S(v) = \{v_1, ..., v_k\}$ reprezentuje zbiór wierzchołków-następników wierzchołka v, to znaczy wierzchołków, do których prowadzą łuki wychodzące z wierzchołka v. Jej funkcja odwrotna $P(v)$ określa zbiór wierzchołków-poprzedników wierzchołka v, to znaczy wierzchołków, od których prowadzą łuki do wierzchołka v. Znając dla danego grafu funkcję S, łatwo jest wyznaczyć funkcję P.

Łatwo również zauważyć, że graf zdefiniowany według jednej z tych definicji daje się wyrazić w równoważny sposób według drugiej definicji.

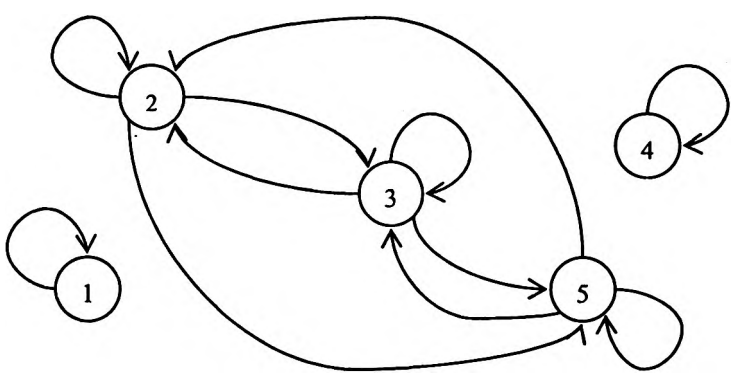
Obie definicje grafów dają podstawę do graficznej ich reprezentacji. Pierwsza definicja pozwala także na graficzną reprezentację relacji binarnych. Pierwszy przykład takiej reprezentacji przedstawiono na rysunku 3.2. Poniżej rozważamy inny przykład, będący graficzną reprezentacją relacji równoważności z przykładu 3.4. W tym przypadku łatwo się przekonać, że analiza niektórych własności relacji, na przykład przechodniości, staje się przejrzysta.

Przykład 3.18

Relacja $R \subseteq A^2$, gdzie $A =_{def} \{1, 2, 3, 4, 5\}$, zdefiniowana następująco:

$R =_{def} \{<1, 1>, <2, 2>, <3, 3>, <4, 4>, <5, 5>, <3, 2>, <2, 3>, <2, 5>, <5, 2>, <3, 5>, <5, 3>\}$

ma postać graficzną przedstawioną na rysunku 3.5.



Rys. 3.5. Graficzna reprezentacja relacji

Podgrafem grafu $G = <V, A>$ nazywa się dowolny graf $G' = <V', A'>$ taki, że $V' \subseteq V$ oraz $A' \subseteq V' \times V'$.

Graf nazywa się *nieskończonym*, gdy nieskończony jest zbiór jego wierzchołków.

Ścieżką w grafie $G = <V, A>$ nazywa się niepusty ciąg łuków

$<v_1, v_2> <v_2, v_3> \ldots <v_{n-1}, v_n> \ldots$

gdzie $<v_i, v_{i+1}> \in A$, dla $i = 1, \ldots, n-1, \ldots$ Ścieżka przechodzi przez wierzchołki:

$v_1, v_2, \ldots, v_n \ldots$

gdzie v_1 jest początkowym wierzchołkiem ścieżki, a jeżeli ścieżka jest skończona, to v_n jest jej wierzchołkiem końcowym. Ścieżkę skończoną, która ma taki sam wierzcho-

łek początkowy i końcowy, nazywa się *cyklem*. Cykl złożony z jednego elementu nazywa się *pętlą*.

Wyróżnia się wiele rodzajów grafów. Określa się między innymi, że graf $G = <V, A>$ jest:

zwrotny, gdy relacja A jest zwrotna (przy każdym wierzchołku jest pętla),

przeciwzwrotny, gdy relacja A jest przeciwzwrotna (graf nie ma pętli),

symetryczny, gdy relacja A jest symetryczna,

przeciwsymetryczny, gdy relacja A jest przeciwsymetryczna,

antysymetryczny, gdy relacja A jest antysymetryczna,

przechodni, gdy relacja A jest przechodnia.

Grafy symetryczne nazywa się także grafami *nieskierowanymi*.

Szczególnym, dalej wykorzystywanym grafem, jest graf nazywany *drzewem*. Jest to graf, który:

- nie ma cykli,
- ma dokładnie jeden wierzchołek v_0, zwany *korzeniem* drzewa, który nie ma poprzedników, to znaczy nie istnieje wierzchołek $v \in V$ taki, że $<v, v_0> \in A$,
- wszystkie pozostałe wierzchołki mają dokładnie jeden poprzednik, to znaczy dla dowolnego wierzchołka $v \neq v_0$ zachodzi $card\{v' \in V \mid <v', v> \in A\} = 1$. Wierzchołek v, który ma następniki, to znaczy $card\{v' \in V \mid <v, v'> \in A\} > 0$, nazywa się wierzchołkiem *rozgałęziającym*. Wierzchołek, który nie ma następników, to znaczy $card\{v' \in V \mid <v, v'> \in A\} = 0$, nazywa się *liściem*, a zbiór wszystkich liści nazywa się *koroną* drzewa.

Lemat 5.1 (*Lemat Königa*)

Jeżeli graf G jest drzewem nieskończonym, w którym każdy wierzchołek ma skończoną liczbę wierzchołków-następników, to w grafie G istnieje ścieżka o nieskończonej długości.

Dowód

Niech v_0 będzie korzeniem grafu G. Zgodnie z założeniem v_0 ma skończoną liczbę wierzchołków-następników. Wśród nich istnieje przynajmniej jeden wierzchołek, niech będzie to v_1, który jest korzeniem nieskończonego poddrzewa G_1 drzewa G, gdyż – w przypadku przeciwnym – gdyby wszystkie wierzchołki-następniki v_0 były korzeniami poddrzew skończonych, to graf G byłby skończony. Powtarzając podobne rozumowanie do następników wierzchołka v_1, znajduje się wśród jego następników wierzchołek v_2, który jest korzeniem nieskończonego podrzewa G_2 drzewa G_1 itd. Jest zatem oczywiste, że ciąg wierzchołków $v_0, v_1, v_2, \ldots$ wyznacza nieskończoną ścieżkę. ■

Ćwiczenia

1. Ile relacji binarnych można zdefiniować na produkcie kartezjańskim $A \times B$, jeżeli A oraz B są zbiorami skończonymi o licznościach $card(A) = n$ oraz $card(B) = m$.

2. Uzupełnij i udowodnij wzory:

   a) $(A \cap B) \times C = (A \times C) \cap (B \times C)$
   b) $(A \cup B) \times C = ?$
   c) $(A \cup B) \times (C \cup D) = ?$

3. Niech $card(A) = n$ oraz $card(B) = m$. Jaka jest liczba funkcji całkowitych oraz częściowych typu $A \rightarrow B$?

4. Niech U będzie pewnym zbiorem uniwersum oraz $\subseteq_U$ niech będzie relacją zawierania pomiędzy podzbiorami zbioru U. Które z własności: symetrię, zwrotność, przechodniość ma relacja $\subseteq_U$?

5. Niech $X =_{def} \{a, b, c, d\}$ oraz $R \subseteq X^2$. Zbadać, które spośród własności: symetrii, przeciwsymetrii, zwrotności, przeciwzwrotności, przechodniości, spójności i równoważności mają następujące relacje binarne:

   a) $R = \{<a, a>, <b, b>, <a, b>\}$
   b) $R = \{<a, a>, <b, b>, <c, c>, <d, d>, <a, b>, <b, a>\}$

6. Pomiędzy ludźmi wyróżnia się różne stosunki: koleżeństwo, znajomość, przyjaźń, wrogość, pokrewieństwo itp. Stosunki te można modelować relacjami binarnymi określonymi na zbiorze ludzi, na przykład: $R_{koleżeństwo} =_{def} \{<a, b> \mid a \text{ jest kolegą } b\}$;

   a) określić, które spośród własności charakteryzujących relacje binarne można przypisać nowo zdefiniowanym relacjom,
   b) jakie związki zachodzą pomiędzy tymi relacjami, chodzi o związki zawierania, na przykład, czy $R_{znajomość} \subseteq R_{koleżeństwo}$, a także o inne, na przykład: czy jeżeli $<a, b> \in R_{wrogość}$ oraz $<b, c> \in R_{wrogość}$, to $<a, c> \in R_{przyjaźń}$?

7. Sprawdzić, czy prawdziwe są następujące stwierdzenia dotyczące relacji binarnych na X:

   a) suma dwóch relacji symetrycznych jest symetryczna,
   b) część wspólna (przekrój) dwu relacji przechodnich jest przechodnia,
   c) jeżeli R jest relacją przechodnią oraz $R \subseteq S \subseteq X^2$, to S jest relacją przechodnią.

8. Sprawdzić, czy prawdziwe są następujące stwierdzenia dotyczące relacji równoważności na X:

   a) suma dwóch relacji równoważności jest relacją równoważności,
   b) przekrój dwóch relacji równoważności jest relacją równoważności.
   c) różnica dwóch relacji równoważności jest relacją równoważności.

9. Niech *ID* będzie zbiorem identyfikatorów. Czy zdefiniowane poniżej relacje binarne $R_1, R_2 \subseteq ID^2$ są relacjami równoważności? Jeżeli tak, to jakie są wyznaczone przez nie zbiory ilorazowe?

   a) $R_1 =_{def} \{<id_1, id_2> \mid$ pierwsza litera identyfikatora id_1 jest taka sama, jak pierwsza litera identyfikatora $id_2\}$,

   b) $R_2 =_{def} \{<id_1, id_2> \mid$ identyfikator id_1 czytany wspak jest taki sam, jak identyfikator $id_2\}$.

10. Niech $BAZA =_{def} Nazwisko \times Wiek \times Zarobek$, gdzie *Nazwisko* jest zbiorem identyfikatorów, *Wiek* i *Zarobek* są pewnymi podzbiorami nieujemnych liczb całkowitych. Czy relacje binarne $R_1, R_2 \subseteq BAZA^2$ są relacjami równoważności? Jeżeli tak, to jakie są wyznaczone przez nie zbiory ilorazowe?

    a) $R_1 =_{def} \{<<n_1, w_1, z_1>, <n_2, w_2, z_2>> \mid w_1 = w_2 \wedge z_1 = z_2\}$,

    b) $R_2 =_{def} \{<<n_1, w_1, z_1>, <n_2, w_2, z_2>> \mid w_1 = w_2 \wedge |z_1 - z_2| < 1000\}$.

11. Ile jest różnych relacji równoważności na zbiorze n-elementowym?

12. Niech $R, S \subseteq X^2$ będą relacjami równoważności. Czy relacjami równoważności są również:

    a) $R \cup S$,
    b) $R \cap S$,
    c) $R \setminus S$,
    d) $R \circ S$.

13. Niech $R \subseteq X^2$ będzie relacją równoważności oraz $x, y \in X$ będą dwoma ustalonymi elementami zbioru X. Czy relacją równoważności jest relacja:

    $S =_{def} (R \cup \{<x, y>, <y, x>\})^+$

14. Jeśli $R_1 \subseteq X^2$, to $R_2 \subseteq X^2$ takie, że $R_1 \subseteq R_2$ nazywamy rozszerzeniem relacji R_1. Czy każdą relację $R \subseteq X^2$ można rozszerzyć do relacji:

    a) symetrycznej,
    b) przeciwsymetrycznej,
    c) zwrotnej,
    d) przeciwzwrotnej,
    e) przechodniej,
    f) spójnej.

15. Wykazać, że relacja R jest przechodnia wtedy i tylko wtedy, gdy spełniony jest warunek

    $R^2 \subseteq R$

16. Niech S, T będą relacjami binarnymi na X^2. Wskaż, które własności są prawdziwe:

    a) $dom(S \cup T) = dom(S) \cup dom(T)$,

b) $dom(S \cup T) \subseteq dom(S) \cup dom(T)$,
c) $dom(S \cap T) \subseteq dom(S) \cap dom(T)$.

17. Pokazać, że złożenie funkcji różnowartościowych jest funkcją różnowartościową.

18. Niech $f: X \to Y$ oraz $A, B \subseteq X$. Uzupełnij i udowodnij wzory:
a) $f(A \cup B) = f(A) \cup f(B)$
b) $f(A \cap B)$? $f(A) \cap f(B)$
c) $f^{-1}(f(A))$? A

19. Funkcja f jest *zgodna* z relacją i wtedy i tylko wtedy, gdy $f \subseteq R$. Niech $X =_{def} \{a, b, c, d\}$ oraz relacja R będzie zdefiniowana następująco:

$R =_{def} \{<a, b>, <a, d>, <c, c>, <b, b>, <b, d>, <c, c>, <d, b>, <c, d>, <d, c>, <d, a>, <d, d>\}$.

Zdefiniować wszystkie funkcje f zgodne z relacją R takie, że:

a) $dom(f) = dom(R)$,
b) $ran(f) = ran(R)$.

Które spośród tych funkcji mają funkcje odwrotne?

20. Podać warunki konieczne i wystarczające na to, aby mnogościowa suma dwóch funkcji była funkcją.

21. Niech R będzie dowolną relacją równoważności na zbiorze X oraz niech relacja Q będzie zdefiniowana następująco:

$Q =_{def} (R \cup \{<a, b>, <b, a>\})^+$,

gdzie $a, b \in X$. Pokazać, że Q jest relacją równoważności oraz jeśli $<a, b> \notin R$, to

$X/Q = (X/R \setminus \{[a]_R, [b]_R\}) \cup \{[a]_R \cup [b]_R\}$.

22. Niech będzie dany pewien graf $G = <V, A>$, gdzie $A \subseteq V^2$. Jaką interpretację można przypisać złożeniu relacji A? Co oznaczają A^2, ..., A^n? W jaki sposób można zbadać, czy graf ma pętle oraz cykle, to jest drogi o długości większej od 1, które rozpoczynają się i kończą się w tym samym wierzchołku?

23. Pokazać, w jaki sposób na podstawie definicji grafu w postaci $G = <V, A>$ zbudować jego definicję o postaci $G = <V, S>$, gdzie $S: V \to 2^V$ jest funkcją wyznaczającą dla dowolnego wierzchołka $v \in V$ zbiór wierzchołków-następników $S(v)$, to znaczy wierzchołków, do których prowadzą łuki z wierzchołka v.

24. Pokazać, w jaki sposób na podstawie definicji grafu w postaci $G = <V, S>$ wyznaczyć funkcję $P(v)$, określającą zbiór wierzchołków-poprzedników wierzchołka v.

25. Zaproponować sposoby reprezentacji grafu w pamięci komputera.

4. Aksjomatyczna i alternatywne teorie zbiorów

4.1. Aksjomatyczne ujęcie teorii mnogości

Używane dotychczas pojęcie zbioru jest rozumiane na jeden z dwóch sposobów, które spotyka się w rozumieniu potocznym. Jest to tak zwane *dystrybutywne* rozumienie zbioru, czyli zespołu (zestawu) wielu przedmiotów (bytów) połączonych w całość ze względu na pewne wspólne własności. Dystrybutywne rozumienie zbioru wiąże się ze stosunkiem pomiędzy elementem a zbiorem: *element należy do zbioru*. Synonimami tak rozumianego zbioru są spotykane w języku naturalnym takie określenia, jak: 'klasa', 'kolekcja', 'wielość', 'mnogość', 'zbiorowość', 'gatunek', 'rodzaj'.

Drugie, nieużywane tu, pojęcie zbioru, tak zwane *kolektywne*, jest rozumiane jako całość złożona z pewnych przedmiotów, które stanowią części całości. Na przykład las może być traktowany jako całość, której częściami są drzewa, krzewy, runo leśne itp. Kolektywne rozumienie zbioru stosunek pomiędzy elementem a zbiorem określa, że: *element jest częścią zbioru*, dlatego synonimami zbioru w sensie kolektywnym są na przykład określenia: 'całość', 'agregat', 'kompleks', 'konglomerat'.

Można twierdzić, że samochód jest zbiorem w sensie kolektywnym, którego częściami są podwozie, koła, silnik, nadwozie itd. Części te są komponentami zbioru i jednocześnie są również zbiorami w sensie kolektywnym, gdyż składają się z innych, mniejszych komponentów. Zbiory kolektywne są zbiorami tranzytywnymi, ponieważ stosunek 'bycia częścią" jest relacją przechodnią.

Zbiory w sensie kolektywnym są wygodne do przedstawiania wielu pojęć w naukach przyrodniczych, społecznych, a także w lingwistyce komputerowej. Rozważania w książce odnoszą się wyłącznie do zbiorów w sensie dystrybutywnym.

Oprócz rozróżnienia zbiorów w sensie kolektywnym i dystrybutywnym, spotyka się jeszcze inne podejścia do pojęcia zbioru, określane jako podejścia alternatywne. Wśród takich podejść w dalszej części rozdziału omawia się bardzo ogólnie wielozbiory, zbiory rozmyte i zbiory przybliżone, które mają liczne zastosowania w informatyce.

W początkowym okresie swego rozwoju teoria mnogości (teoria zbiorów w sensie dystrybutywnym) była budowana na podstawie intuicyjnego pojęcia zbioru. Droga ta okazała się zawodna, gdyż intuicja nie dawała jednoznacznych odpowiedzi na pewne

subtelne pytania. W konsekwencji pojawiły się sprzeczności, jak na przykład omówiona wcześniej antynomia Russella. W celu ich eliminacji zbudowano różne aksjomatyczne teorie zbiorów. Poniżej przedstawiamy zestawy aksjomatów opracowane przez Zermela[7]. Zestaw ten jest wystarczający do praktyki matematycznej, zwłaszcza do definiowania liczb naturalnych, całkowitych, wymiernych i rzeczywistych ze zwykłymi działaniami arytmetycznymi. Bardziej rozpowszechniona jest nieco silniejsza teoria, zwana teorią Zermela–Fraenkla[8]. Aksjomaty Zermela są tu przedstawiane za pomocą języka naturalnego.

1. *Aksjomat ekstensjonalności*

   Dwa zbiory są równe wtedy i tylko wtedy, gdy mają te same elementy.

2. *Aksjomat wyróżniania*

   Dla dowolnego zbioru Z i dowolnego jednoargumentowego predykatu (funkcji zdaniowej) P istnieje zbiór T zawierający dokładnie te elementy Z, które spełniają warunek $P(x)$.
   Jeżeli żaden element Z nie spełnia predykatu P, na przykład, gdy $P(x)$ jest warunkiem postaci $x \notin Z$, to T jest zbiorem pustym $\emptyset$. Aksjomat wyróżniania zapewnia więc istnienie zbioru pustego $\emptyset$.

3. *Aksjomat par nieuporządkowanych*

   Jeżeli Z_1, Z_2 są zbiorami, to para nieuporządkowana $\{Z_1, Z_2\}$ jest zbiorem.

4. *Aksjomat sumy zbiorów*

   Niech Z będzie niepustą rodziną zbiorów, tj. zbiorem, którego elementy są zbiorami. Dla każdej takiej rodziny istnieje zbiór S, którego elementami są dokładnie te obiekty, które są elementami zbiorów należących do Z.

5. *Aksjomat nieskończoności*

   Istnieje zbiór Z, który zawiera zbiór pusty i jest taki, że jeżeli x należy do Z, to suma x oraz $\{x\}$ także jest w Z.
   Rozróżnienie między elementem x a zbiorem jednoelementowym $\{x\}$ ma zasadnicze znaczenie. Aksjomat gwarantuje istnienie zbiorów nieskończonych.

6. *Aksjomat zastępowania*

   Niech dla każdego x istnieje dokładnie jedno y takie, że spełniony jest dwuargumentowy predykat (funkcja zdaniowa) $P(x, y)$, wtedy dla każdego zbioru Z istnieje zbiór Z', do którego należą wszystkie i tylko te elementy y, które przy pewnym x ze zbioru Z, spełniają predykat P.

---

[7] Ernst Zermelo (1871–1953).

[8] Abraham Fraenkel (1891–1965).

7. *Aksjomat zbioru potęgowego*

Dla każdego zbioru Z istnieje rodzina zbiorów, której elementami są wszystkie podzbiory zbioru Z. Rodzinę tę nazywa się zbiorem potęgowym i oznacza 2^Z.

8. *Aksjomat wyboru*

Dla dowolnej rodziny niepustych i rozłącznych zbiorów istnieje zbiór, który z każdym ze zbiorów tej rodziny ma jeden i tylko jeden wspólny element.
Aksjomat wyboru jest z jednej strony intuicyjnie oczywisty, ale z drugiej strony budzi różne kontrowersje. Ich zasadniczym powodem jest to, że w przypadku nieprzeliczalnej rodziny zbiorów nie wiadomo, w jaki sposób tworzyć nowy zbiór, który miałby dokładnie jeden element wspólny z każdym zbiorem tej rodziny. Z całą pewnością proces tworzenia takiego zbioru nie mógłby być postępowaniem efektywnym, to znaczy opartym na realizacji pewnego algorytmu.

9. *Aksjomat regularności (ufundowania)*

W każdym niepustym zbiorze Z istnieje taki element X, że żaden element zbioru X nie jest elementem zbioru Z.
Konsekwencją aksjomatu jest to, że nie istnieją zbiory X, Y, Z o takich własnościach, jak na przykład, że $X \in X$, że zachodzi $X \in Y$ oraz $Y \in X$, że zachodzi $X \in Y$, $Y \in Z$, $Z \in X$ itd. Aksjomat ten ogranicza dziedzinę złożoną ze zbiorów przez wyeliminowanie z niej obiektów o własnościach w rodzaju wyżej wymienionych.

4.2. Definicje zbiorów liczbowych

Na gruncie aksjomatycznego ujęcia teorii mnogości można formalnie zdefiniować liczby naturalne. Ponieważ jedynymi obiektami, których istnienie gwarantuje teoria mnogości, są zbiory, więc liczby naturalne także definiuje się jako szczególne rodzaje zbiorów.

Dla dowolnego zbioru Z jego *następnikiem* nazwa się zbiór

$$Succ(Z) =_{def} Z \cup \{Z\}$$

Zachodzi więc $Z \subseteq Succ(Z)$ oraz $Z \in Succ(Z)$.

Punktem wyjścia w konstrukcji zbioru liczb naturalnych jest przyjęcie istnienia zbioru pustego. Zbiór liczb naturalnych definiuje się jako najmniejszy zbiór *Nat*, definiowany rekursywnie w sposób następujący:

1. $\emptyset \in Nat$
2. jeżeli $Z \in Nat$, to $Succ(Z) \in Nat$

Elementy zbioru *Nat* nazywa się liczbami naturalnymi i są nimi:

$\emptyset$,

$\emptyset \cup \{\emptyset\} = \{\emptyset\}$,

$\{\emptyset\} \cup \{\{\emptyset\}\} = \{\emptyset, \{\emptyset\}\}$,

$\{\emptyset, \{\emptyset\}\} \cup \{\{\emptyset, \{\emptyset\}\}\} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$ itd.

Zbiór liczb naturalnych jest więc rodziną zbiorów. W celu uproszczenia notacji elementów tej rodziny wprowadza się powszechnie znane oznaczenia:

$0 =_{def} \emptyset$,

$1 =_{def} \{\emptyset\} = \{0\}$

$2 =_{def} \{\emptyset, \{\emptyset\}\} = \{0, 1\}$

$3 =_{def} \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} = \{0, 1, 2\}$

.....

$n =_{def} \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, ..., \{\emptyset, \{\emptyset\}, ..., \{...\{\emptyset\}...\} ...\} = \{0, 1, 2, ..., n-1\}$

które są znacznie wygodniejsze w użyciu.

Uwaga

Zbiór liczb naturalnych jest przykładem tak zwanej induktywnej rodziny zbiorów. Rodzina zbiorów A jest *induktywna*, gdy spełnia warunki:

1. $\emptyset \in A$,
2. dla każdego zbioru $X \in A$, również zbiór $X \cup \{X\} \in A$.

Operacja, która zbiorowi X przyporządkowuje $X \cup \{X\}$, nazywa się operacją *następnika*. Istnienie zbiorów induktywnych jest postulowane aksjomatem nieskończoności. Zbiór liczb naturalnych *Nat* jest najmniejszym zbiorem induktywnym, to znaczy dla każdego zbioru induktywnego A zachodzi $Nat \subseteq A$.

Możliwe są także inne mnogościowe sposoby definiowania liczb naturalnych, na przykład:

$0 =_{def} \emptyset$,

$1 =_{def} \{\emptyset\} = \{0\}$

$2 =_{def} \{\{\emptyset\}\} = \{1\}$

$3 =_{def} \{\{\{\emptyset\}\}\} = \{2\}$

.....

$n =_{def} \{\{...\{\emptyset\}...\}\} = \{n-1\}$

Traktując liczby naturalne jako induktywnie zdefiniowane zbiory, można pokazać następujące twierdzenie.

Twierdzenie 4.1

Dla dowolnych liczb $m, n \in Nat$ zachodzą związki:

1. Jeśli $m \in n$, to $m \subseteq n$.
2. $n \notin n$.
3. Jeśli $Succ(m) = Succ(n)$, to $m = n$.
4. Jeśli $m \subseteq n$ oraz $m \neq n$, to $m \in n$.
5. Zachodzi $m \subseteq n$ lub $n \subseteq m$.
6. Zachodzi dokładnie jedna z możliwości: $m \in n$, $n = m$, $m \in n$.

Dowód twierdzenia można znaleźć na przykład w książce [Tiuryn 2003].

Przy wprowadzonych oznaczeniach operację tworzenia nowego zbioru *Succ* można traktować też jako funkcję dodawania jedynki do danej liczby naturalnej. Jest to funkcja całkowicie określona o sygnaturze $Succ : Nat \rightarrow Nat$. Nazywa się ją *operacją następnika* i można pisać:

$Succ(0) = 1$
$Succ(Succ(0)) = Succ(1) = 2$
$Succ(Succ(Succ(0))) = Succ(Succ(1)) = Succ(2) = 3$

Operacja następnika pozwala na zdefiniowanie innych operacji (działań). Na przykład *dodawanie* oraz mnożenie są funkcjami o sygnaturze:

$\_+\_ : Nat \times Nat \rightarrow Nat$
$\_*\_ : Nat \times Nat \rightarrow Nat$

Dodawanie można zdefiniować rekursywnie:

$m + 0 = m$ dla dowolnego $m \in Nat$
$m + Succ(n) = Succ(m + n)$ dla dowolnych $m, n \in Nat$

Dysponując dodawaniem również rekursywnie można określić *mnożenie*:

$m * 0 = 0$ dla dowolnego $m \in Nat$
$Succ(m) * n = m * n + n$ dla dowolnych $m, n \in Nat$

Mając liczby naturalne, można zdefiniować inne rodzaje liczb: liczby całkowite, wymierne, rzeczywiste i zespolone.

Definicję liczb całkowitych poprzedza się pewnym wyjaśnieniem intuicyjnym. Każdej liczbie całkowitej przypisuje się parę liczb naturalnych $<m, n>$ takich, że różnica $m - n$ jest równa tej liczbie całkowitej. Na przykład liczbie całkowitej –2 może być przyporządkowana para $<4, 6>$, liczbie 0 – para $<10, 10>$, a liczbie 3 – para $<4, 1>$. Dwie pary $<m_1, n_1>$ oraz $<m_2, n_2>$, które spełniają warunek

$m_1 - n_1 = m_2 - n_2$

reprezentują tę samą liczbę całkowitą. Ponieważ różnica dwóch liczb naturalnych nie zawsze jest liczbą naturalną, dlatego zamiast takiego warunku można sformułować inny warunek równoważny, w którym nie odwołuje się do różnicy. Jest to warunek postaci

$$m_1 + n_2 = m_2 + n_1$$

Przyjmuje się teraz następującą definicję relacji binarnej $R \subseteq Nat^2 \times Nat^2$, określonej na parach liczb naturalnych w sposób następujący:

$$R =_{def} \{<<m_1, n_2>, <m_1, n_2>> \mid m_1 + n_2 = m_2 + n_1\}$$

Łatwo sprawdzić, że R jest relacją równoważności na Nat^2. Zbiór liczb całkowitych jest określony jako zbiór ilorazowy Nat^2/R, czyli

$$Całkowite = Nat^2/R$$

Konstrukcja liczb wymiernych opiera się na założeniu, że każdej liczbie wymiernej można przyporządkować parę $<l, m>$, gdzie $l \in Całkowite$ oraz $m \in Nat\backslash\{0\}$. Dalsza część konstrukcji jest podobna do konstrukcji zbioru liczb całkowitych. Definiuje się mianowicie relację $Q \subseteq (Całkowite \times Nat)^2$ w sposób następujący:

$$Q =_{def} \{<<l_1, m_2>, <l_1, m_2>> \mid l_1 * m_2 = l_2 * m_1\}$$

Q jest relacją równoważności na $Całkowite \times Nat$. Zbiór liczb wymiernych jest określony jako zbiór ilorazowy $(Całkowite \times Nat)/Q$, czyli

$$Wymierne = (Całkowite \times Nat)/Q$$

Definicja zbioru liczb rzeczywistych jest bardziej złożona i dlatego jest tu pomijana.

4.3. Wielozbiory

Uogólnieniem pojęcia zbioru jest pojęcie wielozbioru. Zbiór jest określony jako kolekcja dobrze wyróżnionych obiektów – elementów zbioru. Czasem nie ma potrzeby jednoznacznego rozróżniania pomiędzy elementami zbioru. Tak jest wtedy, gdy elementami zbioru jest wiele kopii tego samego rodzaju obiektów. Jeżeli na przykład rozważa się zbiór, którego elementami są różne owoce – jabłka, gruszki, śliwki, to może nas interesować tylko liczba poszczególnych rodzajów owoców, bez rozróżniania konkretnych owoców.

Definicja 4.1

Jeżeli A jest dowolnym zbiorem, to *wielozbiorem* (albo *multizbiorem*) W nad zbiorem A jest para

$$W =_{def} <A, f>$$

gdzie f jest *funkcją liczności* wielozbioru. Funkcja f jest dowolną, całkowicie określoną na A, o wartościach w zbiorze liczb naturalnych, czyli jest funkcją o sygnaturze $f: A \rightarrow Nat$ oraz dziedzinie $dom(f) = A$.

Jeżeli $a \in A$, to $f(a)$ jest liczbą elementów a w danym wielozbiorze W. Wielozbiór W jest pusty, gdy $f(a) = 0$ dla każdego $a \in A$.

Niech będą dane dwa wielozbiory nad zbiorem A:

$W_1 = <A, f>$ oraz $W_2 = <A, g>$

W_1 jest podwielozbiorem W_2, co oznacza się $W_1 \subseteq W_2$, jeżeli $f(a) \leq g(a)$, dla każdego $a \in A$.

Wielozbiory W_1 oraz W_2 są identyczne, co pisze się $W_1 = W_2$, wtedy i tylko wtedy, gdy $W_1 \subseteq W_2$ oraz $W_2 \subseteq W_1$.

Na wielozbiorach definiuje się operacje mnogościowe sumy, przekroju i różnicy.

Suma wielozbiorów jest zdefiniowana następująco:

$W_1 \cup W_2 = <A, h>$

gdzie h jest funkcją liczności, spełniającą warunek

$h(a) = f(a) + g(a)$ dla dowolnego $a \in A$

Przekrój wielozbiorów jest zdefiniowany następująco:

$W_1 \cap W_2 = <A, h>$

gdzie h jest funkcją liczności, spełniającą warunek

$h(a) = min(f(a), g(a))$ dla dowolnego $a \in A$

Różnica wielozbiorów jest zdefiniowana następująco:

$W_1 \backslash W_2 = <A, h>$

gdzie h jest funkcją liczności, spełniającą warunek

$h(a) = max(f(a) - g(a), 0)$ dla dowolnego $a \in A$

min oraz *max* są funkcjami, które wyliczają odpowiednio większą oraz mniejszą liczbę spośród dwóch liczb, które są jej argumentami.

Przykład 4.1

Niech $W_1 = <A, f_1>$ oraz $W_2 = <A, f_2>$, gdzie $A = \{a, b, c, d, e\}$ oraz funkcje liczności są zdefiniowane następująco:

$f_1 = \{<a, 4>, <b, 3>, <c, 2>, <d, 1>, <e, 0>\}$
$f_2 = \{<a, 0>, <b, 1>, <c, 2>, <d, 3>, <e, 4>\}$

wówczas

$W_1 \cup W_2 = <A, \{<a, 4>, <b, 4>, <c, 4>, <d, 4>, <e, 4>\}>$

$W_1 \cap W_2 = <A, \{<a, 0>, <b, 1>, <c, 2>, <d, 1>, <e, 0>\}>$

$W_1 \setminus W_2 = <A, \{<a, 4>, <b, 2>, <c, 0>, <d, 0>, <e, 0>\}>$

Uwaga

Często, gdy rozważa się rodzinę wielozbiorów $W_i = <A, f_i>$, dla $i \in I$, nad ustalonym zbiorem A, przyjmuje się uproszczoną notację – wielozbiór W_i utożsamia się z funkcją liczności f_i. Wtedy, zamiast pisać na przykład $W_1 \cup W_2$, pisze się $f_1 \cup f_2$.

4.4. Zbiory rozmyte

Zbiory rozmyte są uogólnieniem zbiorów, którym można się posługiwać w sytuacjach określonych nieprecyzyjnie lub niejednoznacznie. Takie sytuacje występują na przykład wtedy, gdy mówi się *wysoki mężczyzna*, *duże miasto* lub *drogi samochód*. Gdy mówi się o kimś, że jest wysokim mężczyzną, wyraża się przekonanie o stopniu przynależności danego mężczyzny do zbioru wysokich mężczyzn.

Definicja 4.2

Jeżeli A jest dowolnym zbiorem, to *zbiorem rozmytym* Z nad zbiorem A jest para

$$Z =_{def} <A, \mu>$$

gdzie μ jest *funkcją przynależności* do zbioru rozmytego. Funkcja μ jest dowolną funkcją całkowicie określoną na A, o wartościach w zbiorze liczb rzeczywistych z przedziału [0, 1], czyli funkcją o sygnaturze $\mu : A \rightarrow [0, 1]$ oraz $dom(f) = A$.

Jeżeli $a \in A$, to $\mu(a)$ określa stopień przynależności elementu a do danego zbioru rozmytego. Dla $a \in A$ wartość funkcji $\mu(a) = 0$ oznacza brak przynależności, $\mu(a) = 1$ oznacza pełną przynależność, $0 < \mu(a) < 1$ oznacza zaś częściową przynależność elementu a do zbioru rozmytego.

Zbiór rozmyty Z jest pusty, gdy $\mu(a) = 0$ dla każdego $a \in A$.

Przykład 4.2

Przyjmując, że za wysokich mężczyzn można uważać tych, którzy mają co najmniej 170 cm wzrostu, rozmyty zbiór wysokich mężczyzn *WysocyMężczyźni* można zdefiniować następująco:

$$WysocyMężczyźni = <Wzrost, \mu_{wzrost}>$$

gdzie

$$Wzrost = \{x \in Rzeczywiste \mid x \geq 100\}$$

$$\mu_{wysoki}(x) = \begin{cases} 0 & \text{dla } x < 170 \\ \frac{1}{15} * (x - 170) & \text{dla } 170 \leq x \leq 185 \\ 1 & \text{dla } x > 185 \end{cases}$$

Niech będą dane dwa zbiory rozmyte $Z_i = <A, \mu_i>$ dla $i = 1, 2$.

Z_1 jest podzbiorem Z_2, co oznacza się $Z_1 \subseteq Z_2$, jeżeli $\mu_1(a) \leq \mu_2(a)$, dla każdego $a \in A$.

Zbiory rozmyte Z_1 oraz Z_2 są identyczne, co pisze się $Z_1 = Z_2$, wtedy i tylko wtedy, gdy $W_1 \subseteq W_2$ oraz $W_2 \subseteq W_1$.

Operacje mnogościowe na zbiorach rozmytych są definiowane następująco:

Suma zbiorów rozmytych

$$Z_1 \cup Z_2 = <A, \mu>$$

gdzie μ jest funkcją przynależności, spełniającą warunek

$$\mu(a) = max(\mu_1(a), \mu_2(a)) \qquad \text{dla dowolnego } a \in A$$

Przekrój zbiorów rozmytych

$$Z_1 \cap Z_2 = <A, \mu>$$

gdzie μ jest funkcją przynależności, spełniającą warunek:

$$\mu(a) = min(\mu_1(a), \mu_2(a)) \qquad \text{dla dowolnego } a \in A$$

Różnica zbiorów rozmytych

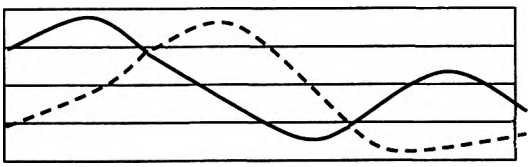
$$Z_1 \setminus Z_2 = <A, \mu>$$

gdzie μ jest funkcją przynależności, spełniającą warunek:

$$\mu(a) = max(\mu_1(a) - \mu_2(a), 0) \qquad \text{dla dowolnego } a \in A$$

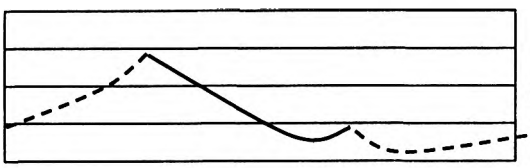
Przykład 4.3

Zbiory rozmyte można przedstawić graficznie za pomocą odpowiadających im wykresów funkcji przynależności. Na pierwszym z rysunków przedstawiono wykresy funkcji przynależności dwóch zbiorów rozmytych nad zbiorem liczb rzeczywistych – linia ciągła (zbiór pierwszy) i przerywana (zbiór drugi), a na następnych – wykresy funkcji przynależności ich sumy i przekroju.



Funkcje przynależności dwóch zbiorów rozmytych – linia ciągła i linia przerywana

Funkcja przynależności sumy dwóch zbiorów rozmytych



Funkcja przynależności przekroju dwóch zbiorów rozmytych

Rys. 4.1. Graficzna ilustracja operacji na zbiorach rozmytych

Uwaga

Badania nad zbiorami rozmytymi zainicjował swoimi pracami Lofti Zadeh w połowie lat sześćdziesiątych ubiegłego wieku. Podane wyżej definicje zawierania i równości zbiorów rozmytych oraz operacje mnogościowe są tymi, które spotyka się najczęściej. W literaturze istnieje rozmaitość innych definicji tych pojęć: [Kacprzyk 1986], [Rutkowska, Piliński, Rutkowski 1997].

Z porównania podanych określeń zbiorów rozmytych z wcześniejszymi określeniami dla wielozbiorów wynika, że z obliczeniowego punktu widzenia różnice są mało istotne. Istotne są natomiast różnice interpretacyjne, gdyż funkcja liczności dla danego elementu $a \in A$ określa liczbę kopii samego elementu w wielozbiorze, podczas gdy funkcja przynależności określa stopień przynależności tego elementu do zbioru rozmytego.

4.5. Zbiory przybliżone

Pojęcie zbiorów przybliżonych wywodzi się z zagadnienia klasyfikacji.

Niech U będzie dowolnym zbiorem uniwersum oraz niech $R \subseteq U^2$ będzie pewną relacją równoważności określoną na U. Relacja równoważności wyznacza podział zbioru U na klasy abstrakcji. Przypomnijmy: dwie klasy abstrakcji są identyczne albo rozłączne, a suma mnogościowa wszystkich klas abstrakcji jest równa zbiorowi U. Zbiór wszystkich klas abstrakcji, oznaczany U/R, jest nazywany zbiorem ilorazowym zbioru U.

Niech będzie dany pewien podzbiór $X \subseteq U$. Podzbiór X jest oczywiście określony przez swoje elementy, ale można go też scharakteryzować tylko poprzez elementy zbioru ilorazowego U/R. Charakteryzacja polega na wprowadzeniu dwóch podzbiorów stanowiących dolne i górne przybliżenie zbioru X.

Dolnym przybliżeniem zbioru X względem relacji R, oznaczanym $\underline{R}X$, jest zbiór zdefiniowany następująco:

$$\underline{R}X = \bigcup \{Y \in U/R \mid Y \subseteq X\}$$

Górnym przybliżeniem zbioru X względem relacji R, oznaczanym $\overline{R}X$, jest zbiór zdefiniowany następująco:

$$\overline{R}X = \bigcup \{Y \in U/R \mid Y \cap X \neq \emptyset\}$$

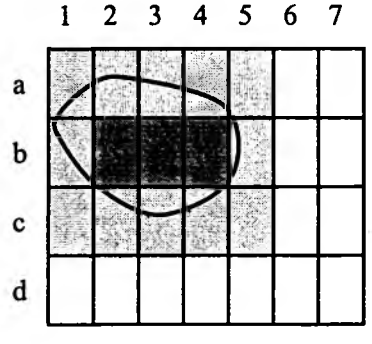
Z definicji wynika, że $\underline{R}X \subseteq X \subseteq \overline{R}X$.

Definicja 4.3

Zbiór X nazywa się *zbiorem przybliżonym* względem relacji R, gdy $\underline{R}X \neq \overline{R}X$. W przeciwnym przypadku, gdy $\underline{R}X = \overline{R}X$, zbiór X nazywa się zbiorem dokładnym względem R.

Przykład 4.4

Ilustracją wprowadzonych pojęć jest rysunek 4.2.



Rys. 4.2. Graficzna ilustracja zbioru przybliżonego

Zbiorem U jest prostokątny obszar na płaszczyźnie X–Y, podzielony na mniejsze prostokąty – kratki są jednoznacznie identyfikowane przez numery wierszy i kolumn. Kratki te są elementami pewnego zbioru ilorazowego dzielącego zbiór U. Zbiór X jest zaznaczony pogrubioną linią. Jego dolnym przybliżeniem $\underline{R}X$ jest ob-

szar zaznaczony trzema mocniej zacieniowanymi kratkami, a górnym jego przybliżeniem $\overline{R}X$ jest obszar zaznaczony wszystkimi zacieniowanymi kratkami.

$\underline{R}X = \{<b, 2>, <b, 3>, <b, 4>\}$

$\overline{R}X = \{<a, 1>, <a, 2>, <a, 3>, <a, 4>, <a, 5>,$
$<b, 1>, <b, 2>, <b, 3>, <b, 4> <b, 5>,$
$<c, 1>, <c, 2>, <c, 3>, <c, 4> <c, 5>\}$

W praktycznych zastosowaniach liczności elementów zbiorów $\underline{R}X$ oraz $\overline{R}X$ dają podstawę do liczbowej oceny dokładności przybliżenia zbioru X. Jeżeli zbiory $\underline{R}X$ oraz $\overline{R}X$ są skończone, to tak zwana miara dokładności przybliżenia zbioru X jest określana jako

$$\alpha_R(X) = card(\underline{R}X) / card(\overline{R}X)$$

Oczywiście $0 \leq \alpha_R(X) \leq 1$.

Uwaga

Zbiory przybliżone zostały wprowadzone przez Zdzisława Pawlaka (1926–2006) w połowie lat osiemdziesiątych ubiegłego wieku. Znalazły one powszechne zastosowanie w informatyce, między innymi w analizie danych, przybliżonej klasyfikacji i przetwarzaniu obrazów [Pawlak 1991].

Ćwiczenia

1. Niech X, Y, Z będą wielozbiorami. Pokazać, że jeżeli $X \subseteq Y$ oraz $Y \subseteq Z$, to $X \subseteq Z$.
2. Niech $W = <U, F>$ będzie wielozbiorem nad zbiorem U, gdzie funkcja liczności F jest zdefiniowana następująco: $F(x) = n$ dla każdego $x \in U$. Jeżeli A jest podwielozbiorem wielozbioru W, to przez A' oznaczamy operację dopełnienia wielozbioru A, którą definiujemy jako: $A' =_{def} W \setminus A$. Sprawdzić, czy zachodzą prawa de Morgana:

   $(A \cap B)' = A' \cup B'$

   $(A \cup B)' = A' \cap B'$
3. Niech X, Y, Z będą zbiorami rozmytymi. Pokazać, że jeżeli $X \subseteq Y$ oraz $Y \subseteq Z$, to $X \subseteq Z$.
4. Niech $Z = <U, \mu>$ będzie zbiorem rozmytym, gdzie funkcja przynależności μ jest zdefiniowana następująco: $\mu(x) = 1$ dla każdego $x \in U$. Jeżeli A jest podzbiorem rozmytym zbioru Z, to przez A' oznaczamy operację dopełnienia zbioru A, którą definiujemy jako: $A' =_{def} Z \setminus A$. Sprawdzić, czy zachodzą prawa de Morgana:

$(A \cap B)' = A' \cup B'$

$(A \cup B)' = A' \cap B'$

5. Niech dany będzie zbiór *Osoba* = *Nazwisko* ×*Wiek* × *Zarobek*, gdzie *Nazwisko* jest zbiorem nazwisk, *Wiek* = *Nat*, *Zarobek* = *Nat*, oraz niech będą dane dwie relacje równoważności: relacja *W* na zbiorze *Wiek* i relacja *Z* na zbiorze *Zarobek*. Uzasadnić, że rodzina zbiorów $\{b_{w,z} \subseteq Osoba \mid w \in Wiek, z \in Zarobek\}$, zdefiniowanych następująco: $b_{w,z} =_{def} \{<o_1, w_1, z_1> \in Osoba \mid w_1 \in [w]_{Wiek} \wedge z_1 \in [z]_{Zarobek}\}$, dla $w \in Wiek$ oraz $z \in Zarobek$, stanowi partycję na zbiorze *Osoba*, czyli wyznacza pewną relację równoważności $R_{W,Z}$ na tym zbiorze. Podać przykłady relacji $W \subseteq Wiek$ i $Z \subseteq Zarobek$. Podać dolne i górne ograniczenia dla przykładowego podzbioru $B \subseteq Osoba$ względem relacji $R_{W,Z}$.

5. Równoliczność zbiorów, liczby kardynalne

5.1. Zbiory przeliczalne

Pojęcie funkcji pozwala na porównywanie liczności zbiorów.

Definicja 5.1

Dwa zbiory A i B są *równoliczne*, co notujemy w postaci $A \sim B$, wtedy i tylko wtedy, gdy istnieje wzajemnie jednoznaczna funkcja (bijekcja)

$$f: A \to B$$

O zbiorach równolicznych mówi się też, że są zbiorami o tej samej *mocy*.

Łatwo zauważyć, że związek równoliczności ma oczywiste własności:

- $A \sim A$,
- jeżeli $A \sim B$, to $B \sim A$,
- jeżeli $A \sim B$ i $B \sim C$, to $A \sim C$.

Uwaga

Związek równoliczności ma własności relacji równoważności. Nie mówi się jednak o relacji równoliczności, gdyż wymagałoby to określenia zbioru, na którym relacja ta jest określona. W tym przypadku dziedziną tą powinien być zbiór, którego elementami są wszystkie możliwe zbiory. Określenie tego, czym jest zbiór wszystkich zbiorów, stwarza jednak problemy interpretacyjne. Wprowadzenie pojęcia zbioru wszystkich zbiorów nasuwa pytanie: skoro zbiór wszystkich zbiorów jest zbiorem, to czy nie powinien być swoim elementem? Próba odpowiedzi na takie pytanie prowadzi do paradoksu Russella. Pojęcie zbioru wszystkich zbiorów jest zatem wewnętrznie sprzeczne.

Dla zbiorów nieskończonych zachodzi charakterystyczna własność, polegająca na tym, że cały zbiór jest równoliczny z pewnym swoim podzbiorem właściwym. Ta własność jest podstawą formalnej definicji zbiorów nieskończonych. Zbiór jest *nieskończony* wtedy i tylko wtedy, gdy ma podzbiór właściwy, który jest z nim równoliczny.

Skończoność zbioru A oznacza, że istnieje $n \in Nat$ takie, że $|A| \sim n$.

Zapis $A \sim n$ wymaga komentarza. Należy przypomnieć, że liczby naturalne zdefiniowano w podrozdziale 4.2. jako najmniejszy zbiór induktywny. Symbol n jest zatem nazwą zbioru reprezentującego liczbę naturalną, co wyjaśnia sensowność zapisu $A \sim n$. W twierdzeniu 5.1 symbol liczby naturalnej n występuje w roli zbioru.

Twierdzenie 5.1

1. Dla dowolnej liczby $n \in Nat$ nie istnieje funkcja różnowartościowa z $n \cup \{n\}$ w n.
2. Dla $n, m \in Nat$, jeśli $m \sim n$, to $m = n$.
3. Żadna liczba naturalna nie jest równoliczna z *Nat*, a zatem *Nat* jest nieskończony.

Dowód twierdzenia można znaleźć na przykład w książce [Tiuryn 2003].

W tym przypadku mówimy, że A ma n elementów i oznaczamy to, pisząc *card*(A). Takie oznaczenie było wprowadzone już wcześniej w rozdziale 2.

Przykład 5.1

a) Zbiór liczb parzystych *Parzyste* jest równoliczny ze zbiorem liczb naturalnych *Nat*. Wystarczy zauważyć, że funkcja f: *Nat* $\rightarrow$ *Parzyste*, która wzajemnie jednoznacznie odwzorowuje zbiór *Nat* w zbiór *Parzyste*, jest zdefiniowana wzorem

$$f(n) = 2*n$$

w którym: $n \in Nat$, a $2*n \in Parzyste$.

b) Podobnie równoliczne są zbiory *Parzyste* i *Nieparzyste*.

c) Zbiór liczb rzeczywistych odcinka $[a, b]$, gdzie $a < b$, jest równoliczny ze zbiorem liczb rzeczywistych odcinka [0, 1]. Odpowiednią bijekcją jest tu f: $[a, b] \rightarrow [0, 1]$, zdefiniowana wzorem

$$f(x) = (x - a)/(b - a)$$

d) Zbiór liczb rzeczywistych jest równoliczny z podzbiorem liczb rzeczywistych odcinka otwartego $(-\pi/2, \pi/2)$. Wzajemnie jednoznaczne odwzorowanie reprezentuje tu funkcja tangens.

Definicja 5.2

Każdy zbiór skończony lub równoliczny ze zbiorem liczb naturalnych nazywa się *zbiorem przeliczalnym*. Zbiór nieskończony, który nie jest równoliczny ze zbiorem liczb naturalnych, nazywa się *zbiorem nieprzeliczalnym*.

5.2. Zbiory nieprzeliczalne

Twierdzenie 5.2

Zbiór potęgowy zbioru liczb naturalnych nie jest równoliczny ze zbiorem liczb naturalnych, czyli jest zbiorem nieprzeliczalnym. Oznacza to, że nie istnieje wzajemnie jednoznaczna funkcja $f: Nat \rightarrow 2^{Nat}$.

Dowód

Dowód pochodzi od Cantora i jest oparty na tzw. *metodzie przekątniowej*. Jeżeli zbiór potęgowy byłby równoliczny zbiorowi liczb naturalnych, to wszystkie podzbiory zbioru liczb naturalnych dałoby się ustawić w ciągi Z_1, Z_2, Z_3, ... w jeden ciąg. Każdy z tych zbiorów zawiera pewne liczby naturalne. Można to przedstawić w postaci tablicy, której przykładowa postać jest podana poniżej. Zbiór Z_1 w tej tablicy nie zawiera liczb 0, 1, 2, 3 itd.; zbiór Z_2 zawiera 0, 1, nie zawiera 2, 3 itd.

	0	1	2	3	
Z_1	nie	nie	nie	nie	...
Z_2	tak	tak	nie	nie	...
Z_3	nie	tak	nie	nie	...
Z_4	tak	tak	nie	tak	...
Z_5	...	...	...	...	...

Definiuje się teraz nowy zbiór Z' w taki sposób, aby był on różny od każdego ze zbiorów Z_1, Z_2, ... Poruszając się wzdłuż przekątnej tabeli od górnego lewego pola, definiuje się w sposób następujący przynależność kolejnej liczby naturalnej do zbioru Z': każde słowo „nie" zastępuje się słowem „tak", a każde słowo „tak" zastępuje się słowem „nie". Jeżeli po takim zastąpieniu na przekątnej w kolumnie k znajduje się „nie", oznacza to, że $k \notin Z'$, a jeżeli znajduje się „tak", oznacza to, że $k \in Z'$.

W rozpatrywanym przykładzie otrzymuje się mianowicie:

	0	1	2	3	
Z_1	tak	nie	nie	nie	...
Z_2	tak	nie	nie	nie	...
Z_3	nie	tak	tak	nie	...
Z_4	tak	tak	nie	nie	...
Z_5	...	...	...	...	...

Zbiór Z', zdefiniowany przez tak określoną przekątną, zawiera 0, nie zawiera 1, zawiera 2 itd.

Tak zdefiniowany zbiór Z' jest różny od każdego ze zbiorów Z_1, Z_2, ..., zatem Z' jest zbiorem, który nie daje się zestawić w ciąg wszystkich podzbiorów zbioru liczb naturalnych. Ponieważ rodzina podzbiorów 2^{Nat} jest nieskończona i nie jest *równoliczna* ze zbiorem liczb naturalnych, jest więc zbiorem nieprzeliczalnym. ■

Twierdzenie pokazuje, że istnieją co najmniej dwa rodzaje nieskończoności. Pierwszy reprezentuje nieskończoność liczb naturalnych i nazywa się nieskończonością przeliczalną. Pozostałe rodzaje nieskończoności nazywa się nieskończonościami nieprzeliczalnymi. Drugi rozpatrywany tu rodzaj nieskończoności, reprezentujący wszystkie podzbiory liczb naturalnych, reprezentuje rodzaj nieprzeliczalności zwany *continuum*.

Na konstrukcji podobnej do prezentowanej w poprzednim dowodzie opiera się dowód twierdzenia 5.3.

Twierdzenie 5.3

Przeliczalna suma mnogościowa zbiorów przeliczalnych jest zbiorem przeliczalnym.

Dowód

Niech Z_1, Z_2, ... będzie ciągiem zbiorów przeliczalnych. Niech $Z_i =_{def} \{z_{i1}, z_{i2}, \ldots\}$ dla $i = 1, 2, \ldots$ Elementy tych zbiorów można ustawić w tabelę:

	1	2	3	4	...
Z_1	z_{11}	z_{12}	z_{13}	z_{14}	...
Z_2	z_{21}	z_{22}	z_{23}	z_{24}	...
Z_3	z_{31}	z_{32}	z_{33}	z_{34}	...
Z_4	z_{41}	z_{42}	z_{43}	z_{44}	...
Z_5	...	...	...	...	...

Wszystkie te elementy, nie pomijając żadnego, można ustawić w jeden wspólny ciąg w sposób, który określają strzałki. Ciąg ten zawiera wszystkie elementy wszystkich ciągów Z_1, Z_2, ... i jest oczywiście równoliczny ze zbiorem liczb naturalnych, co dowodzi tezy twierdzenia. ■

Twierdzenie 5.4

Podzbiór zbioru przeliczalnego jest zbiorem przeliczalnym.

Dowód

Niech A będzie zbiorem przeliczalnym oraz $B \subseteq A$. Jeżeli $A = B$ lub B jest zbiorem skończonym, to oczywiście B jest zbiorem przeliczalnym. Niech B będzie zbiorem nieskończonym, różnym od A. Ponieważ A jest zbiorem przeliczalnym, istnieje wzajemnie jednoznaczna funkcja $f : Nat \rightarrow A$. Należy pokazać, że istnieje wzajemnie jednoznaczna funkcja $g : Nat \rightarrow B$. Funkcję tę można zdefiniować rekursywnie, na podstawie funkcji f, w sposób następujący:

- $g(0) = f(k_1)$, gdzie k_1 jest najmniejszą liczbą naturalną, taką że $f(k_1) \in B$,
- $g(n) = f(k_n)$, gdzie k_n jest najmniejszą liczbą naturalną, taką że $k_{n-1} < k_n$ oraz $f(k_n) \in B$.

Funkcja g jest różnowartościowa i przekształca Nat na B. Zbiór B jest zatem równoliczny ze zbiorem liczb naturalnych, a więc jest zbiorem przeliczalnym. ■

Przykładowym, ważnym zbiorem nieprzeliczalnym jest zbiór liczb rzeczywistych. Wynika to z następującego twierdzenia:

Twierdzenie 5.5

Zbiór liczb rzeczywistych z odcinka [0, 1] jest zbiorem nieprzeliczalnym.

Dowód

Wystarczy pokazać, że nie istnieje funkcja $f : Nat \rightarrow [0, 1]$, która jest bijekcją, to znaczy $dom(f) = Nat$ oraz $ran(f) = [0, 1]$. Funkcję f można przedstawić w postaci ciągu – zbioru par:

$$\{<0, f(0)>, <1, f(1)>, <0, f(2)>, \ldots\}$$

Dalej ciąg ten będzie zapisywany w uproszczonej postaci:

$$f(0), f(1), f(2), \ldots, f(n), \ldots$$

gdzie wyrazy ciągu $f(n)$ są liczbami z przedziału [0, 1].

Pokażemy, że dla dowolnie wybranego ciągu f istnieje liczba $c \in [0, 1]$, która nie należy do tego ciągu.

Oznaczmy przez $[a_n, b_n] \subseteq [0, 1]$, dla $n \in Nat$, ciąg przedziałów z odcinka [0, 1]. Ciąg tych przedziałów jest zdefiniowany następująco:

Niech $[a_0, b_0] = [0, 1]$. Podzielmy ten odcinek na trzy podprzedziały: [0, 1/3], [1/3, 2/3], [2/3, 1] i wybierzmy z nich ten podprzedział, do którego nie należy wyraz $f(0)$. Wybrany podprzedział oznaczmy przez $[a_1, b_1]$. Podobnie postępując z przedziałem $[a_1, b_1]$, wyznaczymy przedział $[a_2, b_2]$, do którego nie należy wyraz $f(1)$ itd.

Na mocy konstrukcji wyznaczony ciąg podprzedziałów $[a_n, b_n]$, dla $n \in Nat$, ma następujące własności:

$f(n-1) \notin [a_n, b_n]$,

$b_n - a_n = 1/3^n$

$0 \le a_n \le a_{n+1} < b_{n+1} \le b_n \le 1$.

Ciągi liczbowe $\{a_n\}_{n \in Nat}$ oraz $\{b_n\}_{n \in Nat}$ są monotoniczne i ograniczone. Są one zbieżne do tej samej granicy $c \in [0, 1]$, gdyż $\lim_{n \to \infty}(b_n - a_n) = 0$. Liczba c należy do każdego z przedziałów $[a_n, b_n]$, a więc jest różna od każdego wyrazu $f(n)$, czyli nie należy do ciągu wyznaczonego przez funkcję f, z czego wynika teza.

Zauważmy, że podział danego odcinka na dwa podprzedziały domknięte, np. [0, ½], [1/2, 1] itd., nie pozwoliłby na zastosowanie opisanego postępowania dla przypadków, gdy ciąg $f(0)$, $f(1)$, $f(2)$, ..., $f(n)$, ... byłby zbieżny do $1/2^k$, dla dowolnego $k \in Nat$. Możliwe natomiast byłoby przeprowadzenie tego postępowania przy założeniu podziału danego odcinka na dowolną, większą niż 3, liczbę podprzedziałów. ■

Twierdzenie 5.6

Zbiór liczb rzeczywistych z odcinka otwartego (0, 1) jest równoliczny zbiorowi punktów wnętrzna kwadratu o boku długości 1.

Szkic dowodu

Każdy punkt kwadratu można określić parą liczb $<x, y>$, stanowiących współrzędne punktu. Przy założeniu, że bok kwadratu ma długość jeden, liczby te można zapisać w postaci nieskończonych ułamków dziesiętnych w postaci:

$x = 0, \alpha_1 \alpha_2 \ldots \alpha_n \ldots$

$y = 0, \beta_1 \beta_2 \ldots \beta_n \ldots$

gdzie α_n, β_n (dla $n = 1, 2, \ldots$) są cyframi 0, 1, ..., 9.

Parze liczb $<x, y>$ przyporządkowujemy liczbę z na odcinku (0, 1), również reprezentowaną nieskończonym ułamkiem dziesiętnym o postaci

$z = 0, \alpha_1 \beta_1 \alpha_2 \beta_2 \ldots \alpha_n \beta_n \ldots$

Jest oczywiste, że przy takim przyporządkowaniu różnym punktom kwadratu odpowiadają różne punkty odcinka. Wynika stąd, że zbiór punktów wnętrza kwadratu o boku jednostkowym jest równoliczny z pewnym podzbiorem odcinka o długości jeden. Moc zbioru punktów kwadratu jest zatem nie większa od mocy zbioru punktów odcinka.

Z drugiej strony zbiór punktów odcinka jest podzbiorem punktów kwadratu. Moc zbioru punktów odcinka jest zatem nie większa od mocy zbioru punktów kwadratu, z czego wynika, że obie moce są równe. ■

5.3. Liczby kardynalne

Pojęcie równoliczności zbiorów jest bardzo ważne. Jest ono między innymi punktem wyjścia do współczesnej definicji liczby. Równoliczność wprowadza pewną klasyfikację zbiorów. Zbiory tego samego rodzaju są równoliczne. Rodzaje te nazywa się *liczbami kardynalnymi*. Takim rodzajem jest na przykład rodzina zbiorów czteroelementowych. Liczby naturalne są odpowiednikiem liczb kardynalnych dla zbiorów skończonych. Liczbą kardynalną zbioru wszystkich liczb naturalnych jest $\aleph_0$ (*alef zero*), a liczbą kardynalną rodziny wszystkich podzbiorów zbioru liczb naturalnych jest $\mathbf{c}$ (*continuum*). Inaczej: liczba kardynalna jest pewną cechą zbioru.

Liczbę kardynalną zbioru A, czyli rodzaj zbiorów, do którego należy zbiór A, będziemy oznaczać przez $|A|$. Liczby kardynalnej nie należy utożsamiać z pojęciem liczby. W szczególności, w przypadku zbioru skończonego A, jego liczba kardynalna $|A|$ jest czym innym niż liczba elementów tego zbioru $card(A)$.

Liczby kardynalne można ze sobą porównywać. Niech $|A| = \alpha$ oraz $|B| = \beta$.

Przyjmujemy, że liczba kardynalna α *jest nie większa* niż liczba kardynalna β, co piszemy $\alpha \leq \beta$, jeżeli zbiór A jest równoliczny z podzbiorem zbioru B.

Jeżeli $\alpha \leq \beta$ oraz $\alpha \neq \beta$, to mówimy, że liczba kardynalna α *jest mniejsza* niż liczba kardynalna β, co piszemy $\alpha < \beta$.

Z wcześniejszych rozważań wynika więc, że $\aleph_0 < \mathbf{c}$.

Porównywanie liczb kardynalnych ma własność zwrotności, to znaczy

$$\alpha \leq \alpha$$

i przechodniości, to znaczy

jeżeli $\alpha \leq \beta$ oraz $\beta \leq \gamma$, to $\alpha \leq \gamma$.

Ważny związek pomiędzy liczbami kardynalnymi wyraża podane niżej twierdzenie Cantora–Bernsteina.

Twierdzenie 5.7

Dla dowolnych liczb kardynalnych α, β:

jeżeli $\alpha \leq \beta$ oraz $\beta \leq \alpha$, to $\alpha = \beta$.

Dowód twierdzenia można znaleźć na przykład w pracy [Rasiowa 1998].

Uogólnieniem twierdzenia 5.3 jest twierdzenie Cantora.

Twierdzenie 5.8

Dla dowolnego zbioru A: $|A| < |2^A|$.

Dowód

Twierdzenie oznacza, że nie istnieje wzajemnie jednoznaczna funkcja odwzorowująca zbiór A w zbiór potęgowy 2^A. Załóżmy, że $f: A \to 2^A$ jest funkcją na 2^A. Niech

$$A_0 = \{a \in A \mid a \notin f(a)\}$$

Ponieważ f jest funkcją na 2^A, to istnieje $a_0 \in A$ taki, że $f(a_0) = A_0$, tak więc $a_0 \in A_0$ wtedy i tylko wtedy, gdy $a_0 \in f(a_0)$.

Innymi słowy: $a_0 \in A_0$ wtedy i tylko wtedy, gdy $a_0 \notin A_0$.

Otrzymana sprzeczność dowodzi, że nie istnieje wzajemnie jednoznaczna funkcja odwzorowująca zbiór A w zbiór potęgowy 2^A. ∎

Ćwiczenia

1. Pokazać równoliczność zbioru liczb naturalnych i zbioru liczb pierwszych.
2. Pokazać równoliczność zbiorów:
   a) odcinek otwarty $(0, 1) \subset$ *Rzeczywiste*,
   b) odcinek półotwarty $[0, 1) \subset$ *Rzeczywiste*,
   c) okręg na płaszczyźnie o środku $(0, 0)$ i promieniu 1.
3. Pokazać, że zbiór potęgowy zbioru A jest równoliczny ze zbiorem funkcji określonych na A i o wartościach w zbiorze $\{0, 1\}$, czyli ze zbiorem $\{f \mid f: A \to \{0, 1\}\}$.
4. Ile jest rosnących ciągów liczb wymiernych zbieżnych do 1?
5. Ile jest relacji równoważności na zbiorze liczb naturalnych takich, że wszystkie ich klasy abstrakcji są skończone?
6. Udowodnić, że każdy zbiór rozłącznych odcinków na prostej jest przeliczalny. Pokazać, że istnieje nieprzeliczalny zbiór rozłącznych odcinków na płaszczyźnie.
7. Udowodnić, że jeżeli A nie jest zbiorem przeliczalnym i B jest zbiorem przeliczalnym, to A/B nie jest zbiorem przeliczalnym.
8. Udowodnić, że produkt kartezjański dwóch zbiorów przeliczalnych jest zbiorem przeliczalnym.
9. Udowodnić, że zbiór liczb wymiernych jest przeliczalny.
10. Udowodnić, że zbiór liczb niewymiernych jest nieprzeliczalny.
11. Udowodnić, że każdy zbiór nieskończony zawiera pewien podzbiór przeliczalny.
12. Udowodnić, że rodzina wszystkich skończonych podzbiorów zbioru przeliczalnego jest przeliczalna.

13. Udowodnić, że dla dowolnych liczb kardynalnych α, β, γ, zachodzą związki:
    a) $\alpha \leq \alpha$,
    b) jeżeli $\alpha \leq \beta$ oraz $\beta \leq \gamma$, to $\alpha \leq \gamma$.

14. Czy istnieje relacja równoważności $R \subseteq Rzeczywiste^2$, której każda klasa abstrakcji jest mocy $\aleph_0$ oraz:
    a) zbiór ilorazowy $Rzeczywiste/R$ jest mocy $\aleph_0$?
    b) zbiór ilorazowy $Rzeczywiste/R$ jest mocy $\mathfrak{c}$?

15. Które z poniższych zdań jest prawdziwe?
    a) Jeśli $f: A \to B$ jest różnowartościowa oraz nie jest na B, to $|A| < |B|$.
    b) Jeśli $|A| < |B|$ oraz $C \neq \emptyset$, to $|A \times C| < |B \times C|$.

6. Zbiory i funkcje obliczalne

6.1. Zbiory obliczalne i rekurencyjne

Z wykonywaniem obliczeń za pomocą komputerów wiąże się pojęcie *algorytmu*. Obliczeniami na komputerach rządzą ścisłe reguły, niekiedy mówi się nawet o regułach mechanicznych, mając na myśli to, że obliczenie można widzieć jako skończonej długości ciąg czynności, z których każda jest jednoznacznie zdefiniowana i – dodatkowo – jest realizowalna za pomocą ściśle zdefiniowanych środków. Do rozwiązania niektórych problemów, na przykład zadania obliczenia największego wspólnego podzielnika dwóch liczb, można wyobrazić sobie istnienie pewnego algorytmu. Trudno natomiast wyobrazić sobie istnienie algorytmu do przepowiadania wyniku rzutu monetą.

Określone tak intuicyjnie pojęcie algorytmu jest nieformalne, co utrudnia lub nawet uniemożliwia jego wykorzystanie w ścisłych rozważaniach. Próby ścisłego zdefiniowania pojęcia algorytmu[9] podjęto w pierwszej połowie ubiegłego stulecia. Ich rezultatem było powstanie kilku – jak się później okazało – równoważnych definicji algorytmu.

Niektóre z tych podejść wiązały się z ograniczeniem czynności wykonywanych w ramach algorytmu do manipulacji na symbolach. Oznacza to, że wykonywanie czynności polega na tworzeniu pewnych ciągów symboli (napisów) na podstawie innych ciągów symboli (napisów). Przykładem są tu algorytmy normalne Markowa[10].

Inne podejścia były oparte na wprowadzeniu pewnych „maszyn", które byłyby zdolne do samodzielnego wykonywania przetwarzania napisów. Przykładem znanych modeli algorytmu są maszyny opracowane przez Turinga[11] oraz Posta[12]. Oba te modele, opracowane niezależnie od siebie, są podobne i wiąże się z nimi hipoteza, nazywana *hipotezą Turinga–Posta* lub – częściej – *hipotezą Turinga*. Z tego względu dalej przedsta-

[9] Termin *algorytm* pochodzi od nazwiska arabskiego matematyka Abu Ja'far Muhammad ibn Musa Al-Kwarizmi (około 780–850).

[10] Andriej A. Markow (ur. 1903), syn innego matematyka Andrieja A. Markowa (1856–1922).

[11] Alan M. Turing (1921–1954).

[12] Emil Post (1897–1954).

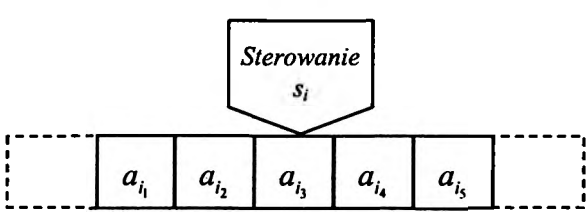
wiono tylko *maszynę Turinga*, a związana z nią, sformułowana w 1936 roku, hipoteza stwierdza [Arbib 1968]:

Nieformalne, intuicyjne pojęcie algorytmu na ciągach symboli jest tożsame ze ścisłym pojęciem procedury, którą może wykonać maszyna Turinga.

Dla hipotezy tego rodzaju nie można nigdy podać formalnego dowodu, ponieważ taki dowód wymaga zdefiniowania pojęć, które zawiera. Można ją tylko obalić przez podanie przykładu intuicyjnie rozwiązywalnego problemu, dla którego nie daje się skonstruować odpowiedniej maszyny Turinga. Jak dotychczas, ilekroć było intuicyjnie oczywiste, że algorytm istnieje, tylekroć okazywało się możliwe skonstruowanie maszyny Turinga wykonującej ściśle ten algorytm i nie ma przesłanek, które wskazywałyby na możliwość zmiany tego stanu rzeczy.

Rozwiązanie pewnego problemu przez wyszukanie odpowiedniego algorytmu sprowadza się zatem do zbudowania odpowiedniej maszyny Turinga. Maszyna Turinga jest tylko konstrukcją teoretyczną i nie służy do rozwiązywania zagadnień praktycznych. Do celów praktycznych wystarczy przyjąć, że maszynę Turinga można utożsamiać z dowolnym komputerem, który dysponuje nieograniczenie pojemną pamięcią. Dokładny jej opis przedstawia się następująco:

Maszyna Turinga jest określona jako urządzenie składające się z nieskończenie długiej taśmy, podzielonej na kratki, zwanej *pamięcią maszyny*, oraz urządzenia sterującego, zwanego *sterowaniem maszyny*. W każdej kratce taśmy może być zapisany jeden z symboli ustalonego, skończonego zbioru symboli A, nazywanego *alfabetem maszyny*. Zakłada się, że alfabet zawiera symbol „pusty": *nic*. Urządzenie sterujące – krócej: maszyna – może się znajdować w jednym ze stanów skończonego zbioru S. Wyróżnia się pewien podzbiór stanów $F \subseteq S$, nazywanych *stanami końcowymi*. W każdym stanie urządzenie sterujące może „obserwować" (przez głowicę czytająco-piszącą) jedną kratkę taśmy T. Gdyby kratki taśmy T można ponumerować liczbami całkowitymi, wtedy znajdujący się w obserwowanej kratce o numerze $i \in Całkowite$ symbol $a_i \in A$ nazywałby się *symbolem obserwowanym*.



Rys. 6.1. Maszyna Turinga

W danym momencie czasu maszynę charakteryzuje jej *konfiguracja*, na którą składa się stan pamięci, czyli aktualna zawartość taśmy, stan urządzenia sterującego oraz położenie głowicy czytająco-piszącej przy jednym z pól taśmy.

Maszyna rozpoczyna pracę w *konfiguracji początkowej*, to jest takiej, w której taśma T ma ustaloną początkową zawartość. Aktualnym stanem urządzenia sterującego jest wyróżniony *stan początkowy*, a głowica czytająco-pisząca znajduje się przy kratce taśmy T wskazanej jako kratka początkowa. W danej konfiguracji takiej, w której stan urządzenia sterującego jest stanem s, maszyna „czyta" obserwowany symbol a, po czym:

- zmienia swój stan s na nowy stan $s' \in S$,
- zmienia obserwowany symbol a na $a' \in A$, w szczególności może to być ten sam symbol a lub symbol pusty,
- przesuwa swoją głowicę czytająco-piszącą o jedną kratkę w lewo albo w prawo, albo pozostawia ją w miejscu.

W ten sposób maszyna przechodzi do nowej konfiguracji i powtarza swoje działanie według opisanego schematu aż do momentu, gdy osiągnie *konfigurację końcową*, to jest taką, w której stan aktualny urządzenia sterującego określa się jako stan końcowy. Po osiągnięciu konfiguracji końcowej maszyna zatrzymuje się i nie wykonuje dalszych czynności.

Formalnie maszynę Turinga MT definiuje się jako siódemkę:

$$MT = <S, A, \delta, \lambda, \rho, s_0, F>$$

gdzie:

S jest zbiorem stanów,
A jest alfabetem,
$\delta : S \times A \to S$ jest funkcją przejść pomiędzy stanami,
$\lambda : S \times A \to A$ jest funkcją wyjść,
$\rho : S \times A \to \{-1, 0, 1\}$ jest funkcją przesunięcia głowicy,
s_0 jest stanem początkowym,
$F \subseteq S$ jest podzbiorem stanów końcowych.

Obliczenie maszyny Turinga MT dla danej taśmy T można opisać w postaci ciągu trójek:

$$< s_0, a_{i_0}, i_0 >, < s_1, a_{i_1}, i_1 >, \ldots, < s_n, a_{i_n}, i_n >, \ldots$$

gdzie:

s_k jest stanem urządzenia sterującego,
a_{i_k} jest zawartością kratki obserwowanej przez głowicę czytająco-piszącą,
i_k oznacza kratkę taśmy T, przy której znajduje się głowica czytająco-pisząca.

Pierwsza trójka $< s_0, a_{i_0}, i_0 >$ jest elementem konfiguracji początkowej maszyny, a następne trójki są określone następująco:

$$s_{k+1} = \delta(s_k, a_{i_k})$$
$$a'_{i_k} = \lambda(s_i, a_{i_k})$$
$$i_{k+1} = i_k + \rho(s_i, a_{i_k})$$

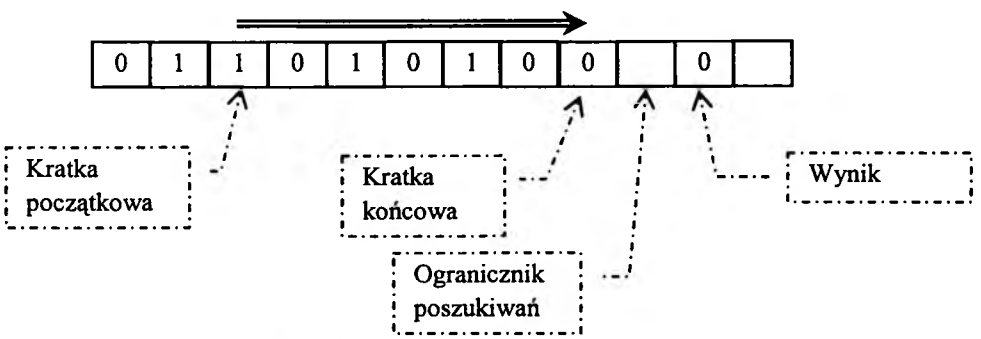
dla $k \in Nat$. Pierwsza z funkcji określa nowy stan, druga – nową zawartość przeczytanej kratki, a trzecia – wskazuje na numer kolejnej kratki, do której przesuwa się głowica.

Ciąg ten zawsze rozpoczyna się w konfiguracji początkowej i może być skończony lub nieskończony. Jeżeli jest on ciągiem skończonym, to ostatnia trójka jest elementem konfiguracji końcowej, co oznacza, że $s_n \in F$.

Obliczenie maszyny rozpoczyna się i kończy pewnym ciągiem symboli zapisanych na taśmie. Symbole na taśmie przed rozpoczęciem obliczeń interpretuje się jako dane algorytmu, a symbole po wykonaniu obliczeń maszyny interpretuje się jako wynik obliczeń algorytmu.

Przykład 6.1

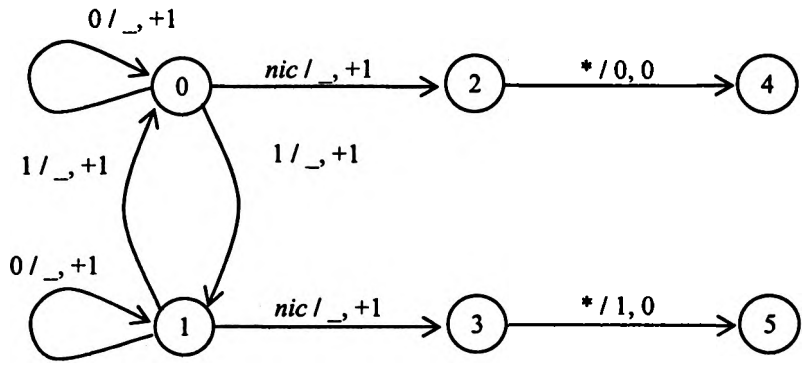
Alfabet maszyny Turinga $A = \{0, 1, nic\}$. Zadaniem maszyny Turinga jest stwierdzenie, czy liczba symboli 1 zapisana na taśmie, poczynając od wskazanej niepustej kratki aż do pierwszej kratki po prawej stronie, która zawiera symbol *nic*, jest parzysta czy nieparzysta. Wynik obliczeń maszyny ma być zapisany na taśmie, w kratce sąsiadującej po prawej stronie z pierwszą kratką zawierającą znaleziony symbol *nic* (kratka pusta).



Rys. 6.2. Przykładowa zawartość taśmy

Maszynę Turinga można w zwarty sposób przedstawić za pomocą diagramu przejść pomiędzy stanami. Diagram ten jest grafem, którego wierzchołkami są stany maszyny. Łuki reprezentujące przejścia pomiędzy stanami są etykietowane napisami postaci *a* / *b*, *c*. Pierwszy element *a* jest symbolem alfabetu maszyny, symbol * oznacza, że może to być dowolny element. Drugi element *b* jest symbolem, który maszyna wypisuje na taśmie, symbol _ oznacza, że napis w danej kratce nie ulega zmianie. Trzeci element *c* może przyjąć wartości –1, 0, +1, co oznacza przesunięcie głowicy maszyny odpowiednio w lewo, brak przesunięcia lub przesunięcie w prawo. Ukośnik rozdziela symbol wejściowy od symboli, które reprezentują reakcję maszyny w danej konfiguracji. Łącznie etykieta *a* / *b*, *c* na przejściu ze stanu

i do stanu *j* oznacza, że: jeżeli w stanie *i* maszyna przeczyta w obserwowanej kratce symbol *a*, to zapisuje w tej kratce symbol *b* i przesuwa się do następnej kratki o *c*, po czym przechodzi do stanu *j*.



Rys. 6.3. Diagram przejść pomiędzy stanami

Przedstawiony diagram wymaga uzupełnienia o wskazanie stanu początkowego – stan 0 – oraz stanów końcowych – stany 4, 5.

Maszyna Posta, a także inne maszyny, na przykład Rabina i Scotta, maszyny wielotaśmowe są równoważne maszynie Turinga w tym sensie, że jeżeli dany problem daje się rozwiązać przez zbudowanie jakiejkolwiek z tych maszyn, to daje się również rozwiązać za pomocą pozostałych maszyn.

Przyjmując maszynę Turinga za pojęcie algorytmu, można sformułować ważne pojęcia. Niech dany będzie pewien zbiór przeliczalny *A*.

Zbiór *A* jest *rekurencyjny*, jeżeli istnieje algorytm rozstrzygania, czy dany element jest czy nie jest elementem *A*.

Zbiór *A* jest *rekurencyjnie przeliczalny*, jeżeli istnieje algorytm, który wylicza wszystkie jego elementy.

Z definicji bezpośrednio wynika, że jeżeli podzbiór liczb naturalnych jest rekurencyjny, to jest rekurencyjnie przeliczalny, ale nie odwrotnie, gdyż okazuje się, że zachodzi twierdzenie:

Twierdzenie 6.1

Istnieje rekurencyjnie przeliczalny zbiór liczb naturalnych, który nie jest rekurencyjny.

Dowód twierdzenia można znaleźć na przykład w książce [Arbib 1968].

6.2. Funkcje obliczalne i rekurencyjne

Pojęcie funkcji obliczalnych wiąże się z funkcjami określonymi na zbiorze liczb naturalnych i o wartościach w zbiorze liczb naturalnych. Intuicyjnie, funkcje obliczalne to takie funkcje, których wartości dla dowolnych argumentów można obliczyć na komputerze w skończonej liczbie kroków. Formalnie funkcja jest obliczalna, gdy istnieje algorytm jej obliczenia, inaczej – istnieje dla niej odpowiednia maszyna Turinga. Próby scharakteryzowania funkcji obliczalnych doprowadziły do wyłonienia klasy funkcji rekurencyjnych. Okazało się, że funkcje obliczalne są funkcjami rekurencyjnymi. Dokładniej wyraża to powszechnie akceptowana hipoteza Churcha, która stwierdza, że:

Każda funkcja obliczalna w nieformalnym sensie jest rekurencyjna.

Definicja klasy *funkcji rekurencyjnych* opiera się na zbiorze pewnych funkcji elementarnych i zbiorze operacji, które pozwalają na konstruowanie z funkcji elementarnych nowych funkcji.

Zbiór funkcji elementarnych zawiera:

- funkcję następnika zdefiniowaną wzorem $Succ(x) = x + 1$,
- funkcję tożsamościową $I(x) = x$,
- funkcje rzutowania $p_i(x_1, ..., x_n) = x_i$, dla $i = 1, ..., n$,
- funkcję zeroargumentową – stałą 0.

Zbiór operacji na funkcjach składa się z trzech operacji. Pierwszą jest – omówiona wcześniej – operacja składania funkcji, dwie pozostałe operacje – nazywane operacją rekursji prostej i operacją minimum – wymagają zdefiniowania.

Operacja rekursji prostej polega na tym, że mając dwie funkcje:

$f : Nat^{n-1} \to Nat$ oraz $g : Nat^{n+1} \to Nat$ dla $n \in Nat \setminus \{0\}$

nową funkcję

$h : Nat^n \to Nat$

definiuje się za pomocą dwóch następujących równości:

$h(x_1, ..., x_{n-1}, 0) = f(x_1, ..., x_{n-1})$

$h(x_1, ..., x_{n-1}, Succ(x_n)) = g(x_1, ..., x_n, h(x_1, ..., x_n))$

Termin rekursja prosta, wprowadzony przez Hilberta[13] i Bernaysa[14] w 1934 roku, nie jest szczęśliwy, gdyż schemat generacji wartości funkcji bardziej się wiąże z iteracją, z jaką mamy do czynienia w językach programowania, niż z rekursją. Rekursja prosta wyraża pewien indukcyjny sposób definiowania wartości.

[13] David Hilbert (1862–1943).

[14] Paul I. Bernays (1888–1977).

Funkcje, które daje się zdefiniować za pomocą operacji składania i rekursji prostej, nazywa się funkcjami *pierwotnie rekurencyjnymi*.

Przykład 6.2

Funkcję dodawania liczb naturalnych, reprezentowaną symbolem + w notacji przedrostkowej, definiuje się za pomocą operacji rekursji prostej w sposób następujący:

$$+(x, 0) = I(x)$$
$$+(Succ(y), x) = Succ(p_3(x, y, +(x, y)))$$

W tym przypadku rolę definiowanej funkcji h pełni +, funkcja f jest funkcją tożsamościową I, a funkcja g jest złożeniem funkcji następnika *Succ* z funkcją rzutowania p_3. Tę samą definicję, w sposób równoważny, można zapisać prościej:

$$+(x, 0) = x$$
$$+(Succ(y), x) = Succ(+(x, y))$$

Po zastosowaniu notacji wrostkowej dla funkcji dwuargumentowych definicja przyjmie jeszcze bardziej czytelną postać:

$$x + 0 = x$$
$$Succ(y) + x = Succ(x + y)$$

Korzystając z funkcji dodawania, podobnie można zdefiniować funkcję mnożenia:

$$x * 0 = 0$$
$$Succ(y) * x = (x * y) + x$$

Przykład 6.3

Odejmowanie w zbiorze liczb naturalnych jest funkcją zdefiniowaną częściowo. Definiowana poniżej funkcja *dif*, określona dla wszystkich liczb naturalnych, jest tylko pewnym odpowiednikiem odejmowania w zbiorze liczb całkowitych. Jej definicja wymaga wprowadzenia funkcji pomocniczej *Pred*, nazywanej funkcją *poprzednika*. Rekursywna definicja jednoargumentowej funkcji poprzednika ma postać:

$$Pred(0) = 0$$
$$Pred(Succ(x)) = x$$

Odejmowanie w dziedzinie liczb całkowitych nieujemnych, oznaczone *dif* w celu odróżnienia od symbolu odejmowania „–" w zbiorze liczb całkowitych, jest zdefiniowane metodą rekursji prostej:

$$dif(x, 0) = 0$$
$$dif(x, Succ(y)) = Pred(dif(x, y))$$

Metodą rekursji prostej można definiować różne funkcje, między innymi funkcje określone wariantowo.

Przykład 6.4

Niech dana będzie funkcja

$$h(x)=\begin{cases}2x & \text{dla } x\le 3\\ 2x-2 & \text{dla } x>3\end{cases}$$

Jej definicja wymaga uprzedniego zdefiniowania funkcji pomocniczych: jednoargumentowej funkcji znaku *sg*, definiowanej rekursją prostą (przypadek zdegenerowany):

$sg(0) = 0$

$sg(Succ(x)) = 1$

oraz funkcji porównań definiowanych przez wyrażenia funkcyjne:

$gt(x, y) = sg(dif(x, y))$

$ge(x, y) = gt(Succ(x), y)$

Definicja funkcji *h* przybierze zatem postać wyrażenia funkcyjnego

$((2*x) * ge(3, x)) + (dif((2*x), 2) * gt(x, 3))$

Niestety, za pomocą operacji rekursji prostej nie można definiować dowolnych funkcji. Przykładem funkcji, która nie daje się zdefiniować w ten sposób, jest przedstawiana już poprzednio, w rozdziale 3, funkcja Ackermanna[15]

$$Ack(x,y)=\begin{cases}y+1 & \text{gdy } x=0\\ Ack(x-1,1) & \text{gdy } y=0\\ Ack(x-1,Ack(x,y-1)) & \end{cases}$$

W 1936 roku Kleene[16] uzupełnił listę schematów kompozycji funkcji o operację minimum, albo inaczej – operację μ-rekursji.

Niech dana będzie funkcja

$f: Nat^{n+1} \to Nat$

taka, że dla każdych $x_1, ..., x_n \in Nat$ istnieje $y \in Nat$ takie, że $f(x_1, ..., x_n, y) = 0$.
Operacja minimum dla funkcji $f: Nat^{n+1} \to Nat$ polega na zdefiniowaniu nowej funkcji

$h : Nat^n \to Nat$

[15] Wilhelm Ackermann (1896–1962).

[16] Stephen Kleene (1909–1994).

która spełnia warunek

$f(x_1, ..., x_n, h(x_1, ..., x_n)) = 0$

oraz dodatkowo – aby zapewnić jednoznaczność definicji funkcji h – warunek

$h(x_1, ..., x_n)$ jest równe najmniejszej wartości $y \in Nat$ takiej, że $f(x_1, ..., x_n, y) = 0$.

Ostatni warunek zapisuje się też w postaci

$h(x_1, ..., x_n) = \mu y[f(x_1, ..., x_n, y) = 0]$

Symbol μy oznacza najmniejszą wartość y, dla której, przy danych wartościach $x_1, ..., x_n$, jest spełniony warunek $f(x_1, ..., x_n, y) = 0$.

Operacja minimum wyznacza więc funkcję, która przyjmuje wartość $h(x_1, ..., x_n) = y$ wtedy i tylko wtedy, gdy:

- $f(x_1, ..., x_n, y) = 0$
- dla dowolnego $y' < y$ $f(x_1, ..., x_n, y') \neq 0$.

Jeżeli dla danego zestawu wartości $x_1, ..., x_n$ funkcja f nie spełnia podanych warunków, to funkcja h dla tych wartości nie jest zdefiniowana.

Przykład 6.5

Operację minimum wykorzystano do definicji funkcji

$h_1(x, y) = (\mu z)[isg(eq(y * z, x)) = 0]$

Funkcja h_1 definiuje najmniejszą wartość z taką, że $y * z = x$. Gdy x nie jest wielokrotnością y, funkcja ta nie jest określona.

$h_2(x, y) = (\mu z)[y * gt(x, y * Succ(z)) = 0]$

Funkcja h_2 wyznacza część całkowitą z dzielenia x przez y.

$h_3(x, y) = (\mu z)[dif(x, z) = 0]$

Funkcja h_3 równa się x dla dowolnych x, y; jest więc równoważna funkcji projekcji p_1.

Funkcje, które można zdefiniować za pomocą operacji składania, rekursji pierwotnej i minimum nazywa się też funkcjami *ogólnie rekurencyjnymi*.

Uwaga

Chociaż funkcje rekurencyjne są definiowane jako funkcje przekształcające liczby naturalne w liczby naturalne, to ich własności można przenieść na dowolne funkcje, których zbiory argumentów i wartości dają się opisać skończonymi ciągami symboli. Niech $\{a_1, a_2, ..., a_n, ...\}$ będzie zbiorem symboli wykorzystywanych do

tworzenia takich opisów. Każdemu opisowi argumentu lub wartości funkcji, który jest reprezentowany przez skończony ciąg postaci

$$a_{i_1}, a_{i_2}, \ldots, a_{i_k}$$

można przyporządkować w jednoznaczny sposób liczbę naturalną. Uniwersalny sposób kodowania, zwany numeracją Gödla, polega na przyporządkowaniu temu ciągowi liczby

$$2^{i_1}3^{i_2}\ldots p_k^{i_k}$$

gdzie 2, 3, ..., p_k są kolejnymi liczbami pierwszymi. Dzięki jednoznaczności rozkładu liczb naturalnych na czynniki pierwsze można sprawdzić, czy dana liczba jest przyporządkowana jakiemuś opisowi, a jeśli tak – to jakiemu.

Ćwiczenia

1. Maszynę Turinga, przedstawioną w przykładzie 6.1, rozbudować w taki sposób, aby stwierdzała parzystą bądź nieparzystą liczbę symboli 1 pomiędzy pierwszą kratką po lewej i pierwszą kratką po prawej stronie początkowego położenia głowicy, które zawierają symbol *nic*.
2. Zdefiniować maszynę Turinga, która jako daną wejściową przyjmuje liczbę naturalną w zapisie binarnym i produkuje jako wynik tę samą liczbę zwiększoną o jeden, również w zapisie binarnym.
3. Zdefiniować maszynę Turinga, która jako dane wejściowe przyjmuje n-elementowy ciąg znaków, $n \in Nat \setminus \{0\}$ oraz liczbę $k \in \{1, \ldots, n\}$ i produkuje jako wynik k-ty element wejściowego ciągu.
4. Wykorzystując operację rekursji prostej, zdefiniować funkcję:
   a) potęgowania,
   b) minimum i maksimum dwóch liczb,
   c) dzielenia całkowitego i reszty z dzielenia całkowitego.
5. Zdefiniować jako funkcję rekurencyjną funkcję:
   a) najmniejszej wspólnej wielokrotności dwóch liczb naturalnych,
   b) największego wspólnego podzielnika dwóch liczb.
6. Zakładając, że znane są operacje dodawania i mnożenia na liczbach naturalnych, definiować rekursywnie operacje dodawania i mnożenia na liczbach całkowitych.

7. Języki formalne i gramatyki

7.1. Ciągi i słowa

Zbiory są nieuporządkowaną kolekcją pewnych elementów. Często potrzebne jest wprowadzenie uporządkowania wśród rozważanej kolekcji obiektów, a jednym ze sposobów uporządkowania jest zdefiniowanie ciągu.

Niech A będzie dowolnym zbiorem. Niepustym *ciągiem* o długości $n \in Nat \setminus \{0\}$ nad zbiorem A będzie się nazywać dowolną całkowicie określoną funkcję o sygnaturze

$$s : \{1, ..., n\} \to A$$

Ciąg o długości zero jest ciągiem *pustym* i będzie oznaczany symbolem ε.

Przez $CiągiSkończone_n(A)$ będzie oznaczany zbiór wszystkich ciągów skończonych długości $n \in Nat$ nad zbiorem A. Z definicji:

$$CiągiSkończone_0(A) =_{def} \{\varepsilon\}$$
$$CiągiSkończone_n(A) =_{def} \{s \mid s : \{1, ..., n\} \to A \wedge dom(s) = \{1, ..., n\}\}$$

dla $n \in Nat \setminus \{0\}$.

Zbiorem wszystkich ciągów skończonych będzie zatem

$$CiągiSkończone(A) =_{def} \bigcup_{n \in Nat} CiągiSkończone_n(A)$$

Zbiór wszystkich ciągów nieskończonych nad A jest zdefiniowany jako

$$CiągiNieskończone(A) =_{def} \{s \mid s : Nat \setminus \{0\} \to A \wedge dom(s) = Nat \setminus \{0\}\}$$

Zbiorem wszystkich ciągów nad A jest zatem zbiór

$$Ciągi(A) =_{def} CiągiSkończone(A) \cup CiągiNieskończone(A)$$

Dalej będą używane następujące oznaczenia: Niech s będzie niepustym ciągiem nad A. Wartość funkcji s dla argumentu i, czyli $s(i)$, oznacza i-ty element ciągu. Skończony ciąg s o długości $n \in Nat \setminus \{0\}$ nad zbiorem A jest zbiorem par:

$$\{<1, s(1)>, ..., <n, s(n)>\}$$

a nieskończony ciąg jest zbiorem par:

$$\{<1, s(1)>, ..., <n, s(n)>, ... \}$$

Zwykle używa się uproszczonego zapisu ciągu, odpowiednio w postaci:

$s(1)\, s(2) \ldots s(n)$ lub $s(1)\, s(2) \ldots s(n) \ldots$

albo

$a_1\, a_2 \ldots a_n$ lub $a_1\, a_2 \ldots a_n \ldots$

gdzie $a_i = s(i)$ dla $i = 1, \ldots, n, \ldots$ dla $i \in Nat \backslash \{0\}$.

Ciągi zapisywane w uproszczonej postaci będą oznaczane literami greckimi α, β, γ itd. Będzie się pisać na przykład $\alpha =_{def} a_1\, a_2 \ldots a_n$, a przez $len(\alpha)$ będzie się oznaczać długość ciągu α. Oczywiście $len(\alpha) = 1$.

Równość ciągów oznacza równość reprezentujących je funkcji. Zapis $\alpha = \beta$ oznacza, że ciąg α jest identyczny z ciągiem β.

Przykład 7.1

Ciągami nad $A = \{0, 1, 2, 4, 5\}$ będą napisy:

0
001
12345

Ciągami nad *Nat* będą napisy:

11
1 1
11 1 0 54

Należy zwrócić uwagę na to, że pierwszy i drugi ciąg są ciągami różnymi. Pierwszy składa się z jednego, a drugi – z dwóch elementów. Aby unikać wątpliwości przy identyfikacji elementów ciągu, można stosować elementy rozdzielające – separatory, na przykład odstępy, jak wyżej, lub inne symbole, różne od elementów ciągu.

Uproszczony zapis ciągów pozwala na stwierdzenie, że zbiór ciągów długości $n \in Nat \backslash \{0\}$ nad zbiorem A można utożsamiać ze zbiorem wszystkich n-krotek nad zbiorem A, czyli z n-krotnym produktem kartezjańskim A^n nad zbiorem A. Oznacza to, że istnieje wzajemnie jednoznaczne odwzorowanie pomiędzy zbiorem ciągów o długości n a produktem kartezjańskim A^n, zatem

zbiorowi $CiągiSkończone_0(A)$ odpowiada zbiór $A^0 =_{def} \{<>\}$
zbiorowi $CiągiSkończone_n(A)$ odpowiada zbiór A^n dla $n \in Nat \backslash \{0\}$
zbiorowi $CiągiSkończone(A)$ odpowiada zbiór $\bigcup_{n \in Nat} A^n$

Ciągi zapisywane w uproszczonej postaci nazywa się *słowami*, a zbiór A nazywa się *alfabetem*. W dalszej części rozdziału będą używane właśnie te terminy.

Zbiór A^* jest zatem zbiorem wszystkich skończonych słów nad alfabetem A. Zbiór wszystkich niepustych skończonych słów nad alfabetem A będzie oznaczany przez A^+

$$A^+ =_{def} A^* \setminus \{\varepsilon\}$$

7.2. Operacje na słowach

Niech dany będzie dowolny, co najwyżej przeliczalny, alfabet A oraz niech A^* będzie zbiorem wszystkich skończonych słów nad A. Na słowach można definiować różne operacje. Podstawową jest operacja konkatenacji słów.

Niech $\alpha, \beta \in A^*$ będą dowolnymi słowami nad alfabetem A.

Konkatenacja słów α, β, co zapisuje się $\alpha \wedge \beta$, jest słowem, które powstaje przez dopisanie na końcu słowa α słowa β. Jeżeli

$$\alpha = a_1 \ldots a_n \quad \text{oraz} \quad \beta = b_1 \ldots b_m$$

to

$$\alpha \wedge \beta = a_1 \ldots a_n \wedge b_1 \ldots b_m =_{def} a_1 \ldots a_n b_1 \ldots b_m$$

Niech $\alpha, \beta, \gamma \in A^*$. Konkatenacja słów ma następujące oczywiste własności:

$$\varepsilon \wedge \varepsilon = \varepsilon$$
$$\varepsilon \wedge \alpha = \alpha \wedge \varepsilon = \alpha$$
$$(\alpha \wedge \beta) \wedge \gamma = \alpha \wedge (\beta \wedge \gamma)$$

Ze względu na te własności nawiasy będą dalej opuszczane.

Słowo $\beta \in A^+$ jest *podsłowem* słowa $\alpha \in A^+$, gdy istnieją słowa $\gamma, \delta \in A^*$ takie, że

$$\alpha = \gamma \wedge \beta \wedge \delta$$

Jeżeli $\gamma \wedge \delta \neq \varepsilon$, to β jest *podsłowem właściwym* słowa α, jeżeli $\gamma = \varepsilon$, to β jest *podsłowem początkowym* słowa α, a jeżeli $\delta = \varepsilon$, to β jest *podsłowem końcowym* słowa α.

Konkatenacja jest podstawą do definiowania innych operacji na słowie:

- n-krotnej *iteracji* słowa,
- *czoła* słowa,
- *ogona* słowa,
- *inwersji* słowa.

Niech $\alpha = a_1 \ldots a_n$. Operacja n-krotnej *iteracji* słowa, dla $n \in Nat$, jest zdefiniowana rekursywnie następująco:

$$\alpha^0 =_{def} \varepsilon$$
$$\alpha^{n+1} =_{def} \alpha^n \wedge \alpha \qquad \text{dla } n \in Nat$$

Operacje czoła *head* i ogona *tail*, określone następująco:

$head(\alpha) =_{def} a_1$ oraz $tail(\alpha) =_{def} a_2 \ldots a_n$

oznaczają odpowiednio pierwszy element słowa $\alpha = a_1 \ldots a_n$ oraz nowe słowo, które powstaje z α przez usunięcie jego pierwszego elementu. Oczywiście, dla dowolnego niepustego słowa zachodzi własność

$\alpha = head(\alpha) \wedge tail(\alpha)$

Operacja inwersji słowa $\alpha = a_1 \ldots a_n$, zapisywana w postaci α^{-1}, określona następująco:

$\alpha^{-1} =_{def} a_n a_{n-1} \ldots a_1$

oznacza lustrzane odbicie tego słowa.

Przykład 7.2

Niech $A = \{a, b, c\}$, wówczas słowami nad A są na przykład:

a aab $cabca$

Konkatenacją dwóch ostatnich słów jest słowo

$aab \wedge cabca = aabcabca$

Iteracjami słów są na przykład

$a^3 = aaa$

$(aab)^2 = aabaab$

$(cabca)^1 = cabca$

Operacje czoła i ogona dla dwóch pierwszych słów wyznaczają słowa:

$head(a) = a$ $tail(a) = \varepsilon$

$head(aab) = a$ $tail(aab) = ab$

Inwersjami słów są:

$a^{-1} = a$

$(aab)^{-1} = baa$

$(cabca)^{-1} = acbac$

Dla uproszczenia notacji, gdy nie będzie to wprowadzać niejednoznaczności, zamiast $\alpha \wedge \beta \wedge \gamma$ będzie się pisać $\alpha\beta\gamma$.

Dalej definiuje się jeszcze dwie operacje na słowach.

Najpierw wprowadza się pojęcie produkcji. Para słów $\beta, \gamma \in A^*$ zapisywana w postaci

$\beta ::= \gamma$

będzie nazywana *produkcją* lub *regułą przepisywania*.

Produkcję $\beta ::= \gamma$ można traktować tak samo jak uporządkowaną parę $<\beta, \gamma>$.

Symbol ::=, czytany: *jest zastępowany przez*, pełni rolę separatora oddzielającego dwa elementy. Słowo β po lewej stronie produkcji jest nazywane *poprzednikiem*, a słowo γ po prawej stronie produkcji jest nazywane *następnikiem* produkcji.

Niech $\alpha \in A^*$ oraz niech $\beta ::= \gamma$ będzie pewną produkcją.

Słowo δ jest wyprowadzeniem ze słowa α na podstawie produkcji $\beta ::= \gamma$, co zapisuje się w postaci

$$\alpha \xrightarrow{\beta ::= \gamma} \delta$$

gdy są spełnione warunki:

$\alpha = \alpha_1 \beta \alpha_2$

$\delta = \alpha_1 \gamma \alpha_2$

Przykład 7.3

Niech $A = \{a, b, c\}$. Ze słowa *aabcaa*, stosując produkcję *aa* ::= *cba*, można wyprowadzić słowa:

cbabcaa oraz *aabccba*

czyli

$$aabcaa \xrightarrow{aa::=cba} cbabcaa \quad \text{oraz} \quad aabcaa \xrightarrow{aa::=cba} aabccba$$

Jak pokazuje przykład, operacja wyprowadzenia nowego słowa δ ze słowa α na podstawie produkcji $\beta ::= \gamma$ nie musi być jednoznaczna. Liczba możliwych wyprowadzeń zależy od liczby wystąpień podsłowa β w słowie α. W szczególnym przypadku, gdy poprzednik reguły nie jest podsłowem w α, nie można wyprowadzić nowego słowa.

Niech $a ::= \beta$ będzie produkcją, w której poprzednik jest tylko pojedynczym symbolem $a \in A$ (słowem długości jeden), a następnik – jak poprzednio – jest dowolnym słowem $\beta \in A^*$ nad alfabetem A. Produkcja takiej postaci będzie nazywana *podstawieniem*.

Niech $\alpha \in A^*$ oraz niech $a ::= \beta$ będzie pewnym podstawieniem.

Słowo γ powstaje ze słowa α *przez podstawienie* $a ::= \beta$, co zapisuje się w postaci

$\alpha[a ::= \beta]$

gdy każde wystąpienie symbolu a w słowie α jest zastąpione słowem β.

Przykład 7.4

Niech $A = \{a, b, c\}$. W wyniku operacji określonej przez podstawienie $a ::= cba$ słowo $abcab$ zostanie przekształcone w słowo

$cbabccbab$

czyli

$abcab[a ::= cba] = cbabccbab$

Podobnie:

$abbacc[a ::= cbc] = cbcbbcbccc$
$abbacc[b ::= cbc] = acbccbcacc$
$abbacc[c ::= cbc] = abbacbccbc$

7.3. Języki formalne

Językiem formalnym L nad alfabetem A nazywa się dowolny podzbiór zbioru A^*, czyli $L \subseteq A^*$.

Język formalny jest tylko pewnym przybliżeniem języka naturalnego lub sztucznego, gdyż wyraża on tylko składniowy aspekt języka. W myśl wprowadzonej definicji alfabetem dla języka naturalnego jest zbiór słów w danym języku, a odpowiadający mu język formalny może być zbiorem wszystkich zdań w tym języku. W przypadku języka programowania alfabetem jest zbiór symboli leksykalnych, a odpowiadający mu język formalny definiuje zbiór wszystkich poprawnie tekstowo zbudowanych programów. Język formalny nie określa znaczenia i tym samym nie gwarantuje sensowności zdania czy programu, wyraża wyłącznie poprawność tekstową (składniową) zdania lub programu.

Przykład 7.5

Niech $A = \{a, b, c\}$, wówczas językami formalnymi nad A są na przykład skończone zbiory słów:

$\{a\}, \{aab, c\}, \{a, b, c, ab, cba\}$

Wykorzystując operację k-iteracji słów, dla $k \in Nat \setminus \{0\}$, można zdefiniować również pewne nieskończone języki formalne nad A, na przykład:

$\{s \in A^* \mid s = a^k \wedge b^l \wedge c^m \wedge k < l < m \wedge k, l, m \in Nat\}$
$\{a, b, c, ab, cba\} \cup \{s \in A^* \mid s = a^k \wedge b^{k+1} \wedge k \in Nat\}$

Niech A oraz B będą dwoma alfabetami. Funkcję $h : A^* \to B^*$ nazywa się *morfizmem* wtedy i tylko wtedy, gdy dla dowolnych $\alpha, \beta \in A^*$

$h(\alpha \wedge \beta) = h(\alpha) \wedge h(\beta)$

Morfizm nazywa się *izomorfizmem*, gdy h jest funkcją różnowartościową.

Przykład 7.6

Dla alfabetów $A = \{0, 1, 2, ..., 9\}$ oraz $B = \{0, 1\}$ izomorfizmem jest funkcja

$h(0) = 0000, h(1) = 0001, ..., h(9) = 1001$

wyrażająca kodowanie binarne liczb dziesiętnych.

Warto zwrócić uwagę, że alfabet przeliczalny A nie ma większej siły ekspresji niż dowolny alfabet skończony B. Oznacza to, że dla dowolnego języka formalnego $L_A \subseteq A^*$ istnieje taki język $L_B \subseteq B^*$, że istnieje wzajemnie jednoznaczne odwzorowanie pomiędzy obu językami $f: L_A \rightarrow L_B$.

Istotnie, niech będzie dany przeliczalny alfabet A o symbolach $a_1, a_2, a_3, ...$ oraz alfabet B zawierający tylko dwa symbole, na przykład 0, 1. Istnieje wzajemne odwzorowanie elementów alfabetu A w pewien podzbiór ciągów zero-jedynkowych nad alfabetem B. Na przykład ciągi binarne 1, 11, 111, ... itd. mogą być kodami indeksów kolejnych symboli $a_1, a_2, a_3, ...$ Dowolne słowo nad alfabetem A można przestawiać jako konkatenację odpowiednich ciągów kodujących nad alfabetem B. Na przykład słowo $a_3\ a_2\ a_2$ w alfabecie A będzie jednoznacznie reprezentowane przez słowo 111011011 w alfabecie B – symbol 0 pełni tu rolę separatora między kodami kolejnych symboli alfabetu A. Oznacza to, że dla dowolnego języka formalnego L_A nad A istnieje funkcja, która wzajemnie jednoznacznie odwzorowuje ten język w pewien język L_B nad B. Ciągi binarne mogą pełnić tę samą rolę, jaką pełnią symbole alfabetu A, co wyjaśnia powszechność stosowania kodowania binarnego.

Ponieważ języki formalne są zbiorami, można więc na nich wykonywać dowolne operacje mnogościowe. Jeżeli L_1, L_2 są językami nad alfabetem A, to także $L_1 \cup L_2$, $L_1 \cap L_2$ oraz L_1/L_2 są językami nad A.

Na językach można zdefiniować operację konkatenacji, która jest uogólnieniem konkatenacji zdefiniowanej na słowach. *Konkatenacja* języków $L_1 \subseteq A^*$, $L_2 \subseteq B^*$, oznaczana $L_1 \wedge L_2$, jest określona następująco:

$$L_1 \wedge L_2 =_{def} \{\alpha\beta \mid \alpha \in A^* \wedge \beta \in B^*\}$$

Korzystając z konkatenacji języków, wprowadza się operację *iteracji języków*, określoną dla dowolnego języka i liczby naturalnej n w sposób następujący:

$$L^0 =_{def} \{\varepsilon\}$$
$$L^{n+1} =_{def} L^n \wedge L \qquad \text{dla } n \in Nat$$

oraz operację *domknięcia* języka (nazywaną także gwiazdką Kleenego) określoną jako

$$L^* =_{def} \bigcup_{n \in Nat} L^n$$

Podobnie można uogólnić operacje *czoła*, *ogona* i *inwersji* dla języka $L \subseteq A^*$:

$HEAD(L) =_{def} \{\alpha \in A^* \mid \exists \beta \in A^* \bullet \alpha\beta \in L\}$

$TAIL(L) =_{def} \{\beta \in A^* \mid \exists \alpha \in A^* \bullet \alpha\beta \in L\}$

$L^{-1} =_{def} \{\alpha \mid \alpha^{-1} \in L\}$

7.4. Gramatyki bezkontekstowe

Nietrywialne języki formalne składają się z nieskończenie wielu słów. Nie można ich definiować enumeracyjnie, czyli przez jawne wyliczenie słów. Nieskończone języki formalne definiuje się rekursywnie, przy czym wykorzystuje się specyficzny mechanizm oparty na pojęciu *gramatyki* języka formalnego.

Gramatyka bezkontekstowa G jest czwórką

$G =_{def} <T, N, P, S>$

gdzie:

- T jest skończonym zbiorem, nazywanym *alfabetem symboli terminalnych*,
- N jest skończonym zbiorem, nazywanym *alfabetem symboli nieterminalnych*,
- P jest skończonym zbiorem *produkcji*,
- S jest wyróżnionym symbolem nieterminalnym, nazywanym *symbolem początkowym*.

Zakłada się, że zbiory symboli terminalnych i nieterminalnych są rozłączne, to jest

$N \cap T = \varnothing$

O pojedynczej produkcji $p \in P$ zakłada się, że jest postaci

$v ::= \alpha$

gdzie jej poprzednik v może być dowolnym symbolem nieterminalnym, czyli $v \in N$, a jej następnik α może być dowolnym niepustym słowem nad sumą mnogościową zbiorów symboli terminalnych i nieterminalnych, czyli $\alpha \in (T \cup N)^+$.

Gramatyka G generuje pewien język formalny $L(G) \subseteq T^*$. Nieformalnie jest to zbiór wszystkich słów nad alfabetem T, które można wyprowadzić z symbolu początkowego gramatyki S, za pomocą przekształceń, określonych przez zbiór P produkcji gramatyki.

Niech dane będą dwa słowa $\alpha, \beta \in (T \cup N)^+$.

Słowo β jest w gramatyce G *bezpośrednio wyprowadzane* ze słowa α, gdy istnieje taka produkcja $p \in P$, że

$\alpha \xrightarrow{p} \beta$

Fakt bezpośredniego wyprowadzenia słowa β ze słowa α w gramatyce G zapisuje się:

$$\alpha \xrightarrow[G]{} \beta$$

lub

$$\alpha \longrightarrow \beta$$

gdy z kontekstu wynika, o jaką gramatykę chodzi.

Słowo β jest w gramatyce G *wyprowadzone* ze słowa α, gdy istnieje skończony ciąg słów $\beta_1, \beta_2, ..., \beta_n \in (T \cup N)^+$ taki, że

$$\alpha = \beta_1 \qquad \beta_n = \beta$$

oraz

$$\beta_i \xrightarrow[G]{} \beta_{i+1} \qquad \text{dla } i \in \{1, 2, ..., n-1\}$$

Dalej symbol gramatyki G będzie pomijany.

Fakt, że słowo β jest *wyprowadzane* ze słowa α, zapisuje się w postaci

$$\alpha \xrightarrow{*} \beta$$

Językiem formalnym $L(G)$ generowanym przez gramatykę G jest zbiór

$$L(G) =_{\text{def}} \{\alpha \in T^* \mid S \xrightarrow{*} \alpha\}$$

Słowo $\alpha \in L(G)$ nazywa się też słowem *wywodliwym* w gramatyce G. Generowany przez gramatykę G język $L(G)$ jest zatem zbiorem wszystkich słów wywodliwych w G.

Poniżej rozpatrujemy przykłady gramatyk i wyprowadzenia słów, przy czym zapis produkcji jest oparty na powszechnie stosowanej tzw. *notacji BNF* (*Backus*[17] *Normal Form* lub *Backus-Naur Form*). Notacja ta wprowadza bardziej zwarty zapis produkcji, które mają tę samą lewą stronę. Zestaw produkcji, na przykład:

$$v ::= \alpha_1$$
$$.....$$
$$v ::= \alpha_m$$

zapisuje się w postaci

$$v ::= \alpha_1 \mid ... \mid \alpha_m$$

gdzie, jak poprzednio, $v \in N$ oraz $\alpha_1, ..., \alpha_m \in (T \cup N)^+$. Symbol | czyta się *lub*.

[17] John Backus (ur. 1924).

Przykład 7.7

Zbiór identyfikatorów tworzy pewien język formalny. Zbiór ten poprzednio był definiowany następująco:

$Ident =_{def} \{s \mid s$ *jest niepustym ciągiem składającym się z liter lub cyfr, którego pierwszym elementem jest litera*$\}$

Generująca zbiór identyfikatorów *Ident* gramatyka G_{ID} jest zdefiniowana następująco:

$G_{ID} =_{def} <T_{ID}, N_{ID}, P_{ID}, S_{ID}>$

gdzie:

$T_{ID} =_{def} \{a, b, ..., z\} \cup \{0, 1, ..., 9\}$

$N_{ID} =_{def} \{$*identyfikator, znak, litera, cyfra*$\}$

$P_{ID} =_{def} \{$*identyfikator* ::= *litera* | *identyfikator znak*
znak ::= *litera*| *cyfra*
litera ::= *a* | *b* | ... | *z*
cyfra ::= 0 | 1 | ... | 9$\}$

S_{ID} = *identyfikator*

Poszczególne produkcje w zbiorze są pisane w oddzielnych wierszach, bez oddzielania przecinkiem.

Rozpatruje się dwa przykłady wyprowadzenia konkretnych identyfikatorów. Pierwsze wyprowadzenie:

$$identyfikator \xrightarrow{identyfikator::=litera} litera$$

$$litera \xrightarrow{litera::=a} a$$

z symbolu początkowego *identyfikator* wyprowadza jednoelementowe słowo *a*.

Drugie z tego samego symbolu początkowego wyprowadza słowo *b*8:

$$identyfikator \xrightarrow{identyfikator::=identyfikator\ znak} identyfikator\ znak$$

$$identyfikator\ znak \xrightarrow{znak::=cyfra} identyfikator\ cyfra$$

$$identyfikator\ cyfra \xrightarrow{identyfikator::=litera} litera\ cyfra$$

$$litera\ cyfra \xrightarrow{litera::=b} b\ cyfra$$

$$b\ cyfra \xrightarrow{cyfra::=8} b8$$

Pokazano zatem dwa wyprowadzenia:

$$identyfikator \xrightarrow{*} a$$

$$identyfikator \xrightarrow{*} b8$$

Oznacza to, że $a \in L(G_{ID})$ oraz $b8 \in L(G_{ID})$.

Przykład 7.8

Przykład pokazuje zbiór napisów reprezentujących liczby wymierne w zapisie dziesiętnym. Gramatyka G_{DEC} jest zdefiniowana następująco:

$G_{DEC} =_{def} <T_{DEC}, N_{DEC}, P_{DEC}, S_{DEC}>$

gdzie:

$T_{DEC} =_{def} \{0, 1, ..., 9\} \cup \{.\}$

$N_{DEC} =_{def} \{liczba, liczba\_całkowita, kropka, cyfra\}$

$P_{DEC} =_{def} \{liczba ::= liczba\_całkowita \mid$
liczba_całkowita kropka liczba_całkowita
liczba_całkowita ::= *cyfra* | *liczba_całkowita cyfra*
kropka ::= .
cyfra ::= 0 | 1 | ... | 9}

$S_{DEC} = liczba$

Łatwo sprawdzić, że na przykład słowa 10.9 oraz 213 są wyprowadzalne w gramatyce G_{DEC}, natomiast słowo .01 nie jest wyprowadzalne w G_{DEC}.

Stosowane języki formalne, oprócz trywialnych przypadków, są zbiorami nieskończonymi i dlatego nie ma algorytmów generujących wszystkie słowa języka. Praktycznie rozwiązuje się dwa zadania.

Pierwsze jest zadaniem analizy – polega na zbadaniu, czy dane słowo jest elementem danego języka $L(G)$. Z tym zadaniem spotyka się podczas kompilacji programu. Celem pracy kompilatora jest w pierwszej kolejności stwierdzenie, czy program jest poprawny składniowo.

Drugie jest zadaniem generacji – polega na wygenerowaniu pewnego podzbioru słów języka, na przykład wszystkich słów o ustalonej długości.

7.5. Klasyfikacja gramatyk

Rozpatrzona gramatyka bezkontekstowa jest szczególnym przypadkiem szerszej klasy gramatyk, zwanych gramatykami struktur frazowych. *Gramatyka struktur frazowych* jest taką samą czwórką jak gramatyka bezkontekstowa, czyli

$G = <T, N, P, S>$

a różnica dotyczy tylko ogólniejszej postaci produkcji. Niech $V = T \cup N$. *Produkcja* albo *reguła przepisywania* $p \in P$ jest tu dowolną parą słów $\alpha \in V^+$ oraz $\beta \in V^*$ zapisywaną, jak poprzednio, w postaci $\alpha ::= \beta$. Generowanie języka formalnego przez gramatykę struktur frazowych jest definiowane, podobnie jak poprzednio, dla gramatyki bezkontekstowej.

Zgodnie z klasyfikacją wprowadzoną przez Chomsky'ego wyróżnia się cztery typy gramatyk struktur frazowych różniące się postacią dopuszczalnych produkcji.

Gramatyki klasy 0, zwane *gramatykami bez ograniczeń*, mają następującą postać produkcji

$\alpha ::= \beta$ dla $\alpha \in V^+, \beta \in V^*$

Gramatyki klasy 1., zwane *gramatykami kontekstowymi*, wymagają, by produkcje były postaci

$\alpha_1 v \alpha_2 ::= \alpha_1 \beta \alpha_2$ dla $\alpha_1, \alpha_2 \in V^*, v \in N, \beta \in V^+$

Gramatyki klasy 2., zwane *gramatykami bezkontekstowymi*, wymagają, by produkcje były postaci

$v ::= \beta$ dla $v \in N, \beta \in V^+$

Gramatyki klasy 3., zwane *gramatykami regularnymi*, wymagają, by produkcje były postaci (gramatyki *prawostronnie* regularne)

$v ::= \beta u$ dla $v \in N, u \in N \cup \{\varepsilon\}, \beta \in V^+$

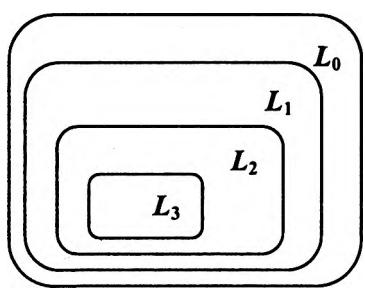
albo postaci (gramatyki *lewostronnie* regularne)

$v ::= u \beta$ dla $v \in N, u \in N \cup \{\varepsilon\}, \beta \in V^+$

Łatwo się przekonać, że każda produkcja gramatyki i jest jednocześnie produkcją gramatyki j, dla $0 \le j \le i \le 3$. Każdy zatem język formalny wygenerowany przez pewną gramatykę klasy i jest również generowany przez pewną gramatykę klasy j. Oznaczając symbolami L_0, L_1, L_2, L_3 zbiory języków formalnych generowanych przez gramatyki poszczególnych klas, można stwierdzić, że zachodzą inkluzje właściwe

$L_3 \subset L_2 \subset L_1 \subset L_0$

co oznacza, że wśród języków generowanych przez gramatyki klasy i istnieje co najmniej jeden język, który nie jest generowany przez gramatyki klasy j, dla $0 \le i \le j$ (rys. 7.1).



Rys. 7.1. Hierarchia Chomsky'ego języków formalnych

Gramatyki klas 1., 2. i 3. są *gramatykami nieskracającymi*, co oznacza, że długość nowego słowa nie jest mniejsza od długości starego słowa, do którego zastosowano produkcję gramatyki.

Nieskracalność gramatyki umożliwia efektywne badanie, czy dane słowo jest wywodliwe w gramatyce. Oznacza to, że można zbudować algorytm, który dla dowolnego słowa, po skończonej liczbie kroków, rozstrzyga, czy to słowo jest wyprowadzalne w danej gramatyce.

Schemat takiego algorytmu jest oczywisty. Niech α będzie badanym słowem. Najpierw generuje się zbiór Z_1 wszystkich słów bezpośrednio wyprowadzalnych z symbolu początkowego gramatyki, których długość nie przekracza długości badanego słowa α. Następnie generuje się zbiór Z_2 wszystkich słów, które są bezpośrednio wyprowadzalne ze słów zbioru Z_1, których długość nie przekracza długości badanego słowa α. Da-

lej generuje się zbiór słów Z_3, który jest bezpośrednio wyprowadzalny ze zbioru Z_2, itd. Każdy z wygenerowanych zbiorów jest oczywiście skończony. Postępowanie takie prowadzi się do momentu, gdy w zbiorze generowanych słów napotka się na słowo α albo gdy długość wszystkich słów należących do ostatniego zbioru będzie przekraczać długość słowa α. Pierwszy przypadek oznacza, że α należy do języka generowanego przez daną gramatykę, a drugi, że α nie należy do tego języka.

7.6. Drzewa rozbioru i diagramy składniowe

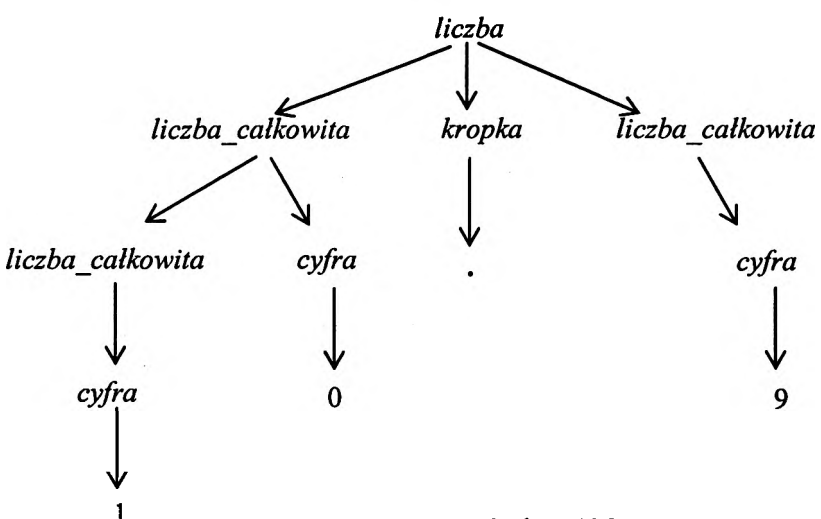
Dysponując pojęciem grafu, można zdefiniować drzewo wywodu – graf ilustrujący wyprowadzenia słowa w gramatyce, zwłaszcza w gramatyce bezkontekstowej.

Drzewem wywodu dla gramatyki $G =_{def} <T, N, P, S>$ jest graf-drzewo $T = <V, A>$, gdzie $A \subseteq V^2$, którego wierzchołki V są etykietowane symbolami ze zbioru $T \cup N$ w taki sposób, że:

- korzeń drzewa jest etykietowany symbolem początkowym S,
- każdy liść drzewa jest etykietowany symbolem terminalnym ze zbioru T,
- jeżeli węzeł v ma etykietę e i węzły $v_1, v_2, ..., v_n$ są jego następnikami, to znaczy $<v, v_1>, <v, v_2>, ..., <v, v_n> \in A$, o etykietach $e_1, e_2, ..., e_n$, to $e ::= e_1\ e_2 ... e_n$ musi być produkcją gramatyki.

Przykład 7.9

Dla przedstawionej wcześniej gramatyki $G_{DEC} =_{def} <T_{DEC}, N_{DEC}, P_{DEC}, S_{DEC}>$ drzewo wywodu dla słowa 10.9 ma postać jak na rysunku 7.1.



Rys. 7.2. Drzewo wywodu słowa 10.9

Jeżeli dla pewnego słowa istnieją dwa różne drzewa wywodu, to gramatykę nazywa się *składniowo wieloznaczną*. Gramatyka G_{DEC} jest składniowo jednoznaczna, natomiast nie jest nią poniżej zdefiniowana gramatyka G_W.

Przykład 7.10

Niech

$G_W =_{def} <T_W, N_W, P_W, S_W>$

gdzie:

T_W = {*wyrażenie*, *składnik*, *czynnik*}

N_W = {*a*, *b*, *c*, +, –, *, /, (,)}

P_W = {*wyrażenie* ::= *składnik* | *składnik* + *wyrażenie* | *składnik* – *wyrażenie*
składnik ::= *czynnik* | *czynnik* * *czynnik* | *czynnik* | *czynnik*
czynnik ::= *a* | *b* | *c* | (*wyrażenie*)}

S_W = *wyrażenie*

W celu przekonania się o niejednoznaczności gramatyki G_W wystarczy rozpatrzyć możliwe wywody, na przykład słowa *a* + *b* – *c*.

Pojęcie drzewa rozbioru stanowi między innymi podstawę do określenia równoważności gramatyk.

Dwie gramatyki G_1 oraz G_2 są *słabo równoważne*, jeżeli generują te same języki, to znaczy $L(G_1) = L(G_2)$, oraz są *silnie równoważne*, jeżeli generują te same zbiory drzew rozbiorów.

Niech $T_1 = <V_1, A_1>$, $T_2 = <V_2, A_2>$ będą drzewami rozbioru oraz niech $h : V_1 \rightarrow V_2$.

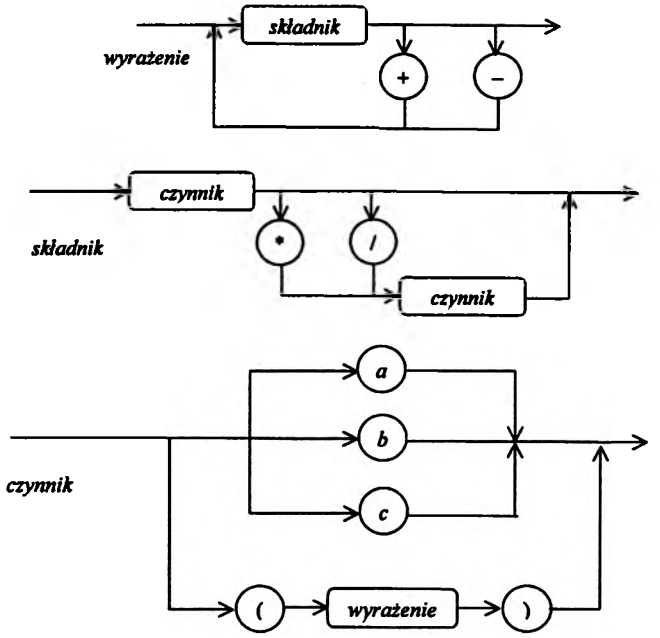
Funkcja *h zachowuje relację* wtedy i tylko wtedy, gdy dla dowolnych v, $v' \in V_1$, jeżeli $<v, v'> \in A_1^*$, to $<h(v), h(v')> \in A_2^*$, gdzie A_1^* i A_2^* są zwrotnymi, przechodnimi domknięciami relacji A_1 i A_2.

Jeżeli funkcja *h* zachowuje relację A_1 oraz dodatkowo jest bijekcją, to jest nazywana *izomorfizmem* drzewa T_1 w drzewo T_2.

Grafy są także wykorzystywane do prezentacji produkcji gramatyki w postaci diagramów składniowych. Wierzchołki tego grafu są etykietowane symbolami ze zbioru $T \cup N$. Każdej produkcji odpowiada pojedynczy graf z etykietowanymi wierzchołkami, który ma dokładnie jeden wierzchołek niemający poprzedników, zwany początkowym, i dokładnie jeden wierzchołek niemający następników, zwany końcowym. Wierzchołki te nie są etykietowane. Każdej ścieżce, która w grafie prowadzi od wierzchołka początkowego do wierzchołka końcowego, odpowiada pewien ciąg etykiet wierzchołków ze zbioru $T \cup N$. Ciąg etykiet stanowi ciąg symboli, które można wygenerować na podstawie danej produkcji.

Przykład 7.11

Produkcjom wcześniej zdefiniowanej gramatyki G_W odpowiadają następujące diagramy składniowe pokazane na rysunku 7.2.



Rys. 7.3. Diagramy składniowe gramatyki G_W

Pierwszy z diagramów, opisujący produkcję *wyrażenie*, jest grafem zawierającym cykl. Powodem pojawienia się cyklu jest to, że symbol *wyrażenie* występuje zarówno po lewej, jak i po prawej stronie produkcji, przy czym po prawej stronie występuje na końcu ciągu symboli.

Występowanie takiego samego symbolu po lewej i po prawej stronie produkcji, a tym samym istnienie cyklu na diagramie składniowym, można interpretować jako rekursywną definicję zbioru słów wyprowadzanych na podstawie danej produkcji.

7.7. Automaty i gramatyki

Gramatyki są mechanizmem generowania języków formalnych. Z gramatykami ściśle się wiąże pojęcie automatów skończonych jako mechanizmu służącego do rozpoznawania, czy dane słowa należą do danego języka formalnego. Przedstawiona dalej definicja automatu skończonego prezentuje tylko pewną klasę automatów skończonych,

służących do rozpoznawania słów należących do języków klas L_3 oraz L_2. Są to automaty Rabina–Scotta (*RS*) oraz automaty ze stosem.

Automat skończony jest modelem urządzenia, którego zadaniem jest rozpoznawanie, czy słowa podawane na jego wejście są słowami danego języka formalnego. Automat działa krokowo; kolejne kroki są związane z analizą kolejnej litery słowa podawanego na jego wejście. Automat w każdym kroku swego działania jest w określonym stanie. Działanie rozpoczyna w ustalonym stanie początkowym, a kończy, gdy zostanie przeczytane całe badane słowo. Podczas kolejnych kroków automat przechodzi pomiędzy stanami, co następuje na skutek odczytania na wejściu kolejnej litery analizowanego słowa. Po zakończenie działania, to znaczy po odczytaniu ostatniej litery analizowanego słowa, automat znajdzie się w pewnym stanie. Jeżeli jest to jeden z jego stanów końcowych, oznacza to, że słowo należy do języka formalnego akceptowanego (rozpoznawanego) przez automat.

Formalnie skończony *automat akceptujący RS* jest określony jako piątka

$$A = <S, X, \delta, s_0, F>$$

gdzie:

S jest skończonym zbiorem stanów,
X jest alfabetem – zbiorem symboli wejściowych,
$\delta: S \times X \rightarrow S$ jest funkcją zmiany stanów,
s_0 jest stanem początkowym,
$F \subseteq S$ jest zbiorem stanów końcowych.

Działanie automatu przy analizie słowa wejściowego $x_1, x_2, ..., x_n$, dla $n > 0$, polega na wykonywaniu ciągu kroków, które określają ciąg stanów:

$$s_0, s_1, ..., s_n$$

taki, że

$$s_{k+1} = \delta(s_k, x_k) \qquad \text{dla } k = 1, ..., n$$

Słowo $x_1, x_2, ..., x_n$ jest akceptowane przez automat, gdy $s_k \in F$, i nie jest akceptowane w przypadku przeciwnym. Zbiór wszystkich akceptowanych słów, oznaczany $L(RS)$, stanowi język formalny akceptowany przez automat *RS*.

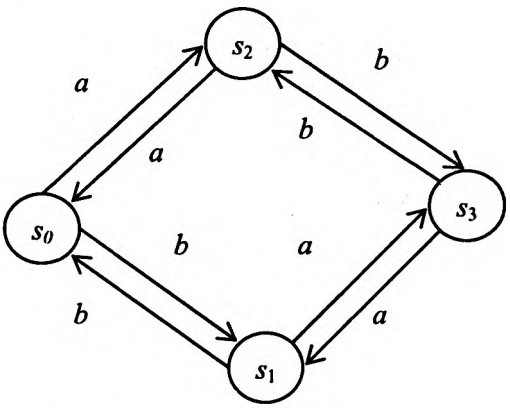
Przykład 7.12

Automat *RS* postaci $<S, X, \delta, s_0, F>$, gdzie:

$S = \{s_0, s_1, s_2, s_3\}$
$X = \{a, b\}$
$F = \{s_0\}$
$\delta(s_0, a) = s_2$, $\delta(s_0, b) = s_1$
$\delta(s_1, a) = s_3$, $\delta(s_1, b) = s_0$

$\delta(s_2, a) = s_0$, $\delta(s_2, b) = s_3$
$\delta(s_3, a) = s_1$, $\delta(s_3, b) = s_2$

można przedstawić graficznie w postaci pokazanej na rysunku 7.4.



Rys. 7.4. Graf przejść stanów automatu

Można też łatwo sprawdzić, że językiem formalnym akceptowanym przez automat jest $L \subseteq \{a, b\}^*$ taki, że

$L = \{\alpha |$ $len(\alpha|_a)$ jest parzysta oraz $len(\alpha|_b)$ jest parzysta$\}$,

gdzie: $len(\alpha)$ oznacza długość ciągu α, a $\alpha|_a$ oraz $\alpha|_b$ oznaczają podciągi ciągu α złożone wyłącznie z symboli a oraz b.

Pomiędzy skończonymi automatami *RS* a gramatykami zachodzą następujące związki:

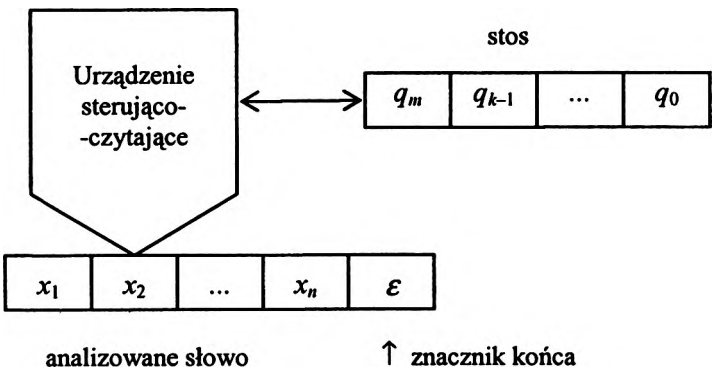
- Dla każdego języka formalnego klasy $\boldsymbol{L_3}$ istnieje skończony automat *RS* akceptujący ten język, to znaczy każde skończone obliczenie automatu kończy się osiągnięciem stanu końcowego.
- Każdy język akceptowany przez skończony automat *RS* jest językiem klasy $\boldsymbol{L_3}$.

Języki klasy $\boldsymbol{L_3}$ nazywa się językami *regularnymi*.

Rozpoznawanie języków klasy $\boldsymbol{L_2}$, czyli języków *bezkontekstowych*, może być również realizowane przez automaty, z tym że są to bardziej złożone automaty, nazywane automatami ze stosem (rys. 7.5).

Stos jest pewnego rodzaju pamięcią, w której przechowuje się ciągi symboli z danego repertuaru symboli. Na stosie można wykonywać dwie zasadnicze operacje: dopisywania symbolu do stosu lub odczytywania i zdjęcia elementu ze stosu. Jeżeli zawartością stosu jest ciąg symboli $q_1\ q_2\ \ldots\ q_k$, to dopisanie symbolu q' polega na dołączeniu go na początku ciągu, czyli zmieni jego zawartość na $q'q_1\ q_2\ \ldots\ q_k$, zdjęcie natomiast

elementu ze stosu polega na usunięciu elementu na początku ciągu, czyli zmieni jego zawartość na $q_2 \ldots q_k$.



Rys. 7.5. Automat ze stosem

Automat ze stosem AS jest definiowany jako siódemka

$$AS = <S, X, Q, \delta, s_0, q_0, F>$$

gdzie:

S jest skończonym zbiorem stanów,

X jest alfabetem – zbiorem symboli wejściowych,

Q jest skończonym zbiorem symboli składowanych na stosie,

$\delta: S \times (X \cup \{\varepsilon\}) \times Q \rightarrow \wp_{fin}(S \times Q^*)$
jest funkcją zmiany stanu i zawartości stosu ($\wp_{fin}(Z)$ oznacza rodzinę wszystkich skończonych podzbiorów zbioru Z),

s_0 jest stanem początkowym,

q_0 jest symbolem początkowym znajdującym się na stosie,

$F \subseteq S$ jest zbiorem stanów końcowych.

Jeżeli zbiór $\delta(s, x, q)$ dla każdego $s \in S$, $x \in X$, $q \in Q$ zawiera co najwyżej jeden element, to automat AS jest automatem deterministycznym, w przypadku przeciwnym – jest automatem niedeterministycznym.

Działanie automatu AS przebiega krokowo, a w każdym kroku następuje zmiana konfiguracji automatu. Konfiguracjami automatu są trójki:

$$<s, \omega, \alpha> \in S \times X^* \times Q^*$$

gdzie:

s jest aktualnym stanem urządzenia sterująco-czytającego,

ω jest częścią analizowanego słowa, nieprzeczytaną jeszcze przez głowicę czytającą; pierwszy symbol ciągu ω znajduje się pod głowicą czytającą; jeżeli sym-

bolem tym jest ε (słowo puste) oznacza to, że całe słowo zostało już przeczytane,

α jest aktualną zawartością stosu.

Krok działania automatu AS polega na przejściu z konfiguracji do konfiguracji w wyniku przeczytania pojedynczego symbolu analizowanego słowa. Przejście takie będzie opisywane relacją $\longrightarrow$ i zapisywane w postaci

$$<s, x\omega, q\alpha> \longrightarrow <s', \omega, \beta\alpha>$$

jeżeli zbiór $\delta(s, x, q)$ zawiera parę $<s', \beta>$, gdzie $s, s' \in S$, $x \in X \cup \{\varepsilon\}$, $\omega \in X^*$, $q \in Q$ oraz $\alpha, \beta \in Q^*$.

Jeżeli $x \neq \varepsilon$, to automat AS jest w stanie s, x jest symbolem znajdującym się pod głowicą czytającą, q jest symbolem będącym się na szczycie stosu. Automat przechodzi do nowego stanu s', przesuwa głowicę czytającą o jedną pozycję w prawo i zamienia symbol na szczycie stosu ciągiem β złożonym z elementów zbioru symboli Q. Jeżeli $\beta = \varepsilon$, to usuwa się element ze szczytu stosu, skracając tym samym jego zawartość.

Jeżeli $x = \varepsilon$, to oznacza, że całe słowo zostało przeczytane. W kroku tym, nazywanym ε-krokiem, automat AS nie przesuwa głowicy czytającej, jednak stan automatu i zawartość stosu mogą się zmieniać.

Początkową konfiguracją automatu AS jest $<s_0, \omega, q_0>$, co oznacza, że automat znajduje się w stanie początkowym $s_0 \in S$, pod głowicą czytającą znajduje się pierwszy symbol analizowanego słowa $\omega \in X^*$, a zawartością stosu jest tylko jeden początkowy symbol $q_0 \in Q$. Konfiguracją końcową jest konfiguracja postaci $<s, \varepsilon, \varepsilon>$, gdzie $s \in F$.

Słowo jest akceptowane przez automat AS, jeżeli

$$<s_0, \omega, q_0> \overset{*}{\longrightarrow} <s, \varepsilon, \varepsilon>,$$

gdzie $\overset{*}{\longrightarrow}$ jest zwrotnym i przechodnim domknięciem relacji $\longrightarrow$.

Zbiór wszystkich akceptowanych słów, oznaczany $L(AS)$, stanowi język formalny akceptowany przez automat AS.

Przykład 7.13

Automat AS postaci $<S, X, Q, \delta, s_0, q_0, F>$, gdzie:

$S = \{s_0, s_1, s_2\}$
$X = \{0, 1\}$
$Q = \{q_0, 0\}$
$F = \{s_0\}$
$\delta(s_0, 0, q_0) = \{<s_1, 0q_0>\}$

$\delta(s_1, 0, 0) = \{<s_1, 00>\}$
$\delta(s_1, 1, 0) = \{<s_2, \varepsilon>\}$
$\delta(s_2, 1, 0) = \{<s_2, \varepsilon>\}$
$\delta(s_2, \varepsilon, q_0) = \{<s_0, \varepsilon>\}$

Zilustrujmy działanie automatu podczas analizy konkretnego słowa 0011:

$$<s_0, 0011, q_0> \longrightarrow <s_1, 011, 0q_0>$$
$$\longrightarrow <s_1, 11, 00q_0>$$
$$\longrightarrow <s_2, 1, 0q_0>$$
$$\longrightarrow <s_2, \varepsilon, q_0>$$
$$\longrightarrow <s_0, \varepsilon, \varepsilon>$$

Ogólnie, można pokazać, że:

$$<s_0, 0, q_0> \longrightarrow <s_1, \varepsilon, 0q_0>$$
$$<s_1, 0^i, 0q_0> \xrightarrow{i} <s_1, \varepsilon, 0^{i+1}q_0>$$
$$<s_1, 1, 0^{i+1}q_0> \longrightarrow <s_2, \varepsilon, 0^i q_0>$$
$$<s_2, 1^i, 0^i q_0> \xrightarrow{i} <s_2, \varepsilon, q_0>$$
$$<s_2, \varepsilon, q_0> \longrightarrow <s_0, \varepsilon, \varepsilon>$$

Na tej podstawie można pokazać, że dla $n \geq 1$ zachodzi

$$<s_0, 0^n1^n, q_0> \xrightarrow{2n+1} <s_0, \varepsilon, \varepsilon>$$

oraz

$$<s_0, \varepsilon, q_0> \xrightarrow{0} <s_0, \varepsilon, q_0>$$

co oznacza, że zbiór akceptowanych słów zawiera się w języku $L = \{0^n1^n \mid n \geq 0\}$. Pomijamy tu pokazanie faktu, że automat akceptuje słowa tylko z tego języka.

Rozpoznawanie języków klas L_1 oraz L_0 jest jeszcze bardziej złożone. Pomijając szczegóły, ograniczymy się tu do stwierdzenia, że do rozpoznawania języka dowolnej klasy, a więc również klasy L_0, można zbudować odpowiednią maszynę Turinga.

Uwaga

Bardziej szczegółowe informacje o językach formalnych, gramatykach i automatach skończonych można znaleźć na przykład w książce [Hopcroft, Ullman 2003].

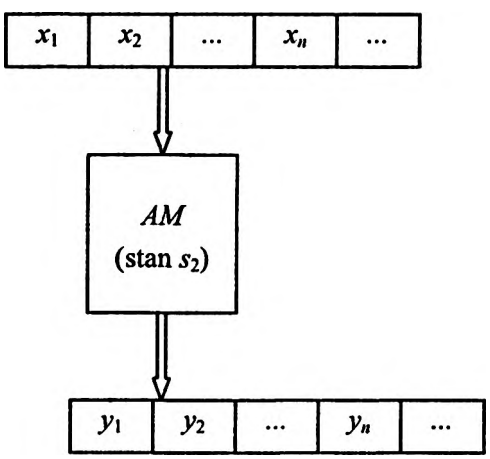
Pojęcie automatu skończonego jest często używane w różnych obszarach informatyki, zwłaszcza w projektowaniu układów cyfrowych z pamięcią (w odróżnieniu od układów cyfrowych bez pamięci – bramek logicznych, zob. rozdz. 9.). Najczęś-

ciej spotykanymi tam pojęciami są skończone automaty Moore'a i automaty Mealy'ego. Oba automaty są modelami „czarnej skrzynki" z jednym wejściem i jednym wyjściem (rys. 7.6). Na podstawie ciągu symboli czytanych na wejściu automat podaje na swoje wyjście inny ciąg symboli; ciąg wyjściowy jest wynikiem przetworzenia ciągu wejściowego. Ponieważ automaty te można stosować zamiennie, poniżej podaje się tylko definicję automatu Moore'a *AM*:

$AM = \langle X, Y, S, \delta, \lambda, s_0 \rangle$

gdzie:

X jest skończonym zbiorem symboli wejściowych,
Y jest skończonym zbiorem symboli wejściowych,
S jest skończonym zbiorem stanów,
$\delta: X \times S \rightarrow S$ jest funkcją przejść,
$\lambda: S \rightarrow Y$ jest funkcją wyjść,
s_0 jest stanem początkowym.



Rys. 7.6. Schemat automatu Moore'a

Automat *AM* przetwarza ciągi $x_1, x_2, ..., x_n, ...$ symboli alfabetu wejściowego X w ciągi $y_1, y_2, ..., y_n, ...$ symboli alfabetu Y. Przetworzenie to jest określone następująco: Niech

$s_0, s_1, s_2, ..., s_n, ...$

będzie ciągiem stanów takim, że

$s_{k+1} = \delta(x_k, s_k)$, dla $k = 1, 2, ..., n, ...$

wówczas $y_k = \lambda(s_k)$.

Ćwiczenia

1. Niech $A =_{def} \{+, =\}$. Które z poniższych zdań są prawdziwe?
   a) Można utworzyć co najwyżej skończoną liczbę języków formalnych nad alfabetem A.
   b) Można utworzyć dokładnie 4 słowa nad alfabetem A.
   c) Zbiór { ++, +++, =, += } jest pewnym językiem formalnym nad A.
   d) Nad alfabetem A można utworzyć dokładnie 2^4 języków formalnych.
   e) Zbiór wszystkich słów nad alfabetem A definiuje pewien nowy alfabet A'.
2. Niech $A =_{def} \{0, 1\}$, $B =_{def} \{0, 1, 2, ..., n\}$ oraz $C = Nat$. Które pary spośród zbiorów A^*, B^*, C^* są zbiorami równolicznymi?
3. Czy zbiór liczb naturalnych Nat jest równoliczny ze zbiorem Nat^* – zbiorem wszystkich skończonych ciągów nad Nat?
4. Czy zbiór wszystkich języków formalnych nad przeliczalnym alfabetem A jest przeliczalny?
5. Niech L będzie językiem formalnym nad alfabetem $\{0, 1\}$. Dwa słowa $u, v \in \{0, 1\}^*$ są równoważne względem języka L, co oznacza się $u \sim_L v$, jeżeli

   $$\forall x \in \{0, 1\}^* \bullet (u \wedge x \in L \Leftrightarrow v \wedge x \in L)$$

   Pokazać, że $\sim_L$ jest relacją równoważności.
6. Przedstawić klasy abstrakcji relacji równoważności słów $\sim_L$ względem następujących języków:
   a) $L_1 = \{1^n \mid 1 \leq n \leq 6\}$
   b) $L_2 = \{0^n 1^n \mid n \in Nat\}$
   c) $L_3 = (0011)$
7. Pokazać, że jeśli dla pewnych słów $u \in L$ oraz $v \in \{0, 1\}^*$ zachodzi $u \wedge v \in [u]_{\sim L}$, gdzie $\sim_L$ jest relacją równoważności słów względem języka L, to $u \wedge v^n \in L$ dla dowolnego $n \in Nat$.
8. Dana jest gramatyka $G = <T, N, P, S>$, gdzie:

   $T =_{def} \{A, B, C\}$
   $N =_{def} \{a, b, c\}$
   $P =_{def} \{a ::= A \mid aA \mid bC$
   $\quad b ::= BcC$
   $\quad c ::= abC \mid ABc \mid AbC \}$
   $S = a$

Czy słowa *AAAA*, *ABCA* należą do języka generowanego przez *G*? Podać zbiór wszystkich słów długości 1, 2 i 3 należących do języka generowanego przez *G*. Scharakteryzować zbiór wszystkich słów generowanych przez gramatykę *G*. Czy można zdefiniować „prostszą" gramatykę, która generuje taki sam język formalny jak gramatyka *G*?

9. Przedstawić drzewa rozbioru składniowego dla wszystkich słów długości 4 generowanych przez gramatykę z zadania 8.

10. Czy jednoznaczne są gramatyki:
    a) gramatyka z zadania 8.,
    b) $G = <\{a, +, *\}, \{S\}, \{S ::= S + S \mid S * S \mid a\}, S>$.

11. Zdefiniować gramatykę generującą język $L = \{a^n b^m c^m \mid m, n \geq 1\}$. Zbadać, czy gramatyka jest jednoznaczna.

12. Zdefiniować gramatykę generującą następujące zbiory:
    a) zbiór wszystkich palindromów (słów, które można odczytywać zarówno w przód, jak i wspak) nad alfabetem $\{a, b\}$,
    b) zbiór wszystkich słów nad alfabetem $\{a, b\}$ zawierających dokładnie dwa razy więcej symboli a niż symboli b,
    c) $L = \{a^i b^j c^k \mid i \neq j \text{ lub } j \neq k\}$.

13. Dana jest gramatyka $G = <\{a, b\}, \{S\}, \{S ::= aS \mid aSbS \mid \varepsilon\}, S>$. Udowodnić, że $L(G) = \{x \mid$ każdy przedrostek x ma co najmniej tyle symboli a, co symboli $b\}$

14. Dla znanego języka programowania, na przykład Pascal, C, C++, zdefiniować gramatykę określającą wybrany podzbiór wyrażeń arytmetycznych z tego języka.

15. Przedstawić w postaci diagramów składniowych produkcje gramatyk zdefiniowanych w zadaniach 5., 7. i 8.

16. Symbol nieterminalny $A \in N$ gramatyki jest zbędny, gdy jest nieaktywny lub nieosiągalny. Symbol $A \in N$ jest nieaktywny, gdy język generowany przez gramatykę $G_A = <T, N, P, A>$ jest pusty. $A \in N$ jest nieosiągalny, gdy nie występuje w żadnym słowie wyprowadzanym w gramatyce G. Wskazać zbędne symbole nieterminalne w gramatyce $G = <T, N, P, S>$, gdzie:

$$
\begin{aligned}
T &=_{\text{def}} \{A, B, C\} \\
N &=_{\text{def}} \{a, b, c, d\} \\
P &=_{\text{def}} \{a ::= A \mid aA \mid bC \mid AcA \\
&\qquad b ::= BcC \mid cAc \\
&\qquad c ::= abC \mid ABc \mid AbC \\
&\qquad d ::= aA \mid dbC\} \\
S &= a
\end{aligned}
$$

17. Gramatykę $G = <T, N, P, S>$, gdzie:

$T =_{def} \{A, B\}$
$N =_{def} \{a, b, c\}$
$P =_{def} \{c ::= a \mid b$
$a ::= Ab \mid Bc \mid B$
$b ::= ab \mid bA \mid ac \mid B\}$
$S = c$

przekształcić do słabo równoważnej gramatyki bezkontekstowej, niezawierającej zbędnych symboli.

18. Zdefiniować automat z pamięcią stosową, rozpoznający słowa języka

$L = \{\alpha\alpha^{-1} \mid \alpha \in \{a, b\}^+\}$

8. Algebry abstrakcyjne

8.1. Algebry jednorodzajowe

Jednorodzajową algebrą abstrakcyjną, albo krótko – algebrą, nazywa się parę

$ALG =_{def} <A, \{c_1, ..., c_m\} \cup \{f_1, ..., f_n\}>$ dla $m \in Nat, n \in Nat \setminus \{0\}$

w której:

A jest dowolnym zbiorem, zwanym *nośnikiem* algebry,

c_i są *stałymi* algebry, to znaczy $c_i \in A$, dla $i = 1, ..., m$,

f_j są *operacjami* albo *działaniami* algebry, to znaczy k-argumentowymi funkcjami o sygnaturze

$$f_j: A^k \to A$$

gdzie: $k \in Nat \setminus \{0\}, j = 1, ..., n$.

Stałą algebry c_i można także rozumieć jako funkcję zeroargumentową, to znaczy jako funkcję o sygnaturze

$c_i : \to A$.

Uwaga

Algebry będą też definiowane jako pary:

$$ALG =_{def} <A, \{c_1, ..., c_m, f_1, ..., f_n\}>$$

lub

$$ALG =_{def} <A, F>$$

gdzie drugim elementem pary jest zbiór stanowiący sumę mnogościową zbioru stałych i operacji. Wynika to z faktu, że stałe można traktować jako funkcje zeroargumentowe.

Przykład 8.1

Przykładem prostej algebry jest zbiór liczb naturalnych *Nat* z operacją dodawania

$$ALG_{Nat} =_{def} <Nat, \{0, 1\} \cup \{\_+\_\}>$$

gdzie:

$0, 1 : \to Nat$

są operacjami zeroargumentowymi, a

$$\_+\_ : Nat^2 \rightarrow Nat$$

jest dodawaniem.

Należy przypomnieć, że podkreślenia obok symbolu funkcji wskazują położenie argumentów.

Przykład 8.2

Bardziej złożona jest algebra określona na zbiorze liczb całkowitych *Całkowite*, z operacjami dodawania, odejmowania i mnożenia

$$ALG_{Całkowite} =_{def} <Całkowite, \{0, 1\} \cup \{ \_+\_, \_-\_, \_*\_ \}>$$

gdzie: 0 oraz 1 są stałymi, czyli mają sygnatury:

$$0, 1 : \rightarrow Całkowite$$

a +, –, * są symbolami operacji dwuargumentowych o sygnaturach:

$$\_+\_, \_-\_, * : Całkowite^2 \rightarrow Całkowite$$

Należy zauważyć, że odejmowanie może być również traktowane jako zmiana znaku liczby. W tym przypadku symbol byłby symbolem przeciążonym, a odpowiadająca mu sygnatura miałaby postać

$$-\_ : Całkowite \rightarrow Całkowite$$

Algebra z tak dołączoną operacją miałaby natomiast postać

$$ALG_{Całkowite} =_{def} < Całkowite, \{0, 1\} \cup \{-\_, \_+\_, \_-\_, \_*\_ \}>$$

W dalszych przykładach dla symboli funkcyjnych dwuargumentowych będzie stosowana notacja wrostkowa, a podkreślenia w napisach określających sygnaturę będą pomijane.

Przykład 8.3

Algebra słów nad pewnym alfabetem A jest zdefiniowana

$$ALG_{A^*} =_{def} <A^*, \{\varepsilon\} \cup \{\hat{}\} >$$

gdzie:

A^* jest nośnikiem algebry,

$\varepsilon : \rightarrow A^*$ jest słowem pustym, czyli stałą algebry,

$\hat{} : A^* \times A^* \rightarrow A^*$ jest konkatenacją, czyli dwuargumentowym działaniem.

Przykład 8.4

W programowaniu przez typ danych rozumie się pewien zbiór wartości i zestaw związanych z nim operacji. Powszechnie używany typ logiczny jest określony przez zbiór

$$Boolean =_{def} \{false, true\}$$

oraz przez zestaw operacji o następujących sygnaturach:

$$not : Boolean \rightarrow Boolean$$

$$and, or : Boolean^2 \rightarrow Boolean$$

gdzie: *not*, *and* oraz *or* są operacjami *negacji*, *koniunkcji* i *dysjunkcji*. Definicję tych funkcji przedstawia tablica:

a	*b*	*not(a)*	*a and b*	*a or b*
false	*false*	*true*	*false*	*false*
false	*true*	*true*	*false*	*true*
true	*false*	*false*	*false*	*true*
true	*true*	*false*	*true*	*true*

Zdefiniowane funkcje warto porównać z definicją spójników logicznych określonych w podrozdziale 1.2. Różnica między tymi definicjami sprowadza się do różnicy symboli.

Algebra, która jest modelem typu logicznego, ma zatem postać

$$ALG_{Boolean} =_{def} <Boolean, \{not, and, or\}>$$

Zbiór stałych jest tutaj pusty.

Przykład 8.5

W językach programowania odpowiednikiem wcześniej przedstawianej algebry, określonej na zbiorze liczb całkowitych *Całkowite*, jest algebra określona na zbiorze *Integer* z odpowiednikami operacji dodawania, odejmowania i mnożenia:

$$ALG_{Integer} =_{def} < Integer, \{0, 1\} \cup \{\oplus, \ominus, \otimes\}>$$

gdzie:

$Integer =_{def} \{-N, ..., N\}$ jest tylko skończonym podzbiorem zbioru *Całkowite*, zakłada się przy tym, że $N \in Nat \setminus \{0, 1\}$,

0 oraz 1 są stałymi o sygnaturach:

$$0, 1 : \rightarrow Integer$$

a $\oplus,\ominus,\otimes$ są – odpowiednio – symbolami dodawania, odejmowania i mnożenia o sygnaturach:

$$\oplus,\ominus,\otimes: Integer^2 \rightarrow Integer$$

Istotna różnica między algebrą $ALG_{Integer}$ a $ALG_{Całkowite}$ wypływa z definicji operacji w algebrze $ALG_{Integer}$. Ze względu mianowicie na ograniczoność zbioru *Integer* operacje dodawania, odejmowania i mnożenia nie są funkcjami całkowicie określonymi. Aby odróżnić je od operacji określonych w zbiorze liczb *Całkowite*, są one zapisywane inaczej. Definicje operacji algebry $ALG_{Integer}$ przedstawiają się następująco:

$$a \oplus b =_{def} \begin{cases} a+b & \text{gdy } |a+b| \leq N \\ \text{nieokreślona w przypadku przeciwnym} \end{cases}$$

$$a \ominus b =_{def} \begin{cases} a-b & \text{gdy } |a-b| \leq N \\ \text{nieokreślona w przypadku przeciwnym} \end{cases}$$

$$a \otimes b =_{def} \begin{cases} a*b & \text{gdy } |a*b| \leq N \\ \text{nieokreślona w przypadku przeciwnym} \end{cases}$$

Symbole występujące po prawej stronie definicji, czyli funkcje dodawania, odejmowania, mnożenia i wartości bezwzględnej, są określone na zbiorze liczb całkowitych i należą do języka arytmetyki. Bez znajomości tych funkcji nie można zrozumieć definicji nowych operacji.

Częściowa określoność operacji algebry $ALG_{Integer}$ ma interpretację praktyczną. Brak określoności operacji oznacza możliwość powstania nadmiaru podczas wykonywania programu. Powstanie nadmiaru podczas obliczenia programu prowadzi do wygenerowania wyjątku lub do zerwania obliczeń z sygnalizacją błędu.

Algebry, które mają operacje określone częściowo, mogą być uciążliwe w zastosowaniach, dlatego – zwłaszcza w programowaniu – dokonuje się pewnej modyfikacji takich algebr tak, aby uzyskać całkowitą określoność ich operacji. Sposób tej modyfikacji wyjaśnia przykład algebry $ALG_{Integer}$.

Definiowaną w przykładzie algebrę $ALG_{Integer \cup \{nadmiar\}}$ można traktować jako algebraiczny model całkowitoliczbowego typu danych występującego w językach programowania.

Przykład 8.6

Niech dana będzie algebra

$$ALG_{Integer \cup \{nadmiar\}} =_{def} <Integer \cup \{nadmiar\},\ \{0, 1\} \cup \{\oplus,\ominus,\otimes\}>$$

Nośnikiem algebry jest zbiór *Integer* z dołączonym elementem *nadmiar*. Operacjami algebry są operacje dodawania, odejmowania i mnożenia, oznaczane – jak

poprzednio – symbolami: $\oplus,\ominus,\otimes$. Operacje te mają inne sygnatury:

$$\oplus,\ominus,\otimes : (Integer \cup \{nadmiar\})^2 \rightarrow Integer \cup \{nadmiar\}$$

Wszystkie operacje arytmetyczne mają wspólną własność:

Jeżeli wartością któregokolwiek argumentu operacji jest *nadmiar*, to wynikiem operacji jest również *nadmiar*, na przykład $nadmiar \oplus 1 = nadmiar$.

W pozostałych przypadkach, gdy oba argumenty operacji są różne od *nadmiar*, definicje operacji są następujące:

$$a \oplus b =_{def} \begin{cases} a+b & \text{gdy } |a+b| \leq N \\ nadmiar \text{ w przypadku przeciwnym} \end{cases}$$

$$a \ominus b =_{def} \begin{cases} a-b & \text{gdy } |a-b| \leq N \\ nadmiar \text{ w przypadku przeciwnym} \end{cases}$$

$$a \otimes b =_{def} \begin{cases} a*b & \text{gdy } |a*b| \leq N \\ nadmiar \text{ w przypadku przeciwnym} \end{cases}$$

Uwaga

Symbol *nadmiar* w powyższym przykładzie jest odpowiednikiem symbolu $\perp$, wprowadzonego w podrozdziale 3.7 na oznaczenie nieokreśloności funkcji. Symbol *nadmiar* odnosi się również do sytuacji, gdy funkcja jest nieokreślona, a ponadto wskazuje na przyczynę nieokreśloności.

8.2. Algebry wielorodzajowe

Uogólnieniem algebr jednorodzajowych są *algebry wielorodzajowe* [Ehrig, Mahr 1985]. Uogólnienie polega na zastąpieniu pojedynczego nośnika skończoną rodziną nośników. Algebra wielorodzajowa jest zdefiniowana jako układ

$$ALG =_{def} <\{A_1, ..., A_k\}, \{c_1, ..., c_m\} \cup \{f_1, ..., f_n\}> \qquad \text{dla } m \in Nat \text{ oraz } k, n \in Nat \setminus \{0\}$$

gdzie:

$A_1, ..., A_k$ są dowolnymi zbiorami, nazywanymi *nośnikami* algebry,

c_i jest *stałą* algebry, to znaczy $c_i \in A_{j_i}$, dla $i = 1, ..., m, j_i \in \{1, ..., k\}$,

f_j jest *operacją* algebry, dla $j = 1, ..., n$, to znaczy jest k_j-argumentową funkcją, $k_j \in Nat \setminus \{0\}$, o sygnaturze:

$$f_j : A_{j_1} \times ... \times A_{j_{k_j}} \rightarrow A_{j_{k_0}}$$

gdzie:

$j_1, ..., j_{k_j} \in \{1, ..., k\}$ są indeksami nośników, które są argumentami operacji,

j_{k_0} jest indeksem nośnika, który jest wynikiem operacji.

Uwaga

Algebra wielorodzajowa będzie też oznaczana krócej

$$ALG =_{def} <A, F>$$

gdzie: A jest zbiorem nośników, a F jest zbiorem operacji, w tym operacji zeroargumentowych, czyli stałych.

Przykład 8.7

Rozpatruje się dwurodzajową algebrę określoną na liczbach całkowitych, która – oprócz operacji arytmetycznych: dodawania, odejmowania, mnożenia, dzielenia całkowitoliczbowego – obejmuje również operacje porównywania liczb: równy, nie mniejszy. Argumentami zarówno działań arytmetycznych, jak i operatorów porównania są liczby, wynikami działań są także liczby, wynikami zaś porównań są wartości logiczne. Tym samym wprowadzenie operacji porównań wprowadza niejawnie dodatkowy nośnik zawierający wartości logiczne. Może nim być na przykład zbiór

$$Boolean =_{def} \{false, true\}$$

Algebrę można przedstawić jako algebrę dwurodzajową

$$ALG_{Całkowite} =_{def}$$

$$<\{Całkowite, Boolean\}, \{0,1\} \cup \{-, +, -, *, /, =, \geq\}>$$

gdzie:

0, 1 są stałymi liczbowymi, *zerem* i *jedynką*,

$-$: $Całkowite \rightarrow Całkowite$ jest jednoargumentową operacją *zmiany znaku* liczby,

$+, -, *, /$: $Całkowite^2 \rightarrow Całkowite$ są dwuargumentowymi operacjami dodawania, odejmowania, mnożenia i dzielenia,

$=, \geq$: $Całkowite^2 \rightarrow Boolean$ są dwuargumentowymi operacjami porównań równy i nie mniejszy.

Algebry wielorodzajowe mogą być modelem złożonych typów danych.

Algebra $ALG_{Całkowite}$ może być potraktowana jako pewna charakterystyka całkowitoliczbowego typu danych spotykanego w językach programowania. Charakterystyka ta nie uwzględnia ograniczoności zbioru wartości typu. Pełną charakterystyką typu jest algebra przedstawiona w przykładzie 8.8.

Przykład 8.8

Typ całkowitoliczbowy na zbiorze

$$Integer =_{def} \{-N, ..., 0, ..., N\}$$

ma określone operacje arytmetyczne: zmiany znaku –, dodawania ⊕, odejmowania ⊖, mnożenia ⊗, dzielenia całkowitoliczbowego ⊘, oraz ma operacje porównywania liczb: równy =, nie mniejszy ≥, których wartościami są elementy zbioru

$$Boolean =_{def} \{false, true\}$$

Pełny model typu całkowitoliczbowego można przedstawić jako algebrę dwurodzajową

$$ALG_{Integer \cup \{nadmiar\}} =_{def} <\{Integer \cup \{nadmiar\}, Boolean\}, \{0,1\} \cup$$

$$\{-, \oplus, \ominus, \otimes, \oslash, =, \geq\}>$$

Stałe i operacje algebry mają sygnatury:

$0, 1 : \rightarrow Integer$
$- : Integer \rightarrow Integer$
$\oplus, \ominus, \otimes : (Integer \cup \{nadmiar\})^2 \rightarrow Integer \cup \{nadmiar\}$
$\oslash : (Integer \cup \{nadmiar\})^2 \rightarrow Integer \cup \{nadmiar\}$
$=, \geq : Integer^2 \rightarrow Boolean$

Oprócz operacji dzielenia, pozostałe operacje definiuje się podobnie, jak w przykładzie 8.6. Definicja operacji dzielenia przedstawia się następująco:

$$a \oslash b =_{def} \begin{cases} a/b & \text{gdy } a \in Integer,\ b \in Integer \setminus \{0\} \text{ oraz } |a/b| \leq N \\ nadmiar & \text{gdy } a \in Integer,\ b \in Integer \setminus \{0\} \text{ oraz } |a/b| > N \\ nadmiar & \text{gdy } b = 0 \text{ lub } a = nadmiar \text{ lub } b = nadmiar \end{cases}$$

Operacje porównań są obcięciem funkcji porównań = oraz ≥, określonych na zbiorze liczb całkowitych *Całkowite*, do zbioru *Integer*.

8.3. Termy

Z każdą algebrą jest związany pewien zbiór napisów, które powstają ze złożenia symboli stałych, zmiennych i działań algebry. Zbiór ten nazywa się zbiorem termów. Poniżej przedstawia się definicję termów, najpierw dla algebr jednorodzajowych, a następnie dla wielorodzajowych [Ehrig, Mahr 1985]. Niech

$$ALG =_{def} <A, \{c_1, ..., c_m\} \cup \{f_1, ..., f_n\}>$$

będzie pewną algebrą jednorodzajową oraz niech V będzie zbiorem zmiennych, to znaczy symboli, którym można przyporządkowywać pewne wartości z dziedziny A.

Symbolem $Term_{ALG}(V)$ oznacza się *zbiór termów* algebry ALG nad zbiorem zmiennych V. Zbiór ten jest zdefiniowany rekursywnie w sposób następujący:

- $V \subseteq Term_{ALG}(V)$ oraz $\{c_1, ..., c_m\} \subseteq Term_{ALG}(V)$, to znaczy, że zmienne i stałe są termami,
- jeżeli $t_1, ..., t_k \in Term_{ALG}(V)$, czyli napisy $t_1, ..., t_k$ są termami oraz f_j jest działaniem k-argumentowym, to $f_j(t_1, ..., t_n) \in Term_{ALG}(V)$, czyli napis postaci $f_j(t_1, ..., t_n)$ jest termem.

Uwaga

Jeżeli $t_1, t_2 \in Term_{ALG}(V)$ oraz f_j jest działaniem dwuargumentowym zapisywanym w konwencji wrostkowej, to za term będzie przyjmowany napis $(t_1 f_j t_2) \in Term_{ALG}(V)$.

Termy są słowami nad pewnym alfabetem wyznaczonym przez daną algebrę. W skład takiego alfabetu wchodzą symbole stałych, zmiennych, działań oraz nawiasów i przecinka. Jak wynika z definicji, termy są napisami złożonymi w tym sensie, że term może się składać z części składowych, które również są termami. Termy, które są częściami składowymi innych termów, nazywa się podtermami. Dokładniej: term t_1 jest *podtermem* termu t_2, gdy t_1 jest podsłowem słowa t_2.

Zbiór termów nad pustym zbiorem zmiennych, czyli $Term_{ALG}(\varnothing)$, nazywa się zbiorem termów stałych.

Termy są napisami, które wyrażają pewne znaczenie – reprezentują one wartości ze zbioru A. Inaczej: są one tekstową reprezentacją pewnych wartości, należących do nośnika A. Termy stałe $Term_{ALG}(\varnothing)$ wyrażają bezpośrednio pewne wartości. Termy, w których występują zmienne $Term_{ALG}(V)$, również reprezentują pewne wartości, ale wartości te zależą od wartościowania zmiennych, czyli od wartości, jakie są przyporządkowane zmiennym V. *Wartościowanie* zmiennych jest wyrażane przez funkcję v o sygnaturze

$$v : V \to A$$

Wartość funkcji $v(v)$ dla zmiennej v wyznacza pewien element ze zbioru A, który zmienna ta reprezentuje.

W dalszym ciągu będzie się pisać $Term(\varnothing)$ i $Term(V)$, gdy z kontekstu wiadomo, o jaką algebrę chodzi.

Przykład 8.9

Do zbioru termów stałych $Term(\varnothing)$, generowanych przez algebrę

$$ALG_{Nat} =_{def} <Nat, \{0, 1\} \cup \{+\}>$$

gdzie + jest – jak poprzednio – dodawaniem w zbiorze liczb naturalnych, zapisywanym w notacji wrostkowej, należą na przykład napisy:

0, 1, (0 + 0), (0 + 1), (1 + 0), (1 + 1), (0 + (0 + 0)), (0 + (0 + 1))

Przy zapisie w notacji przedrostkowej te same napisy przyjmą postać:

0, 1, +(0, 0), +(0, 1), +(1, 0), +(1, 1), +(0, +(0, 0)), +(0, +(0, 1))

Niektóre termy, na przykład 1, (0 + 1), (1 + 0), ((0 + 0) + 1), reprezentują tę samą wartość – liczbę naturalną 1. Zbiór termów reprezentujących tę samą wartość jest oczywiście nieskończony.

Niech $V = \{a, b, c\}$. Do zbioru termów $Term(V)$ generowanych przez algebrę ALG_{Nat} będą na przykład należeć:

$a, b, c, (a + 0), (0 + b), (a + c), (c + b), (a + (b + 0)), (1 + (b + 1))$

Jeżeli założyć funkcję wartościowania

$v = \{<a, 1>, <b, 0>, <c, 1>\}$

to wyżej wymienione termy będą kolejno reprezentować wartości:

1, 0, 1, 1, 0, 2, 1, 0, 2

Zbiór termów dla algebry wielorodzajowej

$ALG =_{def} <\{A_1, ..., A_k\}, \{c_1, ..., c_m\} \cup \{f_1, ..., f_n\}>$ dla $m \in Nat$ oraz $k, n \in Nat \setminus \{0\}$

jest bardziej złożony. Wynika to z podziału termów na rodzaje. Rodzaj termu wskazuje na jedną z dziedzin $A_1, ..., A_k$ algebry ALG, której wartości term reprezentuje.

Niech V_i będzie zbiorem zmiennych rodzaju A_i. Zbiorem wszystkich zmiennych jest $V = V_1 \cup ... \cup V_k$. Zmienna $v \in V_i$ jest rodzaju A_i $(i = 1, ..., k)$, co będzie zapisywane $v: A_i$. Zmiennej $v \in V_i$ można przyporządkowywać wartości tylko ze zbioru A_i.

Stałe także mają swój rodzaj. Dla $c \in \{c_1, ..., c_m\}$ zapis $c : A_i$ oznacza, że stała c jest rodzaju A_i, czyli jest elementem zbioru A_i $(i = 1, ..., k)$.

Dalej, zamiast pisać rodzaj A_i, będzie się pisać krótko rodzaj i.

Zbiór termów rodzaju i $(i = 1, ..., k)$ dla algebry wielorodzajowej ALG nad zbiorem zmiennych V, oznaczany $Term_i(V)$, jest zdefiniowany rekursywnie w sposób następujący:

- jeżeli $c_j : A_i$, to $c_j \in Term_i(V)$
  $V_i \subseteq Term_i(V)$
- jeżeli napisy $t_1, ..., t_k$ są termami rodzajów $i_1, ..., i_k$ oraz $f_j : A_{j_1} \times ... \times A_{j_k} \rightarrow A_i$ jest działaniem k-argumentowym, to napis postaci $f_j(t_1, ..., t_n)$ jest termem rodzaju i, czyli $f_j(t_1, ..., t_n) \in Term_i(V)$.

Zbiór wszystkich termów dla algebry wielorodzajowej ALG nad zbiorem zmiennych V, oznaczany $Term(V)$, jest określony jako mnogościowa suma

$$Term(V) = \bigcup_{i=1}^{k} Term_i(V).$$

Uwaga

W programowaniu przyporządkowanie rodzajów stałym, zmiennym oraz wyrażeniom nazywa się *typowaniem* lub *typizacją*. Typizacja przejawia się w sposobie deklarowania stałych i zmiennych, w rozróżnianiu na przykład wyrażeń arytmetycznych i logicznych, w sprawdzaniu poprawnego użycia zmiennych w wyrażeniach itd.

Przykład 8.10

W zdefiniowanej wcześniej algebrze

$$ALG_{Integer \cup \{nadmiar\}} =_{def}$$

$$< \{Integer \cup \{nadmiar\}, Boolean\}, \{0, 1\} \cup \{-, \oplus, \ominus, \otimes, \oslash, =, \geq\}>$$

wyróżnia się dwa rodzaje termów: $Integer \cup \{nadmiar\}$ oraz $Boolean$. Zakłada się, że:

$$Integer = \{-10, ..., -1, 0, ..., 10\}.$$

Niech ponadto $V_{Integer \cup \{nadmiar\}} =_{def} \{a, b\}$ oraz $V_{Boolean} =_{def} \varnothing$.

Termami rodzaju $Integer \cup \{nadmiar\}$ są na przykład:

$$0, 1, a, b, (a \ominus b), (a \ominus (a \otimes b)).$$

Termy te reprezentują pewne wartości ze zbioru $Integer \cup \{nadmiar\}$, przy czym zależą one od wartościowania v zbioru zmiennych. Niech $v = \{<a, 3>, <b, 4>\}$.

Term 0 reprezentuje wówczas wartość 0, term a – wartość 3, term b – wartość 4, a term $(a \ominus b)$ – wartość 7. Wartością termu $(a \ominus (a \otimes b))$ jest natomiast *nadmiar*, gdyż wartością jego podtermu $(a \otimes b)$ jest *nadmiar*, ponieważ $a*b > 10$.

Termami rodzaju *Boolean* są na przykład:

$$((a \ominus b) = (1 \otimes a)), \qquad ((-a \oplus b) \geq (b \oslash a))$$

Wartościami obu termów, przy wartościowaniu v, jest *false*.

Wartości termów przy danym wartościowaniu v w algebrze

$$ALG =_{def} <\{A_1, ..., A_k\}, \{c_1, ..., c_m\} \cup \{f_1, ..., f_n\}> \quad \text{dla } m \in Nat \text{ oraz } k, n \in Nat \setminus \{0\}$$

można zdefiniować ogólnie.

Przez $WAR_v(t)$ oznacza się wartość termu t przy wartościowaniu v. WAR_v jest funkcją o sygnaturze

$$WAR_v : Term(V) \rightarrow A$$

gdzie:

$$Term(V) = \bigcup_{i=1}^{k} Term_i(V),$$

$$A = \bigcup_{i=1}^{k} A_i.$$

Funkcję obliczania wartości termów WAR_v można zdefiniować rekursywnie w sposób następujący:

- jeżeli term jest postaci v, gdzie v jest zmienną, czyli $v \in V$, to $WAR_v(v) = v(v)$,
- jeżeli term jest postaci c, gdzie c jest stałą, czyli $c \in \{c_1, ..., c_m\}$, to $WAR_v(c) = c$,
- jeżeli term jest postaci $f(t_1, ..., t_k)$, gdzie f jest k-argumentowym działaniem, czyli $f \in \{f_1, ..., f_n\}$, $t_1, ..., t_k$ są termami, czyli $t_1, ..., t_k \in Term(V)$, to

  $WAR_v(f(t_1, ..., t_k)) = f(WAR_v(t_1), ..., WAR_v(t_k))$.

Niech $Term_{ALG}(\emptyset)$ będzie zbiorem termów stałych pewnej algebry ALG. Wartość termu stałego t nie zależy od wartościowania v. Dla dowolnych dwóch wartościowań v oraz v' zachodzi zatem

$$WAR_v(t) = WAR_{v'}(t)$$

Niech $WAR(t)$ oznacza wartość termu stałego t. Definiuje się relację binarną $\approx$ na zbiorze termów stałych:

jeżeli $t_1, t_2 \in Term_{ALG}(\emptyset)$, to $t_1 \approx t_2$ wtedy i tylko wtedy, gdy $WAR(t_1) = WAR(t_2)$.

Relacja $\approx$ jest oczywiście relacją równoważności i wyznacza podział zbioru termów stałych na klasy abstrakcji. Do jednej klasy abstrakcji należą wszystkie termy tego samego rodzaju, które reprezentują tę samą wartość. Oznacza to, że relację $\approx$ można byłoby określić jako rodzinę relacji binarnych zdefiniowanych na podzbiorze termów stałych ustalonego rodzaju, czyli $\approx =_{def} \{\approx_1, ..., \approx_k\}$, gdzie $\approx_i$ dla $i = 1, ..., k$ jest zdefiniowane:

jeżeli $t_1, t_2 \in Term_i(\emptyset)$, to $t_1 \approx t_2$ wtedy i tylko wtedy, gdy $WAR(t_1) = WAR(t_2)$.

Przykład 8.11

Dla algebry $ALG_{Integer \cup \{nadmiar\}}$, zdefiniowanej we wcześniejszym przykładzie, relacja R wyznacza podział termów stałych rodzaju $Integer \cup \{nadmiar\}$ na klasy abstrakcji, z których każda reprezentuje termy przyjmujące wartości –10, ..., –1, 0, ..., 10 oraz *nadmiar*. Każda z klas zawiera nieskończenie wiele termów. Na przykład do jednej klasy termów o wartości 0 należą między innymi:

0 $\quad$ $((0 \oplus 1) \otimes 0)$ $\quad$ $(((1 \otimes 2) \ominus 3) \oplus 1)$

Do klasy termów o wartości *nadmiar* należą między innymi:

$(1 \oplus nadmiar) \quad ((0 \ominus nadmiar) \otimes 1) \quad (((1 \oslash 0) \ominus 0) \ominus 0)$

Niech $Term_{ALG}(V)$ będzie zbiorem termów algebry ALG na zbiorem zmiennych V. Na zbiorze tym można również zdefiniować relację równoważności, która jest uogólnieniem relacji równoważności $\approx$, zdefiniowanej na zbiorze termów stałych. Będzie ona oznaczona tym samym symbolem $\approx$ i będzie zdefiniowana jako rodzina relacji

$$\approx =_{def} \{\approx_1, ..., \approx_k\},$$

gdzie $\approx_i$ dla $i = 1, ..., k$ jest zdefiniowane:

jeżeli $t_1, t_2 \in Term_i(V)$, to $t_1 \approx_i t_2$ wtedy i tylko wtedy, gdy $WAR_v(t_1) = WAR_v(t_2)$ dla każdego wartościowania v.

Jeżeli dwa termy należą do jednej klasy abstrakcji wyznaczonej przez relację $\approx$, to dla dowolnie wybranego wartościowania reprezentują one tę samą wartość.

Przykład 8.12

Dla poprzednio rozważanej algebry

$$ALG_{Całkowite} =_{def} <\{Całkowite, Boolean\}, \{0, 1\} \cup \{-, +, -, *, /, =, \geq\}>$$

i zbioru termów nad zbiorem zmiennych $V_{Całkowite} =_{def} \{a, b\}$ oraz $V_{Boolean} =_{def} \emptyset$, relacja $\approx_V$ wyznacza podział termów rodzaju $Integer \cup \{nadmiar\}$ na klasy abstrakcji, z których każda reprezentuje termy przyjmujące te wartości dla dowolnie ustalonego wartościowania. Na przykład do tej samej klasy termów należą między innymi:

$1 \qquad ((a - a) + 1) \qquad (((1/1 - b) - 0) + b)$

Do innej klasy należą między innymi termy:

$a \qquad ((a - b) + b) \qquad (((a*1) - b) + b)$

8.4. Algebry Boole'a

Wśród algebr, które w informatyce mają szerokie zastosowania, szczególną rolę odgrywają *algebry Boole'a*. Stanowią one pewną klasę algebr zdefiniowaną przez określenie pewnych własności wyrażanych w postaci równości. Taki sposób definiowania algebr nazywa się *definiowaniem równościowym*.

Algebrą Boole'a określa się każdą algebrę o strukturze

$$BOOL =_{def} <A, \{0, 1\} \cup \{-, +, *\}>$$

gdzie:

- A jest dowolnym zbiorem, nośnikiem algebry,
- 0, 1 są stałymi, nazywanymi *zerem* i *jednością boolowską*,
- $-$ jest działaniem jednoargumentowym, nazywanym dopełnieniem boolowskim,
- $+, *$ są działaniami dwuargumentowymi, nazywanymi umownie *dodawaniem* i *mnożeniem boolowskim* (nie należy tych działań utożsamiać z działaniami arytmetycznymi ani mnogościowymi),

która ponadto, dla dowolnych $a, b, c \in A$, spełnia następujące własności:

1. $(a + b) = (b + a)$	$(a * b) = (b * a)$
2. $((a + b) + c) = (a + (b + c))$	$((a * b) * c) = (a * (b * c))$
3. $((a + b) * c) = ((a * c) + (b * c))$	$((a * b) + c) = ((a + c) * (b + c))$
4. $(a + 0) = a$	$(a * 1) = a$
5. $(a + (-a)) = 1$	$(a * (-a)) = 0$

Własności te wyrażają:

1. *przemienność* dodawania i mnożenia,
2. *łączność* dodawania i mnożenia,
3. *rozdzielność* mnożenia względem dodawania oraz dodawania względem mnożenia,
4. *neutralność* zera względem dodawania oraz jedynki względem mnożenia,
5. *neutralność* elementu odwrotnego względem dodawania oraz względem mnożenia.

Własności są wyrażane poprzez równości. Równość jest napisem postaci $t_1 = t_2$, gdzie składowe t_1 i t_2 są termami ze zbioru $Term_{BOOL}(V)$ nad dowolnym zbiorem zmiennych V. Równość $t_1 = t_2$ jest spełniona, gdy dla dowolnego wartościowania v oba termy reprezentują tę samą wartość, czyli $WAR_v(t_1) = WAR_v(t_2)$.

Od algebry Boole'a wymaga się, aby były spełnione wszystkie wyżej wymienione równości.

Przykład 8.13

Łatwo sprawdzić, że wcześniej zdefiniowana algebra

$$ALG_{Boolean} =_{def} \langle Boolean, \{not, and, or\}\rangle$$

ma wszystkie wymagane własności, a zatem jest algebrą Boole'a.

Przykład 8.14

Niech U będzie dowolnym zbiorem, a 2^U rodziną wszystkich jego podzbiorów. Podobnie można sprawdzić, że algebrą Boole'a jest struktura

$$BOOL_U =_{def} \langle 2^U, \{\emptyset, U\} \cup \{\setminus, \cup, \cap\}\rangle$$

w której: 2^U jest nośnikiem algebry, $\emptyset$ oraz U są zerem i jednością, a działania mnogościowe $\setminus$, $\cup$, $\cap$ są odpowiednio dopełnieniem, sumą i iloczynem algebry.

Ogólniej, zamiast pełnej rodziny podzbiorów 2^U wystarczy przyjąć dowolną jej podrodzinę, zamkniętą ze względu na działania dopełnienia, sumy i iloczynu. Zamknięta jest na przykład rodzina złożona ze zbioru pustego i zbioru pełnego, czyli $\{\emptyset, U\}$, to znaczy wynikiem każdej z operacji wykonanej na elementach tej rodziny jest element należący do tej rodziny.

8.5. Homomorfizm algebr

Definiowanie algebry abstrakcyjnej wygodnie jest rozpoczynać od opisania jej struktury. Najprostszą jej charakterystyką jest sygnatura.

Sygnaturą algebry nazywa się parę

$$Sig =_{def} <S, OP>$$

gdzie: S jest niepustym zbiorem identyfikatorów (nazw) nośników (rodzajów), OP jest zbiorem deklaracji operacji. *Deklaracja operacji* będzie zapisywana w postaci

$$op : s_1\ s_2\ \ldots\ s_n \rightarrow s$$

gdzie: op jest identyfikatorem (nazwą) operacji, $s_1\ s_2\ \ldots\ s_n$ jest listą, której elementy s_1, s_2, $\ldots\ s_n \in S$ są identyfikatorami rodzajów argumentów, a $s \in S$ jest identyfikatorem (nazwą) rodzaju wartości operacji. Deklaracja operacji o nazwie op wskazuje na nazwy zbiorów jej argumentów i nazwę zbioru jej wartości. Jeżeli op jest operacją zeroargumentową, czyli stałą, to jej deklaracja ma postać

$$op : \rightarrow s$$

Zakłada się, że każda deklaracja operacji ma różną nazwę operacji, dlatego dalej, zamiast pisać $(op : s_1\ s_2\ \ldots\ s_n \rightarrow s) \in OP$, będzie się pisać krótko $op \in OP$.

Uwaga

Zapis postaci

$$op : s_1\ s_2\ \ldots\ s_n \rightarrow s$$

nie jest tym samym co zapis

$$op : s_1 \times s_2 \times \ldots \times s_n \rightarrow s$$

Przykład 8.15

Przykłady dwóch sygnatur $Sig_i =_{def} <S_i, OP_i>$ dla $i = 1, 2$, gdzie:

$S_1 =_{def} \{A\}$ $\quad$ $OP_1 =_{def} \{e : \rightarrow A,\ d : A\ A \rightarrow A\}$

$S_2 =_{def} \{A, B\}$ $\quad$ $OP_2 =_{def} \{e : \rightarrow A,\ d : A\ A \rightarrow A,\ r : A\ A \rightarrow B\}$

Sygnatura jest tylko pewną charakteryzacją algebry, a nie określeniem algebry. Może być wiele algebr mających tę samą sygnaturę.

Algebrą nad sygnaturą *Sig*, krótko *Sig-algebrą*, nazywa się parę

$ALG =_{def} <A, F>$

gdzie:

$A =_{def} \{A_s \mid s \in S\}$ jest rodziną zbiorów zwanych *nośnikami* lub *dziedzinami* algebry,

$F =_{def} \{f_{op} \mid op \in OP\}$ jest rodziną funkcji zwanych *operacjami* algebry, przy czym każdej deklaracji operacji $op \in OP$:

$op : s_1 s_2 \ldots s_n \rightarrow s$

odpowiada funkcja

$f_{op} : A_{s_1} \times \ldots \times A_{s_n} \rightarrow A_s$

Dwie algebry o tej samej sygnaturze nazywa się algebrami *podobnymi*.

Przykład 8.16

Przykładami dwóch algebr podobnych o sygnaturze Sig_1 z poprzedniego przykładu są:

$ALG_1 =_{def} <Nat, \{1, +\}>$

gdzie nośnikiem algebry ALG_1 jest zbiór liczb naturalnych *Nat*, stała 1 jest liczbą naturalną jeden, zaś operacja + jest dodawaniem w zbiorze liczb naturalnych.

$ALG_2 =_{def} <Nat, \{1, *\}>$

Nośnikiem algebry ALG_2 jest zbiór liczb naturalnych *Nat*, stała 1 jest liczbą naturalną jeden, zaś operacja * jest mnożeniem w zbiorze liczb naturalnych.

Przykładami dwóch podobnych algebr dwurodzajowych o sygnaturze Sig_2 są:

$ALG_3 =_{def} <\{Nat, Logiczne\}, \{1, +, =\}>$

Nośnikami algebry ALG_3 są zbiór liczb naturalnych *Nat* oraz zbiór wartości logicznych $Logiczne =_{def} \{prawda, fałsz\}$, 1 oraz + są – jak poprzednio – stałą jeden i dodawaniem w zbiorze liczb naturalnych, natomiast operacja = jest równością w zbiorze liczb naturalnych.

$ALG_4(X) =_{def} <\{2^X, Logiczne\}, \{\emptyset, \cup, =\}>$

Algebra $ALG_4(X)$ jest algebrą parametryzowaną zbiorem X. Jej nośnikami są: rodzina podzbiorów pewnego zbioru X oraz zbiór *Logiczne*, a operacjami są: stała $\emptyset$, która jest zbiorem pustym, operacja $\cup$, która jest sumą mnogościową, oraz =, która jest równością zbiorów.

Niech będą dane dwie algebry: $ALG_1 =_{def} <A, F>$ oraz $ALG_2 =_{def} <B, G>$ o sygnaturze $Sig = <S, OP>$. Oznacza to, że

$A =_{def} \{A_s \mid s \in S\}$ oraz $B =_{def} \{B_s \mid s \in S\}$ są *nośnikami* algebr, a
$F =_{def} \{f_{op} \mid op \in OP\}$ oraz $G =_{def} \{g_{op} \mid op \in OP\}$ są rodzinami *operacji* tych algebr.

Homomorfizmem z algebry ALG_1 w algebrę ALG_2 nazywa się taką rodzinę funkcji H

$$H =_{def} \{h_s : A_s \to B_s \mid s \in S\}$$

że dla każdej funkcji $f_{op} : A_{s_1} \times \ldots \times A_{s_n} \to A_s$, dla $op \in OP$, zachodzi warunek

$$h_s(f_{op}(a_1, \ldots, a_n)) = g_{op}(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$$

dla dowolnych $a_j \in A_{s_j}$, $j = 1, \ldots, n$. Fakt, że H jest homorfizmem zapisuje się

$$H : ALG_1 \to ALG_2$$

Przykład 8.17

Dane są dwie algebry jednorodzajowe:

$ALG_1 = <Nat, \{1, +\}>$

$ALG_2 = <Nat_n, \{1, /\}>$

gdzie: $Nat_n = \{0, 1, \ldots, n-1\}$, a / oznacza dodawanie modulo n.

Homomorfizmem jest funkcja $h : Nat \to Nat_n$, zdefiniowana wzorem

$h(m)$ = reszta z dzielenia m przez n, dla $m \in Nat$.

Jeżeli każda z funkcji h_s homomorfizmu $H = \{h_s : A_s \to B_s \mid s \in S\}$ jest funkcją wzajemnie jednoznaczną, to H nazywa się *izomorfizmem*.

8.6. Algebra ilorazowa termów

Dana algebra wielorodzajowa $ALG = <A, F>$ o sygnaturze $Sig = <S, OP>$,

gdzie:

$A = \{A_s \mid s \in S\}$ jest rodziną nośników algebry ALG,

$F = \{f_{op} \mid op \in OP\}$ jest zbiorem operacji algebry ALG,

generuje zbiór termów nad ustalonym zbiorem zmiennych V

$$Term(V) = \bigcup_{s \in S} Term_s(V)$$

Zbiór ten może być podstawą do utworzenia nowej algebry wielorodzajowej, zwanej *algebrą termów* [Ehrig, Mahr 1985], [Rasiowa 1998], która jest podobna do algebry ALG.

Algebrę termów

$$ALG_{Term} =_{def} <A, F>$$

dla algebry *ALG* definiuje się następująco:

$A = \{Term_s(V) \mid s \in S\}$ jest rodziną nośników algebry termów,

$F = \{f_{op} \mid op \in OP\}$ jest zbiorem operacji algebry termów, przy czym operacja f_{op} ma sygnaturę:

$$f_{op} : Term_{s_1}(V) \times \ldots \times Term_{s_n}(V) \rightarrow Term_s(V)$$

gdy deklaracja operacji ma postać: $op : s_1\, s_2, \ldots, s_n \rightarrow s$ i jest zdefiniowana następująco:

jeżeli $t_j \in Term_{s_j}$ dla $j = 1, \ldots, n$, to $f_{op}(t_1, \ldots, t_n) =_{def} f_{op}(t_1, \ldots, t_n)$.

Każdemu nośnikowi A_s i każdej operacji f_{op} w algebrze *ALG* odpowiadają nośnik $Term_s(V)$ i operacja f_{op} w algebrze termów ALG_{Term}.

Niech $\approx$ będzie wcześniej określoną rodziną relacji binarnych $\approx_s$ na zbiorze termów rodzaju $s \in S$.

Relacje te są relacjami równoważności i mają następującą własność:

Jeżeli t_{s_j}, t'_{s_j} są termami rodzaju s_j oraz $t_{s_j} \approx_{s_j} t'_{s_j}$, dla $j = 1, \ldots, n$, to

$$f_{op}(t_{s_1}, \ldots, t_{s_n}) \approx_s f_{op}(t'_{s_1}, \ldots, t'_{s_n})$$

dla $op \in OP$.

Rodzinę relacji równoważności o tej własności nazywa się *kongruencją*.

Kongruencja jest podstawą do zdefiniowania algebry, nazywanej *ilorazową algebrą termów* $ALG_{\approx}$ dla algebry *ALG*. Dodatkowo algebry te są homomorficzne, to znaczy istnieje homomorfizm $H : ALG \rightarrow ALG_{\approx}$.

Definicja algebry $ALG_{\approx}$ jest następująca:

$$ALG_{\approx} =_{def} < \overline{A}, \overline{F} >$$

gdzie:

$\overline{A} = \{Term_s(V)/\approx_s \mid s \in S\}$ jest rodziną zbiorów ilorazowych termów rodzajów $s \in S$,

$\overline{F} = \{ \overline{f}_{op} \mid op \in OP\}$ jest zbiorem operacji ilorazowej algebry termów, przy czym operacja $\overline{f}_{op}$ ma sygnaturę

$$\overline{f}_{op} : Term_{s_1}(V) / \approx_{s_1} \times \ldots \times Term_{s_n}(V) / \approx_{s_n} \rightarrow Term_s(V) / \approx_s$$

gdy deklaracja operacji ma postać: $op : s_1\, s_2 \ldots s_n \rightarrow s$ i jest zdefiniowana następująco:

jeżeli $[t_j] \in Term_{s_j} / \approx_{s_j}$, dla $j = 1, \ldots, n$, to $\overline{f}_{op}([t_1], \ldots, [t_n]) =_{def} [f_{op}(t_1, \ldots, t_n)]$.

Należy przypomnieć, że $[t_j]$ jest oznaczeniem klasy abstrakcji generowanej przez term t_j.

Ćwiczenia

1. Niech będzie dana algebra $ALG_{Nat} =_{def} <Nat, \{0, 1\}, \{+, -, *\}>$. Przedstawić gramatykę generującą poprawne wyrażenia (termy) zbudowane ze zmiennych reprezentujących liczby oraz z operacji podanej algebry.
2. Niech będzie dana algebra $ALG_{Int} =_{def} <Int, \{+, -, *\}>$, gdzie $Int =_{def} \{-100, ..., 0, 1, ..., 100\}$. Przedstawić definicję działań algebry pozwalającą na określenie ich wyniku dla dowolnych argumentów.
3. Zdefiniować algebrę definiującą typ znakowy **(string)** w języku Pascal lub C.
4. Przedstawić wielorodzajową algebrę, która będzie wyrażać znaczenie typu wyliczeniowego, zdefiniowanego w języku Pascal w sposób następujący:

   type *DniTygodnia* = (*pon*, *wt*, *śr*, *czw*, *piąt*, *sob*, *niedz*).
5. Zdefiniować gramatykę, która będzie generować zbiór termów rodzaju *DniTygodnia*, określonych przez algebrę zdefiniowaną w zadaniu 4.
6. Dla algebry z zadania 4. zdefiniować algebrę termów i ilorazową algebrę termów.
7. Niech $A =_{def} \{1, 2, 3, 5, 6, 10, 15, 30\}>$. Pokazać, że algebra *ALG* zdefiniowana następująco:

   $ALG = <A, \{1, 30\} \cup \{-, +, *\}>$

   gdzie dla $a, b \in A$

   $-a$ oznacza liczbę stanowiącą wynik dzielenia 30 przez a,

   $a + b$ oznacza najmniejszą wspólną wielokrotność liczb a oraz b,

   $a * b$ oznacza największy wspólny podzielnik liczb a oraz b,

   jest algebrą Boole'a.
8. Przedstawić wszystkie homomorfizmy algebry słów $ALG_1 = <\{a\}^*, \{\varepsilon, ^\wedge\}>$ nad alfabetem $\{a\}$ w algebrę słów $ALG_2 = <\{a, b\}^*, \{\varepsilon, ^\wedge\}>$ nad alfabetem $\{a, b\}$, gdzie ε jest słowem pustym, a $^\wedge$ jest konkatenacją słów.
9. Niech $ALG = <\{a, b\}^*, \{^\wedge\}>$, gdzie $^\wedge$ jest konkatenacją słów, będzie algebrą słów nad alfabetem $\{a, b\}$. Jaka jest moc zbioru wszystkich homomorfizmów $h : \{a, b\}^* \rightarrow \{a, b\}^*$ algebry słów *ALG* w samą siebie?
10. Pokazać, że istnieje homomorfizm między dowolną algebrą a generowaną przez nią ilorazową algebrą termów.

9. Rachunek zdań

9.1. Składnia

Rachunek zdań jest podstawową częścią logiki klasycznej. Elementy rachunku były nieformalnie wprowadzone i używane w poprzednich rozdziałach. W tym rozdziale przedstawia się składnię i semantykę rachunku zdań.

Język rachunku zdań, tak jak każdy język formalny, definiuje się przez podanie alfabetu – zbioru symboli podstawowych, oraz przez podanie zasad tworzenia z nich napisów – słów nad alfabetem. Symbole alfabetu nazywa się jednostkami *leksykalnymi*. Takie wyróżnienie wynika stąd, że jednostki leksykalne mogą być słowami nad innym alfabetem.

Alfabet języka rachunku zdań składa się z następujących czterech kategorii jednostek *leksykalnych*:

- symboli *stałych logicznych* reprezentowanych przez napisy **true** oraz **false**;
- przeliczalnej liczby symboli *zmiennych zdaniowych*, reprezentowanych przez dowolnie ustalone identyfikatory, dalej najczęściej będą używane pojedyncze małe litery p, q, r, ... ;
- symboli *spójników logicznych*:

negacji	$\neg$
koniunkcji	$\wedge$
dysjunkcji (lub *alternatywy*)	$\vee$
implikacji	$\Rightarrow$
równoważności	$\Leftrightarrow$

- dwóch symboli *pomocniczych*:

lewy nawias	(
prawy nawias	)

Alfabet rachunku zdań jest zbiorem nieskończonym, ale co najwyżej przeliczalnym, gdyż dopuszcza się używanie dowolnej liczby identyfikatorów do reprezentacji zmiennych zdaniowych. W praktycznych zastosowaniach dysponuje się oczywiście zawsze skończoną liczbą zmiennych zdaniowych. Ich postać – ustalana dowolnie – nie ma wpływu na znaczenie języka.

Z alfabetu tworzy się pewne napisy – formuły, które – z definicji – są napisami poprawnie zbudowanymi. Dalej pojedyncze formuły będą oznaczane małymi literami greckimi.

Zbiór *formuł* rachunku zdań *FORM*, nad określonym wyżej alfabetem, jest definiowany rekursywnie w następujący sposób:

- symbole zmiennych zdaniowych oraz symbole stałych logicznych są formułami, nazywa się je formułami *elementarnymi* albo *atomowymi*;
- jeżeli α oraz β są formułami, to formułami nazywanymi formułami *złożonymi* są napisy:

$$\neg\alpha, (\alpha \Rightarrow \beta), (\alpha \wedge \beta), (\alpha \vee \beta), (\alpha \Leftrightarrow \beta).$$

Zbiór formuł *FORM* jest językiem formalnym rachunku zdań. Formuły rachunku zdań też nazywa się *zdaniami*.

Jeżeli α jest formułą, to każde podsłowo słowa α, które jest formułą, nazywa się *podformułą* α.

Przykład 9.1

Jeżeli dana jest formuła $(\alpha \wedge \beta)$, to jej podformułami są α oraz β, a także wszystkie podformuły α oraz β. Dla formuły

$$(\neg(p \vee q) \wedge \neg r),$$

gdzie: p, q, r są zmiennymi zdaniowymi, jej podformułami są formuły:

$$\neg(p \vee q), \quad (p \vee q), \quad p, \quad q, \quad \neg r, \quad r.$$

Uwaga

W celu zredukowania liczby nawiasów w formułach dalej się przyjmuje (jak w rozdziale 1.), że spójniki logiczne mają ustaloną kolejność stosowania (wiązania) spójników (od najsilniejszego do najsłabszego):

$$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow.$$

Pozwala to pisać na przykład:

$\neg\alpha \wedge \beta$ zamiast $(\neg\alpha \wedge \beta)$,

$\neg\alpha \wedge \beta \vee \gamma$ zamiast $((\neg\alpha \wedge \beta) \vee \gamma)$,

gdzie: α oraz β są dowolnymi formułami.

Gdy takie same spójniki występują obok siebie, zakłada się dodatkowo, że występujące obok siebie spójniki $\wedge$, $\vee$ łączą w lewo, a występujące obok siebie spójniki $\Rightarrow$, $\Leftrightarrow$ łączą w prawo. Na przykład:

$p \wedge q \wedge r$ znaczy $(p \wedge q) \wedge r$,

$p \Rightarrow q \Rightarrow r$ znaczy $p \Rightarrow (q \Rightarrow r)$.

Przedstawiony język formalny rachunku zdań abstrahuje od postaci zmiennych zdaniowych. Język ten można ukonkretnić, definiując odpowiednią gramatykę bezkontekstową.

Przyjmuje się konwencję powszechnie stosowaną w językach programowania, że identyfikatorem jest niepusty skończony ciąg znaków, którego pierwszym elementem jest dowolna litera alfabetu łacińskiego, a elementami pozostałymi są litery lub cyfry arabskie.

Gramatykę generującą język formalny rachunku zdań (*RZ*) można zdefiniować następująco:

$G_{RZ} =_{def} <T_{RZ}, N_{RZ}, P_{RZ}, S_{RZ}>$,

gdzie:

$T_{RZ} =_{def} \{\mathbf{true}, \mathbf{false}\} \cup \{a, b, ..., z\} \cup \{0, 1, ..., 9\} \cup \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\} \cup \{(,)\}$

$N_{RZ} =_{def} \{$*formuła, formuła-elementarna, stała-logiczna, zmienna-zdaniowa, identyfikator, litera, cyfra spójnik-logiczny*$\}$

$S_{RZ} =_{def}$ *formuła*

a zbiór produkcji P_{RZ}, zapisany w konwencji BNF, ma postać

$P_{RZ} =_{def} \{$*formuła* ::= *formuła-elementarna* $|\neg$*formuła*$|$
(*formuła binarny-spójnik formuła*)
binarny-spójnik-logiczny ::= $\wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow$
formuła-elementarna ::= *stała-logiczna* | *zmienna-zdaniowa*
stała-logiczna ::= **true** | **false**
zmienna-zdaniowa ::= *identyfikator*
identyfikator ::= *litera* | *identyfikator cyfra* | *identyfikator litera*
litera ::= *a* | *b* | ... | *z*
cyfra ::= 0 | 1 | ... | 9 $\}$

Generowany przez podaną gramatykę G_{RZ} język formalny $L(G_{RZ})$ jest konkretyzacją zdefiniowanego rekursywnie zbioru formuł *FORM*.

W celu skrócenia zapisów całe formuły będą oznaczane pojedynczymi symbolami i dlatego będzie przydatne pojęcie równości tekstowej formuł. Fakt, że dwie formuły α, β są *identyczne tekstowo*, będzie zapisywany w postaci $\alpha \equiv \beta$.

9.2. Semantyka

Język formalny rachunku zdań w postaci przedstawionej wyżej jest językiem bez interpretacji. Interpretacja języka polega na ustaleniu znaczenia elementów języka – jego jednostek leksykalnych oraz formuł. W celu przedstawienia interpretacji jest konieczne posiadanie pewnego zestawu pojęć i sposobu ich reprezentacji w jakimś

zrozumiałym języku, to znaczy potrzebne jest posiadanie *metajęzyka* – języka służącego do opisu innego języka. Używanym tu metajęzykiem będzie język teorii mnogości, który był przedstawiany w poprzednich rozdziałach.

Określenie interpretacji języka polega na ustaleniu dziedzin interpretacji, to jest zbiorów obiektów, które będą wyrażać znaczenie elementów języka, oraz na ustaleniu sposobu przyporządkowania elementom języka obiektów z dziedziny interpretacji.

Dziedziną interpretacji rachunku zdań jest zbiór wartości logicznych:

$$Logiczne =_{def} \{prawda, fałsz\}$$

Dalej, zamiast pisać *prawda*, *fałsz*, będą używane skróty **P**, **F**.

Przyporządkowanie znaczenia elementom języka obiektów nad dziedziną interpretacji dokonuje się w dwóch etapach: najpierw określa się znaczenie symboli stałych i spójników logicznych, a następnie określa się znaczenie formuł.

W pierwszym etapie wprowadza się *funkcję interpretacji bazowej* I, krótko – *interpretację*, która określa znaczenie symboli stałych i spójników logicznych.

Interpretacją (znaczeniem) symboli **true** oraz **false** są wartości logiczne, odpowiednio *prawda* oraz *fałsz*. Formalnie wyraża to funkcja interpretacji I w sposób następujący:

$$I(\mathbf{true}) =_{def} \mathbf{P}$$
$$I(\mathbf{false}) =_{def} \mathbf{F}$$

Symbolom spójników logicznym: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, interpretacja I przyporządkowuje funkcje o następujących sygnaturach:

$$I(\neg) : Logiczne \rightarrow Logiczne$$
$$\_\, I(\wedge) \,\_ , \_\, I(\vee) \,\_ , \_\, I(\Rightarrow) \,\_ , \_\, I(\Leftrightarrow) \,\_ : Logiczne^2 \rightarrow Logiczne$$

Funkcje te są określone szczegółowo przez tablicę 9.1.

Tablica 9.1

a	b	$I(\neg)(a)$	$a\ I(\wedge)\ b$	$a\ I(\vee)\ b$	$a\ I(\Rightarrow)\ b$	$a\ I(\Leftrightarrow)\ b$
P	**P**	**F**	**P**	**P**	**P**	**P**
F	**P**	**P**	**F**	**P**	**P**	**F**
F	**F**	**P**	**F**	**F**	**P**	**P**
P	**F**	**F**	**F**	**P**	**F**	**F**

Tablica określa tak zwaną *standardową* albo *główną* interpretację spójników logicznych. W dalszym ciągu symbolom spójników logicznych będzie przyporządkowywana wyłącznie standardowa interpretacja, dlatego w zapisie symboli stałych i spójników logicznych symbol interpretacji I będzie pomijany. W zależności od kontekstu symbole spójników będą traktowane wyłącznie jako symbole bądź jako wyżej zdefiniowane funkcje. Tak właśnie było w podrozdziale 1.2, w którym po raz pierw-

szy zdefiniowano znaczenie spójników logicznych; symbole spójników logicznych były tam traktowane jako funkcje.

Dziedzina interpretacji wraz z funkcją interpretacji stałych i spójników logicznych wyznaczają algebrę jednorodzajową postaci

$$<Logiczne, \{I(\mathbf{true}), I(\mathbf{false})\}, \{I(\neg), I(\wedge), I(\vee), I(\Rightarrow), I(\Leftrightarrow)\}>.$$

Drugim etapem definiowania interpretacji języka rachunku zdań jest określenie znaczenia (semantyki) dowolnych formuł.

Dokonuje się tego rekursywnie – podobnie jak dla termów (podrozdział 8.3) – rozpoczynając od formuł elementarnych. Stałe, które są formułami elementarnymi, mają już ustaloną interpretację. Formułami elementarnymi są też zmienne zdaniowe. Pojedyncza zmienna reprezentuje prostą, niepodzielną wypowiedź, której można dowolnie przypisać wartość logiczną *prawda* albo *fałsz*. Interpretacja (znaczenie) symbolu zmiennej zdaniowej polega więc na przypisaniu temu symbolowi wartości **P** (*prawda*) albo **F** (*fałsz*). Niech *ZmienneZdaniowe* oznacza zbiór zmiennych zdaniowych. Przypisanie wartości zmiennej będzie wyrażać funkcja *wartościowania* zmiennych

$$v : ZmienneZdaniowe \rightarrow Logiczne,$$

która zmiennej zdaniowej $p \in ZmienneZdaniowe$ przyporządkowuje pewną wartość logiczną $v(p) \in Logiczne$.

Mając ustaloną funkcję interpretacji bazowej I oraz funkcję wartościowania v można jednoznacznie zdefiniować nową funkcję, która każdej formule $\alpha \in FORM$ przyporządkowuje wartość logiczną *prawda* albo *fałsz*. Ta nowa funkcja

$$INT_v : FORM \rightarrow Logiczne$$

jest zdefiniowana rekursywnie w sposób następujący:

- Jeżeli formuła α jest formułą w postaci stałej logicznej, to:

  $INT_v(\mathbf{true}) =_{def} I(\mathbf{true}) = \mathbf{P}$
  $INT_v(\mathbf{false}) =_{def} I(\mathbf{false}) = \mathbf{F}$

- Jeżeli formuła α ma postać zmiennej zdaniowej p, to

  $INT_v(\alpha) =_{def} v(p)$.

- Jeżeli formuła jest formułą złożoną, to:

  $INT_v(\neg\alpha) =_{def} I(\neg)(INT_v(\alpha))$,
  $INT_v(\alpha \circ \beta) =_{def} INT_v(\alpha)\ I(\circ)\ INT_v(\beta)$,

  gdzie: $\circ$ jest dowolnym binarnym spójnikiem logicznym, czyli $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$, a $I(\circ)$ jest jego interpretacją.

Interpretacja stałych logicznych nie zależy od wartościowania v, a interpretacja zmiennych zdaniowych nie zależy od interpretacji bazowej I.

Uwaga

Do opisu semantyki formalnego języka rachunku zdań został użyty pewien metajęzyk. Warunkiem decydującym o wyborze danego metajęzyka jest jego zrozumiałość i dostateczna siła ekspresji, pozwalająca na wyrażenie odpowiednich faktów. W naszym przypadku metajęzykiem jest język elementarnej teorii zbiorów, który został wprowadzony w poprzednich rozdziałach. Do opisu języka elementarnej teorii mnogości był natomiast użyty język naturalny, który pełnił rolę metajęzyka względem języka teorii mnogości. Idea definiowania znaczenia języka za pomocą innego języka – metajęzyka pochodzi od polskiego logika Alfreda Tarskiego[18].

Rozróżnienie pomiędzy językiem a metajęzykiem wyeliminowało wiele, znanych jeszcze w starożytności, paradoksów lingwistycznych w rodzaju: *to zdanie jest prawdziwe* albo *to zdanie jest fałszywe*. Przypuścimy, że chcemy wykazać, że *Ziemia jest płaska*. W tym celu wystarczy rozpatrzyć zdanie:

Albo *całe to zdanie jest fałszywe*, albo *Ziemia jest płaska*.

Zdanie to jest albo prawdziwe, albo fałszywe. Jeśli jest fałszywe, to – zgodnie z jego treścią – Ziemia musi być płaska. Jeśli zaś jest prawdziwe, to prawdziwa musi być albo jego pierwsza część: *całe to zdanie jest fałszywe*, albo druga: *Ziemia jest płaska*. Skoro przyjęliśmy, że całe zdanie jest prawdziwe, prawdziwa więc musi być jego druga część, czyli że Ziemia jest płaska. Zastępując zdanie *Ziemia jest płaska* dowolnym innym zdaniem, mocą takiego samego rozumowania możemy wykazać jego prawdziwość.

Rozróżnienie pomiędzy językiem a metajęzykiem ma jeszcze jedną zdumiewającą konsekwencję – nie istnieje nic takiego jak prawda absolutna. Można prowadzić dowody w obrębie jednego języka, określające, które wyrażenia języka uznajemy za prawdziwe. Pojęcie prawdy nie jest sformułowane w tym języku, lecz w jego metajęzyku. Z kolei, w obrębie metajęzyka mamy do czynienia z innymi wyrażeniami i dowodami, które określają ich prawdziwość, ale pojęcie prawdziwości zdań metajęzyka jest określone w jego metajęzyku (czyli metametajęzyku).

Formuła α *spełnia interpretację INT przy wartościowaniu* v, co będzie zapisywane w postaci

$$INT_v \models \alpha$$

wtedy i tylko wtedy, gdy $INT_v(\alpha) = \mathbf{P}$. Symbol $\models$ jest nazywany symbolem spełniania.

Formuła α *spełnia interpretację INT*, co będzie zapisywane w postaci

$$INT \models \alpha$$

wtedy i tylko wtedy, gdy α spełnia interpretację *INT* przy dowolnym wartościowaniu v.

[18] Alfred Tarski (1901–1983).

Ponieważ jest rozważana tylko interpretacja standardowa, symbol *INT* będzie pomijany

$$\models \alpha$$

Formułę taką nazywa się *tautologią* rachunku zdań.

Dwie formuły α oraz β są *równoważne semantycznie*, jeżeli przy tej samej interpretacji i przy tym samym wartościowaniu są jednocześnie spełnione albo niespełnione. Fakt równoważności semantycznej formuł zapisuje się w postaci

$$\alpha = \beta.$$

Uwaga

Symbol równoważności semantycznej = należy odróżnić od symbolu równoważności tekstowej ≡. Dla dwóch dowolnych formuł α, β, jeżeli $\alpha \equiv \beta$, to oczywiście również $\alpha = \beta$, natomiast wynikanie odwrotne nie zachodzi.

Pomiędzy spójnikiem równoważności $\Leftrightarrow$ a równoważnością semantyczną = zachodzi związek wyrażający się przez własność

Formuła postaci $\alpha \Leftrightarrow \beta$ jest tautologią, wtedy i tylko wtedy, gdy $\alpha = \beta$.

9.3. Dowodzenie metodą zero-jedynkową

Bezpośrednio z definicji interpretacji wynika, że sprawdzenie, czy dana formuła jest tautologią, może polegać na wyliczeniu prawdziwości formuły dla wszystkich możliwych wartościowań zmiennych zdaniowych występujących w tej formule. Liczba takich wartościowań wynosi 2^n, gdzie n jest liczbą zmiennych zdaniowych. Postępowanie takie określa się mianem *metody zero-jedynkowej*. Jej istotę wyjaśnia przykład.

Przykład 9.2

W celu pokazania, że formuła

$$p \Rightarrow (q \Rightarrow p)$$

jest tautologią rachunku zdań, wystarczy zbudować tablicę prawdziwościową, w której są zestawione wszystkie możliwe wartościowania zmiennych i obliczone dla nich wartości formuły i ewentualnie jej podformuł.

p	q	$q \Rightarrow p$	$p \Rightarrow (q \Rightarrow p)$
F	F	P	P
F	P	F	P
P	F	P	P
P	P	P	P

Ponieważ formuła jest spełniona przy dowolnym wartościowaniu występujących w niej zmiennych, jest więc tautologią.

Metoda zero-jedynkowa oblicza wartości formuły dla wszystkich możliwych wartościowań jej zmiennych. Ponieważ liczba takich wartościowań jest skończona, zatem po skończonej liczbie kroków otrzymuje się niezawodną odpowiedź na pytanie, czy formuła jest tautologią. Metoda zero-jedynkowa daje więc zawsze podstawę do stwierdzenia, czy dana formuła rachunku zdań jest czy nie jest tautologią. Problem badania czy formuła jest tautologią jest *rozstrzygalny*. Ogólnie, pojęcie rozstrzygalności danego problemu – pytania, na które odpowiedzią jest *tak* albo *nie* – oznacza, że istnieje procedura (algorytm), która w skończonej liczbie kroków daje jedną z tych odpowiedzi.

Metoda zero-jedynkowa jest mało efektywna. Można ją usprawnić, zauważając, że obliczenie wartości *fałsz* dla pewnego wartościowania przesądza, iż formuła nie może być tautologią. Obliczenia wartości formuły można zakończyć w momencie pierwszego napotkania takiego wartościowania.

Dla rachunku zdań istnieją jeszcze inne, efektywniejsze sposoby rozstrzygania, czy formuła jest tautologią.

9.4. Wybrane tautologie

Tautologie są schematami formuł, które są zawsze prawdziwe, niezależnie od wyrażanych treści. Są one prawdziwe z uwagi na swoją strukturę. Poniżej przedstawiono częściej spotykane tautologie. Są one wykorzystywane w dowodach matematycznych i dlatego nazywa się je *prawami logicznymi* albo *prawami rachunku zdań*. Niektóre z nich mają też tradycyjne nazwy. Jeżeli α oraz β są dowolnymi formułami, to tautologiami są formuły:

Prawo implikacji

$$\models \alpha \Rightarrow \beta \Leftrightarrow \neg\alpha \vee \beta$$

Prawa kontrapozycji

$$\models \neg\alpha \Rightarrow \beta \Leftrightarrow \neg\beta \Rightarrow \alpha$$

$$\models \alpha \Rightarrow \neg\beta \Leftrightarrow \beta \Rightarrow \neg\alpha$$

Prawa de Morgana

$$\models \neg(\alpha \wedge \beta) \Leftrightarrow \neg\alpha \vee \neg\beta$$

$$\models \neg(\alpha \vee \beta) \Leftrightarrow \neg\alpha \wedge \neg\beta$$

Prawa zaprzeczenia implikacji

$$\models \neg(\alpha \Rightarrow \beta) \Leftrightarrow \alpha \wedge \neg\beta$$

Prawa zaprzeczenia równoważności

$$\models \neg(\alpha \Leftrightarrow \beta) \Leftrightarrow \neg(\alpha \Rightarrow \beta) \vee \neg(\beta \Rightarrow \alpha)$$

Prawa podwójnego zaprzeczenia

$$\models \neg\neg\alpha \Leftrightarrow \alpha$$

Prawo wyłączonego środka

$$\models \alpha \vee \neg\alpha \Leftrightarrow \mathbf{true}$$

Prawo sprzeczności

$$\models \alpha \wedge \neg\alpha \Leftrightarrow \mathbf{false}$$

Prawa idempotentności

$$\models \alpha \wedge \alpha \Leftrightarrow \alpha$$
$$\models \alpha \vee \alpha \Leftrightarrow \alpha$$

Prawa przemienności

$$\models \alpha \wedge \beta \Leftrightarrow \beta \wedge \alpha$$
$$\models \alpha \vee \beta \Leftrightarrow \beta \vee \alpha$$

Prawa łączności

$$\models \alpha \wedge (\beta \wedge \gamma) \Leftrightarrow (\alpha \wedge \beta) \wedge \gamma$$
$$\models \alpha \vee (\beta \vee \gamma) \Leftrightarrow (\alpha \vee \beta) \vee \gamma$$

Prawa rozdzielności

$$\models \alpha \wedge (\beta \vee \gamma) \Leftrightarrow \alpha \wedge \beta \vee \alpha \wedge \gamma$$
$$\models \alpha \vee (\beta \wedge \gamma) \Leftrightarrow (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$$

Prawa uproszczeń

$$\models \alpha \wedge \mathbf{true} \Leftrightarrow \alpha$$
$$\models \alpha \vee \mathbf{true} \Leftrightarrow \mathbf{true}$$
$$\models \alpha \wedge \mathbf{false} \Leftrightarrow \mathbf{false}$$
$$\models \alpha \vee \mathbf{false} \Leftrightarrow \alpha$$
$$\models \alpha \wedge (\alpha \vee \beta) \Leftrightarrow \alpha$$
$$\models \alpha \vee (\alpha \wedge \beta) \Leftrightarrow \alpha$$

Fakt, że przedstawione prawa są tautologiami łatwo sprawdzić metodą zero-jedynkową. Na podstawie tych praw oraz twierdzenia o zastępowaniu można badać czy tautologiami są inne formuły.

9.5. Dowodzenie transformacyjne

Niech α będzie formułą, w której występuje zmienna zdaniowa p, oraz niech β będzie pewną inną formułą. Przez

$$\alpha[p ::= \beta]$$

oznacza się formułę, która powstaje z formuły α przez tekstowe zastąpienie każdego wystąpienia zmiennej p w formule α przez formułę β.

Przykład 9.3

Jeżeli

$$\alpha \equiv (p \Leftrightarrow q) \wedge p \Rightarrow r \text{ oraz } \beta \equiv r \vee s,$$

to

$$\alpha[p ::= \beta] \equiv ((r \vee s) \Leftrightarrow q) \wedge (r \vee s) \Rightarrow r$$

Bezpośrednio z definicji tautologii i tekstowego zastępowania wystąpienia zmiennej wynika następujące użyteczne twierdzenie o zastępowaniu.

Twierdzenie 9.1

Jeżeli formuła $\alpha_1 \Leftrightarrow \alpha_2$ jest tautologią, β pewną formułą oraz p – zmienną zdaniową, to formuła

$$\beta[p ::= \alpha_1] \Leftrightarrow \beta[p ::= \alpha_2]$$

jest także tautologią.

Na podstawie twierdzenia o zastępowaniu formułuje się następujące reguły równoważnego semantycznie przekształcania formuł:

Reguła zastąpienia

Jeżeli α jest formułą, a β jest jej podformułą, to zastąpienie podformuły β dowolną inną równoważną semantycznie formułą nie zmienia wartości logicznej formuły α.

W zapisie symbolicznym, zgodnym z konwencją wprowadzoną w rozdziale 1., regułę można przedstawić w postaci

$$\frac{\beta \text{ jest podformułą } \alpha \qquad \beta = \gamma}{\alpha = \alpha[\beta ::= \gamma]}$$

lub w równoważnej postaci

β jest podformułą α

$$\frac{\models \beta \Leftrightarrow \gamma}{\models \alpha \Leftrightarrow \alpha[\beta ::= \gamma]}$$

gdzie $\alpha[\beta ::= \gamma]$ oznacza testowe zastąpienie podformuły β formułą γ w formule α.

Przykład 9.4

Niech dana będzie formuła

$$\alpha \equiv (p \Leftrightarrow q) \wedge p \Rightarrow r.$$

Jej podformuła

$$\beta \equiv p \Leftrightarrow q$$

jest równoważna semantycznie formule

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

Formuła α jest zatem równoważna semantycznie formule

$$(p \Rightarrow q) \wedge (q \Rightarrow p) \wedge p \Rightarrow r$$

czyli

$$(p \Leftrightarrow q) \wedge p \Rightarrow r = (p \Rightarrow q) \wedge (q \Rightarrow p) \wedge p \Rightarrow r$$

Reguła przechodniości

Jeżeli formuły α i β są równoważne semantycznie oraz formuły β i γ są równoważne semantycznie tautologiami, to również formuły α i γ są równoważne semantycznie.

W zapisie symbolicznym regułę można przedstawić w postaci

$$\frac{\alpha = \beta \quad \beta = \gamma}{\alpha = \gamma}$$

lub w równoważnej postaci

$$\frac{\models \alpha \Leftrightarrow \beta \quad \models \beta \Leftrightarrow \gamma}{\models \alpha \Leftrightarrow \gamma}$$

Reguły zastąpienia i przechodniości pozwalają na tekstowe transformacje formuł, które można wykorzystać do badania równoważności lub badania czy formuła jest tautologią.

Ogólny schemat dowodzenia transformacyjnego, w celu pokazania, że formuła α jest tautologią, można wyrazić w postaci ciągu formuł

$$\alpha_0, \alpha_1, \alpha_2, ..., \alpha_n$$

gdzie: $\alpha_0 \equiv \alpha$ jest dowodzoną formułą, α_n jest formułą, o której wiadomo, że jest tautologią, pomiędzy zaś kolejnymi formułami α_i, dla $i = 0, 1, ..., n-1$, zachodzą równoważności semantyczne $\alpha_i = \alpha_{i+1}$. Równoważności te zachodzą na mocy definicji lub w wyniku stosowania reguły zastąpienia z powołaniem się na odpowiednie prawa logiczne.

Przykład 9.5

Dana jest formuła

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

Aby pokazać, że jest ona tautologią, należy wykazać, że jest równoważna formule **true**. Prowadzi do tego następujący ciąg transformacji:

$\neg q \wedge (p \Rightarrow q) \Rightarrow \neg p$	
$= \neg q \wedge (\neg p \vee q) \Rightarrow \neg p$	– prawo implikacji
$= (\neg q \wedge \neg p) \vee (\neg q \wedge q) \Rightarrow \neg p$	– prawo rozdzielności
$= (\neg q \wedge \neg p) \vee \mathbf{false} \Rightarrow \neg p$	– prawo sprzeczności
$= \neg q \wedge \neg p \Rightarrow \neg p$	– prawo uproszczenia
$= \neg(q \vee p) \Rightarrow \neg p$	– prawo de Morgana
$= \neg(\neg(q \vee p)) \vee \neg p$	– prawo implikacji
$= (q \vee p) \vee \neg p$	– prawo podwójnej negacji
$= q \vee (p \vee \neg p)$	– prawo łączności
$= q \vee \mathbf{true}$	– prawo uproszczenia
$= \mathbf{true}$	– prawo uproszczenia

Każdy krok transformacji jest przeprowadzony zgodnie z regułą zastąpienia wykorzystującą wskazane prawo logiczne. Na mocy reguły przechodniości stwierdza się, że $\neg q \wedge (p \Rightarrow q) \Rightarrow \neg p = \mathbf{true}$, co oznacza, że formuła $\neg q \wedge (p \Rightarrow q) \Rightarrow \neg p$ jest tautologią.

Formuła, którą dowiedziono, jest prawem logicznym, nazywanym *modus tollens*.

9.6. Postaci kanoniczne formuł

W wielu zastosowaniach jest wygodne, aby formuły miały pewną standardową (kanoniczną) postać. Pozwala to między innymi na ułatwienie badania równoważności formuł. Wyróżnia się dwa rodzaje postaci kanonicznych – koniunkcyjną postać normalną i dysjunkcyjną postać normalną.

Literałem nazywa się zmienną zdaniową lub jej negację. Jeżeli p, q, ... są zmiennymi zdaniowymi, to p, q, ... są literałami *pozytywnymi*, a $\neg p$, $\neg q$, ... są literałami *negatywnymi*. Pojedynczy literał będzie oznaczany symbolem λ.

Klauzulą albo *dysjunkcją elementarną* będzie nazywana formuła postaci

$$\lambda_1 \vee \lambda_2 \vee \ldots \vee \lambda_n$$

gdzie: λ_1, λ_2, ..., λ_n są literałami ($n \geq 1$). Pojedyncza klauzula będzie oznaczana symbolem κ.

Formuła α jest w *koniunkcyjnej postaci normalnej* (*CNF – Conjunctive Normal Form*) wtedy i tylko wtedy, gdy jest koniunkcją klauzul, to znaczy gdy jest postaci

$$\kappa_1 \wedge \kappa_2 \wedge \ldots \wedge \kappa_n$$

gdzie: κ_i, dla $i = 1, \ldots, n$, są klauzulami.

Przykład 9.6

Jeżeli dane są trzy zmienne zdaniowe p, q, r, to wyznaczają one osiem różnych semantycznie klauzul. Są to:

$p \vee q \vee r$ $\quad p \vee q \vee \neg r$ $\quad p \vee \neg q \vee r$ $\quad p \vee \neg q \vee \neg r$
$\neg p \vee q \vee r$ $\quad \neg p \vee q \vee \neg r$ $\quad \neg p \vee \neg q \vee r$ $\quad \neg p \vee \neg q \vee \neg r$

Z tekstowego punktu widzenia klauzul zawierających trzy zmienne jest więcej. Każda z innych klauzul jest równoważna semantycznie jednej z wyżej wymienionych. Na przykład klauzule $q \vee p \vee \neg r$ oraz $p \vee \neg r \vee q$ są równoważne klauzuli

$$p \vee q \vee \neg r.$$

Koniunkcją elementarną będzie nazywana formuła postaci

$$\lambda_1 \wedge \lambda_2 \wedge \ldots \wedge \lambda_n$$

gdzie: λ_1, λ_2, ..., λ_n są literałami ($n \geq 1$). Pojedyncza koniunkcja elementarna będzie oznaczana symbolem δ.

Formuła α jest w *dysjunkcyjnej postaci normalnej* (*DNF – Disjunctive Normal Form*) wtedy i tylko wtedy, gdy jest dysjunkcją koniunkcji elementarnych, to znaczy gdy jest postaci

$$\delta_1 \vee \delta_2 \vee \ldots \vee \delta_n$$

gdzie δ_i, dla $i = 1, \ldots, n$, są koniunkcjami elementarnymi.

Przykład 9.7

Jeżeli dane są trzy zmienne zdaniowe p, q, r, to wyznaczają one osiem różnych koniunkcji elementarnych. Są to:

$p \wedge q \wedge r$ $p \wedge q \wedge \neg r$ $p \wedge \neg q \wedge r$ $p \wedge \neg q \wedge \neg r$

$\neg p \wedge q \wedge r$ $\neg p \wedge q \wedge \neg r$ $\neg p \wedge \neg q \wedge r$ $\neg p \wedge \neg q \wedge \neg r$

Dla każdej formuły α istnieje równoważna jej semantycznie formuła w koniunkcyjnej postaci normalnej oraz w dysjunkcyjnej postaci normalnej – formuły te będą oznaczane odpowiednio przez $CNF(\alpha)$ oraz $DNF(\alpha)$.

Uzasadnieniem tych twierdzeń jest przedstawiony poniżej, który dla dowolnej formuły α wyznacza nową formułę w koniunkcyjnej postaci normalnej, oznaczaną $CNF(\alpha)$, która jest równoważna semantycznie formule α, czyli $\alpha = CNF(\alpha)$.

Algorytm sprowadzania formuł do koniunkcyjnej postaci normalnej

Dane: dowolna formuła $\alpha \in FORM$.

Wynik: formuła $CNF(\alpha) \in FORM$ taka, że $\alpha = CNF(\alpha)$.

Procedura: procedura postępowania polega na etapowym, tekstowym przekształcaniu formuły α. Formuła pośrednia jest oznaczana przez β, początkowo przyjmie postać formuły α.

1. Eliminacja z formuły β spójników logicznych różnych od koniunkcji, dysjunkcji i negacji:
   - każdą podformułę formuły β, postaci $\beta_1 \Leftrightarrow \beta_2$, zastępuje się tekstowo formułą postaci $(\beta_1 \Rightarrow \beta_2) \wedge (\beta_2 \Rightarrow \beta_1)$,
   - każdą podformułę postaci $\beta_1 \Rightarrow \beta_2$ zastępuje się tekstowo formułą postaci $\neg\beta_1 \vee \beta_2$.
2. Dopóki β nie jest w postaci koniunkcyjnej normalnej, dopóty powtarza się zastępowanie podformuł formuły β zgodnie z regułami podanym w tablicy:

Lp.	Podformuła zastępowana	Formuła zastępująca
1	$\neg\neg\beta_1$	β_1
2	$\neg(\beta_1 \vee \beta_2)$	$\neg\beta_1 \wedge \neg\beta_2$
3	$\neg(\beta_1 \wedge \beta_2)$	$\neg\beta_1 \vee \neg\beta_2$
4	$\beta_1 \vee (\beta_2 \wedge \beta_3)$	$(\beta_1 \vee \beta_2) \wedge (\beta_1 \vee \beta_3)$

3. Formułę β, otrzymaną po zakończeniu poprzedniego kroku, definiuje się jako $CNF(\alpha)$.

Algorytm dokonuje na formułach przekształceń semantycznie równoważnych, a ponadto rozpatruje wszystkie niezbędne przypadki, co gwarantuje jego poprawność.

Algorytm sprowadzania formuły do normalnej postaci dysjunkcyjnej jest prostą modyfikacją podanego wyżej algorytmu. Polega to na następującej zamianie ostatniego wiersza w tablicy zastępowania formuł:

Lp.	Formuła zastępowana	Formuła zastępująca
4a	$\beta_1 \wedge (\beta_2 \vee \beta_3)$	$(\beta_1 \wedge \beta_2) \vee (\beta_1 \wedge \beta_3)$

Należy zwrócić uwagę, że w przypadkach stosowania ostatniej z reguł zastępowania (przypadek 4 lub 4a) algorytm powoduje tekstowe wydłużenie przekształcanej formuły. W niektórych przypadkach algorytm może prowadzić do zwiększenia długości formuły.

9.7. Funkcjonalna pełność

Wprowadzony język rachunku zdań używa zbioru spójników złożonego z negacji, koniunkcji, dysjunkcji, implikacji i równoważności. W poprzednim podrozdziale pokazano algorytm sprowadzania formuły do postaci kanonicznej, w której występują tylko spójniki negacji, koniunkcji i dysjunkcji. Oznacza to, że dla dowolnej formuły rachunku zdań istnieje równoważna semantycznie formuła zawierająca tylko te trzy spójniki.

Stwierdzenie to można wyrazić w sposób ogólniejszy, mówiąc, że za pomocą tych spójników można wyrazić dowolną n-argumentową, $n > 0$, funkcję prawdziwościową, to jest funkcję typu: $Logiczne^n \rightarrow Logiczne$.

Dany zbiór spójników logicznych jest *funkcjonalnie pełny*, jeżeli za ich pomocą da się wyrazić wszystkie możliwe funkcje prawdziwościowe, to znaczy że dowolną funkcję prawdziwościową da się przedstawić jako formułę, w której występują spójniki logiczne należące do tego zbioru.

Twierdzenie 9.2 $\{\neg, \wedge, \vee\}$

Zbiór spójników złożony z negacji, koniunkcji i dysjunkcji jest funkcjonalnie pełny.

Dowód

Szkic dowodu przedstawia się następująco:

Zakłada się, że $f(p_1,..., p_n)$ jest dowolną n-argumentową funkcją prawdziwościową. Niech $INT_v(f(p_1, ..., p_n))$ oznacza interpretację funkcji f dla wartościowania v. Interpretacją funkcji f dla wartościowania v jest jedna z wartości **P** lub **F**. Pojedyncze wartościowanie v przypisuje każdej ze zmiennych $p_1, ..., p_n$ jedną z wartości **P** lub **F**.

Jeżeli $INT_v(f(p_1, ..., p_n)) = \mathbf{P}$, to koniunkcję elementarną

$$\lambda_1 \wedge \lambda_2 \wedge ... \wedge \lambda_n$$

określa się w następujący sposób

$\lambda_i = p_i$, gdy $v(p_i) = \mathbf{P}$ oraz $\lambda_i = \neg p_i$, gdy $v(p_i) = \mathbf{F}$.

Łatwo zauważyć, że koniunkcja ta jest prawdziwa dla wartościowania v i fałszywa dla każdego innego wartościowania.

Niech $v_1, ..., v_K$, gdzie K spełnia ograniczenie $0 \leq K \leq 2^n$, będzie zbiorem tych wszystkich wartościowań, dla których funkcja f przyjmuje wartość **P**. Dla danego wartościowania v_j, dla $j = 1, ..., K$, przez δ_j oznacza się wyżej określoną koniunkcję elementarną. Łatwo sprawdzić, że formuła

$$\alpha \equiv \delta_1 \vee \delta_2 \vee ... \vee \delta_K$$

jest semantycznie równoważna funkcji f, to znaczy dla dowolnego wartościowania v, $INT_v(f(p_1, ..., p_n)) = INT_v(\alpha)$. Ponieważ w formule występują tylko spójniki negacji, koniunkcji i dysjunkcji, stąd wynika teza. ■

Funkcjonalnie pełny zbiór spójników jest *minimalny*, jeżeli każdy jego właściwy podzbiór nie jest zbiorem funkcjonalnie pełnym.

Zbiór spójników $\{\neg, \wedge, \vee\}$ nie jest zbiorem minimalnym. Oznacza to, że po usunięciu z niego pewnych spójników pozostanie on nadal zbiorem funkcjonalnie pełnym. Łatwo się przekonać, że zbiorami minimalnymi są zbiory spójników $\{\neg, \vee\}$ oraz $\{\neg, \wedge\}$. Z praw de Morgana wynika, że na przykład koniunkcję można wyrazić za pomocą dysjunkcji. Zbiór $\{\neg, \wedge\}$ jest zatem funkcjonalnie pełny. Za pomocą tylko samej negacji albo tylko samej koniunkcji nie można natomiast wyrazić dowolnej formuły.

Innym przykładem minimalnego zbioru funkcjonalnie pełnego jest zestaw $\{\Rightarrow, \textbf{false}\}$. Występuje w nim jeden spójnik i jedna stała.

Dwa interesujące przykłady minimalnych, funkcjonalnie pełnych zbiorów spójników są oparte na spójnikach *NAND* lub *NOR*, które są zdefiniowane następująco:

$$NAND(p, q) =_{def} \neg(p \wedge q)$$
$$NOR(p, q) =_{def} \neg(p \vee q)$$

Łatwo pokazać, że za ich pomocą można zdefiniować wcześniej wprowadzone spójniki, na przykład:

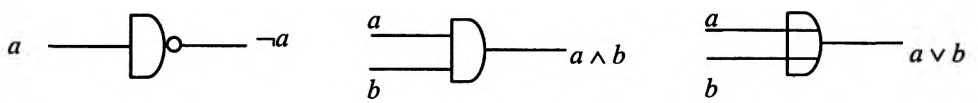
$$\neg p = NAND(p, p)$$
$$\neg p = NOR(p, p)$$

Spójniki są interesujące, między innymi dlatego, że opierając się na każdym z nich, można budować układy przełączające – fragmenty urządzeń komputerowych.

Przykład 9.8

Stosowane w konstrukcji urządzeń komputerowych kombinacyjne cyfrowe układy przełączające charakteryzują się pewną liczbą wejść, na które podaje się dwa sygnały: zero albo jeden, oraz przynajmniej jednym wyjściem, na którym rów-

nież pojawia się taki sygnał. Wartość sygnału wyjściowego jest funkcją sygnałów wejściowych. Sygnały o wartościach zero i jeden mogą kodować wartości logiczne fałsz i prawda. Wyjście układu można więc scharakteryzować przez funkcję prawdziwościową, której argumentami są wejścia układu. Układ realizuje się za pomocą układów elementarnych. Przykładem zestawu elementarnych bramek logicznych, za pomocą których można zrealizować dowolny układ przełączający, są bramki nazywane NOT, AND, OR, pokazane na rysunku 9.1, które są realizatorami spójników $\neg$, $\wedge$, $\vee$.



Bramka NOT Bramka AND Bramka OR

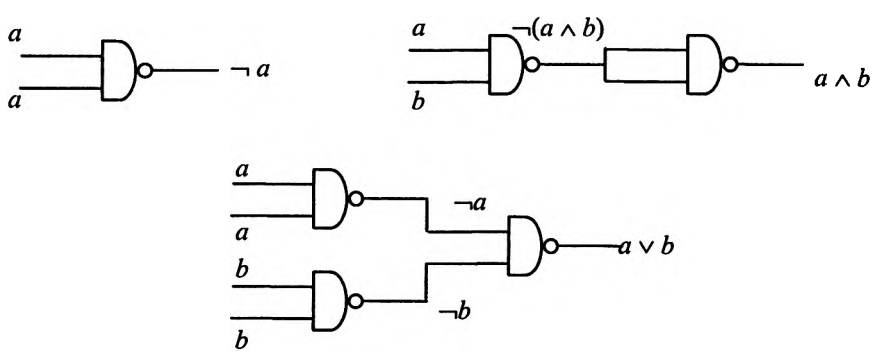
Rys. 9.1. Schematy bramek logicznych NOT, AND i OR

Każdą z bramek można zbudować za pomocą bramek NAND lub NOR, które są realizatorami spójników *NAND* oraz *NOR* (rys. 9.2).

Bramka NAND Bramka NOR

Rys. 9.2. Schematy bramek NAND i NOR

Używając na przykład bramki NAND, otrzymujemy konstrukcje pokazane na rysunku 9.3.



Rys. 9.3. Przykładowe schematy układów logicznych

9.8. Rekursja i indukcja strukturalna

Rekursja jest ważnym i często wykorzystywanym sposobem definiowania zbiorów (rozdział 2.), relacji i funkcji (podrozdział 4.6). Podane niżej twierdzenie dotyczy rekursywnego definiowania funkcji określonych na zbiorze formuł rachunku zdań. Twierdzenie to gwarantuje jednoznaczność rekursywnie definiowanych funkcji.

Twierdzenie 9.3 (*Zasada rekursji strukturalnej*)

Niech pewna funkcja f będzie określona na zbiorze formuł *FORM* w sposób następujący:

krok początkowy: na formułach elementarnych wartości funkcji f są określone bezpośrednio,

kroki indukcyjne: na formułach złożonych wartości funkcji f są określone pośrednio:

- wartość funkcji f na formule $\neg\alpha$ jest określona w terminach wartości funkcji f na α,
- wartość funkcji f na formule $(\alpha \circ \beta)$ jest określona w terminach wartości funkcji f na formułach α i β, gdzie $\circ$ oznacza dowolny binarny spójnik logiczny.

Funkcja f jest zdefiniowana jednoznacznie (istnieje dokładnie jedna tak zdefiniowana funkcja).

Dowód twierdzenia pomijamy.

Przykład 9.9

Stosując zasadę rekursji strukturalnej, na zbiorze formuł *FORM* definiuje się następującą funkcję d:

- jeżeli α jest formuła elementarną, to $d(\alpha) = 0$,
- $d(\neg\alpha) = d(\alpha) + 1$,
- $d((\alpha \circ \beta)) = d(\alpha) + d(\beta) + 1$ dla dowolnego spójnika binarnego $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$.

Funkcja $d(\alpha)$ określa *stopień* formuły α. Stopień formuły, jak łatwo zauważyć, oznacza liczbę spójników logicznych w formule.

Przykład 9.10

Stosując zasadę rekursji strukturalnej, definiuje się na zbiorze formuł *FORM* następujące funkcje $l(\alpha)$ oraz $p(\alpha)$, oznaczające odpowiednio liczbę lewych i prawych nawiasów w formule α. Definicja funkcji $l(\alpha)$:

- jeżeli α jest formułą elementarną, to $l(\alpha) = 0$

- $l(\neg\alpha) = l(\alpha)$
- $l((\alpha \circ \beta)) = l(\alpha) + l(\beta) + 1$ dla dowolnego spójnika binarnego $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$.

Podobnie wygląda definicja funkcji $p(\alpha)$.

Rekursja strukturalna została tu przedstawiona tylko dla rachunku zdań. Ogólnie rekursję strukturalną można stosować do dowolnych zbiorów definiowanych w sposób rekursywny, zwłaszcza do języków formalnych definiowanych za pomocą gramatyki bezkontekstowej.

Przykład 9.11

Dana jest gramatyka $G =_{def} \langle T, N, P, S\rangle$, gdzie:

$T =_{def} \{0, 1, @, \#, +, *, (,)\} \cup \{a, b, ..., z\}$

$N =_{def} \{wyr, op\_unarny, op\_binarny, zmienna\}$

$P =_{def} \{wyr ::= 0 \mid 1 \mid zmienna \mid (wyr\ op\_binarny\ wyr) \mid op\_unarny\ wyr$

$zmienna ::= a \mid b \mid ... \mid z$

$op\_binarny ::= + \mid *$

$op\_unarny ::= @ \mid \#\}$

$S =_{def} wyr$

Funkcja $lw : L(G) \times \{a, b, ..., z\} \to Nat$, która oblicza liczbę wystąpień wskazanej zmiennej w słowie języka generowanego przez gramatykę G, jest zdefiniowana następująco:

a) dla wyrażeń elementarnych:

$$lw(0, x) = lw(1, x) = 0 \qquad \text{dla } x \in \{a, b, ..., z\}$$

$$lw(y, x) = \begin{cases} 1 & \text{dla } y = x \\ 0 & \text{dla } y \neq x \end{cases} \qquad \text{dla } x, y \in \{a, b, ..., z\}$$

b) dla wyrażeń złożonych:

$$lw((\alpha \circ \beta), x) = lw(\alpha, x) + lw(\beta, x) \qquad \text{dla } \alpha, \beta \in L(G), \circ \in \{+, *\}$$

$$lw(\propto \alpha, x) = lw(\alpha, x) \qquad \text{dla } \alpha \in L(G), \propto \in \{@, \#\}$$

Omówiona w rozdziale 1. indukcja matematyczna dotyczyła sposobu dowodzenia własności, które zachodzą dla wszystkich liczb naturalnych. Zbiór liczb naturalnych jest liniowo uporządkowanym zbiorem przeliczalnym. Liniowy porządek wyznacza relacja większości pomiędzy liczbami naturalnymi. Często interesują nas własności, które zachodzą dla innych zbiorów przeliczalnych, ale nieuporządkowanych liniowo. Przykładem takiego zbioru jest zbiór wszystkich formuł rachunku zdań. Dowodzenie własności postaci $P(\alpha)$, gdzie $\alpha \in FORM$, opiera się na indukcji strukturalnej, która jest uogólnieniem indukcji matematycznej.

Twierdzenie 9.4 (*Zasada indukcji strukturalnej dla rachunku zdań*)

Niech P będzie pewną własnością dotyczącą formuł. Własność $P(\alpha)$ ma każda formuła α rachunku zdań, pod warunkiem, że:

krok początkowy: własność tę ma każda formuła elementarna,

krok indukcyjny:

- jeżeli własność tę ma formuła α, to ma ją także formuła $\neg\alpha$,
- jeżeli własność tę mają formuły α oraz β, to ma ją także formuła $(\alpha \circ \beta)$, gdzie $\circ$ oznacza dowolny binarny spójnik logiczny.

Dowód

Uzasadnieniem dla podanego postępowania jest następujące rozważanie: Niech S będzie zbiorem tych formuł rachunku zdań, które mają własność P. Krok początkowy i kroki indukcyjne stwierdzają, że formuły należące do S spełniają warunki:

- jeżeli α jest formułą elementarną, to $\alpha \in S$,
- jeżeli $\alpha \in P$, to $\neg\alpha \in P$
- jeżeli $\alpha, \beta \in P$, to $(\alpha \circ \beta) \in S$, gdzie $\circ$ jest dowolnym binarnym spójnikiem logicznym.

Ponieważ zbiór formuł *FORM* jest najmniejszym zbiorem spełniającym wyżej wymienione warunki, zatem zbiór formuł $FORM \subseteq S$, z czego wynika, że każda formuła ma własność P. ■

Przykład 9.12

Rozpatruje się własność P: w dowolnej formule rachunku zdań liczba nawiasów otwierających jest równa liczbie nawiasów zamykających. Przez $lewy(\alpha)$ oraz $prawy(\alpha)$ oznacza się liczby nawiasów otwierających i zamykających w formule α. Własność $P(\alpha)$ może być zapisana $lewy(\alpha) = prawy(\alpha)$.

- Formuły elementarne nie zawierają nawiasów, mają zatem własność P.
- Zakłada się, że własność P mają dowolne formuły α, β. Oznacza to, że $lewy(\alpha) = prawy(\alpha)$ oraz $lewy(\beta) = prawy(\beta)$. Rozpatruje się dowolną formułę złożoną: $(\alpha \circ \beta)$, gdzie $\circ$ jest dowolnym binarnym spójnikiem logicznym. Własności P zachodzą więc również dla $(\alpha \circ \beta)$:

$$\begin{aligned} lewy((\alpha \circ \beta)) &= lewy(\alpha) + lewy(\beta) + 1 \\ &= prawy(\alpha) + prawy(\beta) + 1 \\ &= prawy((\alpha \circ \beta)) \end{aligned}$$

Zasada indukcji strukturalnej została tu zdefiniowana tylko dla rachunku zdań. Jest ona również stosowana w rachunku kwantyfikatorów (zob. następny rozdział).

Ćwiczenia

1. Wskazać ciągi znaków, które są słowami języka rachunku zdań:
   a) $((p \vee q))$
   b) $p \vee q$
   c) $q \vee p \Leftrightarrow (\vee(q, p))$

2. Daną formułę rachunku zdań przedstawić w pełnej postaci z nawiasami, a następnie określić zbiór wszystkich jej podformuł:
   a) $a \wedge b \wedge c \vee d \Leftrightarrow e \Rightarrow \neg f \vee g \Rightarrow h$
   b) $a \wedge (b \wedge c \vee d) \Leftrightarrow e \Rightarrow (\neg f \vee g) \Rightarrow h$

3. Podać algorytm, który dowolną formułę rachunku zdań zapisaną w postaci wrostkowej transformuje na formułę zapisaną w postaci przedrostkowej.

4. Funktorem zdaniowym n-argumentowym nazywamy dowolną funkcję f o sygnaturze
   f: $\{prawda, fałsz\}^n \rightarrow \{prawda, fałsz\}$.
   Jaka jest liczba takich funktorów n-argumentowych? Zdefiniować wszystkie funktory jedno- i dwuargumentowe.

5. Stosując metodę zero-jedynkową, wykazać, że następujące formuły są tautologiami:
   a) $p \Rightarrow (q \Rightarrow p)$
   b) $(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$
   c) $(p \vee q) \Leftrightarrow (\neg p \wedge \neg q)$

6. Opierając się na systemie dowodzenia opartym tylko na regułach podstawienia i przechodniości, pokazać, że następujące formuły są tautologiami:
   a) $\neg a \wedge (a \Rightarrow b) \Rightarrow \neg a$
   b) $a \wedge (a \Rightarrow b) \Rightarrow b \Leftrightarrow$ **true**

7. Sprawdź (w dowolny sposób), czy są tautologiami następujące formuły:
   a) $p \vee (q \wedge r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$
   b) $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$
   c) $\alpha \Leftrightarrow \beta$
   d) $\alpha \vee \neg \beta$
   e) $\neg \alpha \Rightarrow \beta$

8. Które zbiory spójników logicznych są zbiorami funkcjonalnie pełnymi:
   a) $\{\neg, \wedge\}$
   b) $\{\neg, \vee\}$
   c) $\{\neg, \Rightarrow\}$
   d) {**false**, $\Rightarrow$}
   e) $\{\neg, \Leftrightarrow\}$

f) {true, $\Rightarrow$}
g) {$\Leftrightarrow$, **false**}

9. Dane są dwa dwuargumentowe funktory logiczne *NAND* oraz *NOR* zdefiniowane następująco:

$NAND(a, b) = \neg(a \wedge b)$ oraz $\mathrm{NOR}(a, b) = \neg(a \vee b)$.

Pokazać, w jaki sposób, za pomocą tych funktorów, można wyrazić spójniki logiczne negacji, koniunkcji i alternatywy. Narysować sieci logiczne realizujące funkcje prawdziwościowe f_1, f_2 zdefiniowane przedstawioną poniżej tablicą, w której symbolami 0 oraz 1 oznaczono odpowiednio fałsz oraz prawdę.

a	b	$f_1(a,b)$	$f_2(a,b)$
0	0	1	1
0	1	0	0
1	0	1	1
1	1	0	1

10. Która ze zdefiniowanych niżej relacji jest relacją równoważności na zbiorze formuł rachunku zdań:
a) $\alpha \sim_1 \beta$ wtedy i tylko wtedy, gdy formuła $\alpha \Leftrightarrow \beta$ jest spełniona,
b) $\alpha \sim_2 \beta$ wtedy i tylko wtedy, gdy formuła $\alpha \Leftrightarrow \beta$ jest sprzeczna,
c) $\alpha \sim_3 \beta$ wtedy i tylko wtedy, gdy formuła $\alpha \Leftrightarrow \beta$ jest spełniona dokładnie dla połowy wartościowań zmiennych.

11. Niech γ będzie dowolnie ustaloną formułą rachunku zdań. Wykazać, że relacja zdefiniowana następująco:

$\alpha \sim \beta$ wtedy i tylko wtedy, gdy formuła $\gamma \Rightarrow (\alpha \Leftrightarrow \beta)$ jest tautologią,
jest relacją równoważności na zbiorze formuł rachunku zdań.

12. Dana jest gramatyka $G =_{def} \langle T, N, P, S \rangle$, gdzie:

$T =_{def} \{0, 1, @, \#, +, *, (,)\}$
$N =_{def} \{wyr, op\_unarny, op\_binarny\}$
$P =_{def} \{wyr ::= 0 \mid 1 \mid (wyr\ op\_binarny\ wyr) \mid op\_unarny\ wyr$
$op\_binarny ::= + \mid *$
$op\_unarny ::= @ \mid \#\}$
$S =_{def} wyr$

a) Czy gramatyka jest jednoznaczna?
b) Podać przykład wyprowadzenia dowolnego słowa generowanego przez gramatykę G o długości większej od 2.
c) Stosując zasadę rekursji strukturalnej, zdefiniować funkcję $left(\alpha)$, która dla dowolnego napisu $\alpha \in L(G)$ określa liczbę lewostronnych nawiasów występujących w napisie α.

d) Stosując zasadę indukcji strukturalnej, pokazać, że dla dowolnego napisu $\alpha \in L(G)$ zachodzi następująca własność:

$left(\alpha)$ = liczba operatorów binarnych występujących w napisie α.

13. Dla gramatyki G z przykładu 9.11 zdefiniować funkcję, która oblicza liczbę:
    a) wystąpień zmiennych w słowie języka generowanego przez gramatykę G,
    b) liczbę różnych zmiennych występujących w słowie języka generowanego przez gramatykę G.

14. Następujące formuły oraz ich negacje sprowadzić do koniunkcyjnej (*CNF*) i do dysjunkcyjnej (*DNF*) postaci normalnej:
    a) $((a \Rightarrow b) \vee c) \Leftrightarrow (b \wedge c)$,
    b) $\neg(a \wedge b) \Rightarrow (b \vee \neg c)$,
    c) $(a \Rightarrow b) \vee (b \Rightarrow a)$.

10. Rachunek kwantyfikatorów

10.1. Składnia

Rachunek kwantyfikatorów jest uogólnieniem rachunku zdań. Język formalny rachunku kwantyfikatorów jest zdefiniowany jako zbiór napisów dwóch kategorii – termów i formuł – nad alfabetem, na który składają się następujące kategorie *jednostek leksykalnych*:

- przeliczalny zbiór V symboli *zmiennych indywiduowych*, reprezentowanych przez identyfikatory; dalej najczęściej będą używane symbole: x, y, ...,
- przeliczalny zbiór F_n symboli *funkcyjnych* n-argumentowych dla $n \in Nat$, reprezentowanych przez identyfikatory; dalej najczęściej będą używane symbole: c – dla symboli funkcyjnych zeroargumentowych, czyli dla stałych indywiduowych, oraz f, g, ... – dla pozostałych symboli funkcyjnych, zbiór wszystkich symboli funkcyjnych będzie oznaczany $F = \bigcup_{n \in Nat} F_n$,
- przeliczalny zbiór P_n symboli *predykatów* n-argumentowych dla $n \in Nat$, reprezentowanych przez identyfikatory; predykaty zeroargumentowe (co najwyżej dwa) są nazywane stałymi logicznymi; dalej najczęściej będą używane symbole p, q, ..., zbiór wszystkich symboli predykatów będzie oznaczany $P = \bigcup_{n \in Nat} P_n$,
- symbole *spójników logicznych*:

implikacji	$\Rightarrow$
koniunkcji	$\wedge$
dysjunkcji (lub *alternatywy*)	$\vee$
negacji	$\neg$
równoważności	$\Leftrightarrow$

- symbole *kwantyfikatorów*:

kwantyfikatora ogólnego	$\forall$
kwantyfikatora szczegółowego	$\exists$

- symbole *pomocnicze*:

nawias otwierający	(
nawias zamykający	)

przecinek
kropka

Zakłada się, że zbiory symboli funkcyjnych F i symboli predykatów P są rozłączne. Para

$Sig =_{def} <F, P>$

będzie nazywana *sygnaturą* języka rachunku kwantyfikatorów. Rachunek kwantyfikatorów określa nie jeden konkretny język, ale rodzinę języków. Każdy język jest jednoznacznie wyznaczony przez sygnaturę. Składnia i semantyka rachunku kwantyfikatorów jest definiowana przy założeniu dowolnej, ale ustalonej sygnatury. Dobór odpowiedniej sygnatury wynika z zamierzonego zastosowania języka.

Reguły składni definiują dwa zbiory napisów – termy i formuły – nad alfabetem rachunku kwantyfikatorów.

Zbiór termów nad sygnaturą *Sig* i zbiorem zmiennych V, oznaczany $TERM(F, V)$, jest definiowany rekursywnie w sposób następujący:

- zmienne indywiduowe i stałe indywiduowe są termami, czyli

  $V \cup F_0 \subseteq TERM(F, V)$

- jeżeli $t_1, ..., t_k$ ($k = 1, 2, ...$) są termami, a f jest symbolem funkcyjnym k-argumentowym, to $f(t_1, ..., t_k)$ jest termem.

Term, który nie zawiera zmiennych indywiduowych nazywa się *termem stałym*.

Uwaga

Zbiór termów wyznacza pewne algebry omówione w rozdziale 8.

Zbiór formuł nad sygnaturą *Sig* i zbiorem zmiennych indywiduowych V, oznaczany $FORM(F, P, V)$, jest definiowany rekursywnie w sposób następujący:

1. symbole predykatów zeroargumentowych (stałe logiczne) są formułami;
2. jeżeli $t_1, ..., t_k$ ($k = 1, 2, ..., k$) są termami oraz p jest symbolem k-argumentowego predykatu, to formułą jest napis $p(t_1, ..., t_k)$;
3. jeżeli α, β są formułami, to formułami są także napisy:

   $\neg\alpha \quad (\alpha \Rightarrow \beta) \quad (\alpha \wedge \beta) \; (\alpha \vee \beta) \quad (\alpha \Leftrightarrow \beta)$

4. jeżeli α jest formułą oraz x jest zmienną indywiduową, to formułami są także:

   $(\exists x \bullet \alpha)$ oraz $(\forall x \bullet \alpha)$

Formuły spełniające podane wyżej warunki (1) i (2) nazywa się formułami *atomowymi*, formuły spełniające pozostałe warunki – formułami *złożonymi*.

Zbiór formuł $FORM(F, P, V)$ jest językiem formalnym rachunku kwantyfikatorów o sygnaturze $<F, P>$ nad zbiorem zmiennych V.

Uwaga

W celu zredukowania liczby nawiasów konwencję przyjętą dla rachunku zdań rozszerza się o ustalenie priorytetów dla kwantyfikatorów. Przyjmuje się, że kwantyfikatory mają priorytet niższy od spójników logicznych. Oznacza to, że formuła zapisana w postaci beznawiasowej

$$\exists x \bullet \alpha \wedge \beta \vee \gamma$$

gdzie: α, β, γ są dowolnymi jej podformułami, w postaci z nawiasami przedstawia się następująco:

$$\exists x \bullet ((\alpha \wedge \beta) \vee \gamma)$$

Kwantyfikatory występujące obok siebie łączą w prawo, to jest formuła

$$\exists x \bullet \exists y \bullet \alpha$$

oznacza

$$\exists x \bullet (\exists y \bullet \alpha)$$

Formuła α występująca po kwantyfikatorze w formule $\exists x.\alpha$ lub $\forall x.\alpha$ nazywa się *zasięgiem kwantyfikatora*. Symbol x występujący bezpośrednio za symbolem kwantyfikatorów nazywa się *wskaźnikiem związania*. Symbol wskaźnika określa rolę zmiennej x występującej w formule α, stanowiącej zasięg kwantyfikatora.

W zdefiniowanym języku kwantyfikatory wiążą jedynie zmienne indywiduowe, dlatego język ten nazywa się *językiem kwantyfikatorów pierwszego rzędu*. W logice rozpatruje się także inne języki, które dopuszczają wiązanie przez kwantyfikatory innych obiektów, na przykład *rachunek kwantyfikatorów drugiego rzędu* dodatkowo pozwala na wiązanie przez kwantyfikatory symboli predykatów. Dalsze rozważania ograniczają się wyłącznie do rachunku kwantyfikatorów pierwszego rzędu.

10.2. Indukcja i rekursja strukturalna

Indukcja strukturalna jest podstawową techniką dowodzenia własności termów i formuł. Przedstawiona dla rachunku zdań zasada indukcji strukturalnej rozszerza się na termy i formuły rachunku kwantyfikatorów.

Twierdzenie 10.1 (*Zasada indukcji strukturalnej dla termów*)

Niech $P(t)$ będzie pewną własnością zachodzącą dla termu $t \in TERM(F, V)$. Aby pokazać, że własność P zachodzi dla każdego termu rachunku kwantyfikatorów, wystarczy pokazać, że:

krok początkowy: własność P zachodzi dla każdej zmiennej $x \in V$, czyli $P(x)$,

krok indukcyjny:

jeżeli własność ta zachodzi dla termów $t_1, ..., t_n$, czyli $P(t_1), ..., P(t_n)$, oraz $f \in F_n$, to własność ta zachodzi dla termu $f(t_1, ..., t_n)$, czyli $P(f(t_1, ..., t_n))$.

Twierdzenie 10.2 (*Zasada indukcji strukturalnej dla formuł*)

Niech $P(\alpha)$ będzie pewną własnością zachodzącą dla formuły $\alpha \in FORM(F, P, V)$. Aby pokazać, że własność P zachodzi dla każdej formuły rachunku kwantyfikatorów, wystarczy pokazać, że:

krok początkowy: własność ta zachodzi dla każdej formuły atomowej,
krok indukcyjny:

- jeżeli własność P zachodzi dla formuły α, to zachodzi także dla formuły $\neg\alpha$,
- jeżeli własność P mają α oraz β, to ma ją także formuła $(\alpha \circ \beta)$, gdzie $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$ oznacza dowolny binarny spójnik logiczny,
- jeżeli własność P zachodzi dla formuły α, a x jest zmienną indywiduową, to P zachodzi także dla formuły $\forall x \bullet \alpha$ oraz dla $\exists x \bullet \alpha$.

Wykorzystując indukcję strukturalną, można pokazać, że termy i formuły dekomponują się jednoznacznie na komponenty składowe.

Lemat 10.1

Niech t oraz s będą termami. Jeżeli $t \equiv s\ w$ dla pewnego słowa nad alfabetem rachunku kwantyfikatorów, to w jest słowem pustym. Inaczej: żaden term nie jest właściwym prefiksem (niepustym początkowym fragmentem) innego termu.

Dowód

Zgodnie z zasadą indukcji strukturalnej rozpatruje się kolejno przypadki. Jeżeli t jest zmienną indywiduową, to t nie ma prefiksu właściwego. Zakłada się teraz, bez utraty ogólności, że t jest postaci $f(t_1, ..., t_n)$ oraz że $t \equiv s\ w$ dla pewnego słowa w, wówczas s musi być postaci $f(s_1, ..., s_n)\ w$. Dla każdego $i = 1, ..., n$ termy t_i oraz s_i są elementami składowymi termu t. Na mocy zatem założenia indukcyjnego – ani t_i, ani s_i nie są swoimi prefiksami, z czego wynika, że $t_i \equiv s_i$. To pociąga, że $w \equiv \varepsilon$, z czego ostatecznie wynika identyczność $t \equiv s$. ■

Lemat 10.2

Niech α oraz β będą dowolnymi formułami. Formuła α nie jest właściwym prefiksem formuły β.

Dowód

Dowód przebiega tak jak dla poprzedniego lematu i pozostawia się go jako ćwiczenie. ■

Na podstawie lematów można dowieść twierdzenia o jednoznaczności dekompozycji termów i formuł.

Twierdzenie 10.3 (*Twierdzenie o rozbiorze*)

1. Każdy term jest albo zmienną, albo stałą, albo termem złożonym postaci $f(t_1, ..., t_n)$, gdzie: f jest jednoznacznie określonym symbolem funkcyjnym, a $t_1, ..., t_n$ są jednoznacznie określonymi termami.
2. Każda formuła ma dokładnie jedną z postaci:
   a) $p(t_1, ..., t_n)$
   b) $\neg\alpha$
   c) $(\alpha \circ \beta)$ dla $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$
   d) $Q\,x \bullet \alpha$ dla $Q \in \{\forall, \exists\}$

Twierdzenie umożliwia jednoznaczne rekursywne definiowanie funkcji na zbiorach termów i formuł.

Przykład 10.1

Funkcja $|\,t\,|$ określająca długość termu t, rozumiana jako liczba jednostek leksykalnych wchodzących w skład termu, jest definiowana następująco:

a) $|\,x\,| =_{def} 1$
b) $|f(t_1, ..., t_n)| =_{def} |\,t_1\,| + ... + |\,t_n\,| + n + 2$

Podobnie, jak dla rachunku zdań, można również definiować rekursywnie funkcje określone na termach i formułach rachunku kwantyfikatorów.

Twierdzenie 10.4 (*Zasada rekursji strukturalnej dla termów*)

Następujący sposób postępowania definiuje jednoznacznie funkcję g określoną na zbiorze termów $TERM(F, V)$:

krok początkowy: na termach elementarnych (zmiennych i stałych indywiduowych) funkcja g jest określona bezpośrednio,

krok indukcyjny: wartość funkcji g dla termów złożonych jest określona pośrednio: wartość funkcji dla termu $f(t_1, ..., t_n)$ jest określona w terminach wartości funkcji na termach składowych $t_1, ..., t_n$.

Twierdzenie 10.5 (*Zasada rekursji strukturalnej dla formuł*)

Następujący sposób postępowania definiuje jednoznacznie funkcję g określoną na zbiorze formuł $FORM(F, P, V)$:

krok początkowy: na formułach elementarnych funkcja g jest określona bezpośrednio,

krok indukcyjny: wartość funkcji g dla formuł złożonych jest określona pośrednio:

- wartość funkcji g na formule $\neg\alpha$ jest określona w terminach wartości funkcji g na formule α,

- wartość funkcji g na formule $(\alpha \circ \beta)$, gdzie $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$ jest określona w terminach wartości na formułach α oraz β,
- wartość funkcji g na formułach $\forall x \bullet \alpha$ oraz dla $\exists x \bullet \alpha$ jest określona w terminach wartości na formule α.

Przykłady rekursywnego definiowania funkcji na termach i formułach rachunku kwantyfikatorów są przedstawione w kolejnym podrozdziale.

10.3. Zmienne wolne i związane

Zmienna indywiduowa x może występować tekstowo w wielu miejscach termu lub formuły. Każde takie pojawienie się zmiennej – poza miejscem bezpośrednio za kwantyfikatorem i przed kropką, gdzie określa się wskaźnik wiązania – nazywa się *wystąpieniem zmiennej*.

Wystąpienie zmiennej w danej formule może być wolne albo związane.

Wystąpienie zmiennej w danej formule nazywa się wystąpieniem *wolnym*, jeżeli wystąpienie to nie znajduje się w zasięgu żadnego kwantyfikatora, natomiast w przypadku przeciwnym – nazywa się wystąpieniem *związanym*. Ta sama zmienna może w danej formule mieć jednocześnie wystąpienia wolne i związane.

Przykład 10.2

Dana jest formuła

$$p(x, y) \Rightarrow \exists x \bullet q(x, y)$$

gdzie: p, q są pewnymi dwuargumentowymi predykatami. Zmienna x ma dwa wystąpienia. Pierwsze wystąpienie – jako argument predykatu p – jest wystąpieniem wolnym, drugie – jako argument predykatu q – jest wystąpieniem związanym. Zmienna y ma też dwa wystąpienia – oba wolne.

Niech V będzie zbiorem zmiennych indywiduowych. Definiuje się funkcje, które dla dowolnej formuły wyznaczają podzbiory zmiennych mające wystąpienia wolne i związane. Najpierw definiuje się pomocniczą funkcję

$$Var : TERM(F, V) \rightarrow 2^V$$

która dla dowolnego termu wyznacza zbiór zmiennych występujących w tym termie. Funkcja jest zdefiniowana rekursywnie:

1. $Var(c) =_{def} \varnothing$ dla $c \in F_0$
2. $Var(x) =_{def} \{x\}$ dla $x \in V$
3. $Var(f(t_1, ..., t_n)) =_{def} Var(t_1) \cup ... \cup Var(t_n)$ dla $f \in F_n$ $(n = 1, 2, ...)$

Term t, który nie zawiera zmiennych indywiduowych, czyli dla którego $Var(t) = \emptyset$, jest termem *stałym*.

Funkcja wyznaczająca zmienne mające wolne wystąpienia w formule jest funkcją typu

$$FV : FORM(F, P, V) \rightarrow 2^V$$

i jest zdefiniowana rekursywnie następująco:

1. $FV(p(t_1, ..., t_k)) =_{def} Var(t_1) \cup ... \cup Var(t_k)$
2. $FV(\neg\alpha) =_{def} FV(\alpha)$
3. $FV(\alpha \circ \beta) =_{def} FV(\alpha) \cup FV(\beta)$
   gdzie $\circ$ oznacza dowolny binarny spójnik logiczny, czyli $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$
4. $FV(Q\, x \bullet \alpha) =_{def} FV(\alpha) \setminus \{x\}$
   gdzie Q oznacza dowolny kwantyfikator, czyli $Q \in \{\forall, \exists\}$

Funkcja wyznaczająca zmienne mające związane wystąpienia w formule jest funkcją typu

$$BV : FORM(F, P, V) \rightarrow 2^V$$

i jest zdefiniowana rekursywnie następująco:

1. $BV(p(t_1, ..., t_k)) =_{def} \emptyset$
2. $BV(\neg\alpha) =_{def} BV(\alpha)$
3. $BV(\alpha \circ \beta) =_{def} BV(\alpha) \cup BV(\beta)$
   gdzie $\circ$ oznacza dowolny spójnik logiczny, czyli $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$
4. $BV(Q\, x \bullet \alpha) =_{def} BV(\alpha) \cup \{x\}$
   gdzie Q oznacza dowolny kwantyfikator, czyli $Q \in \{\forall, \exists\}$

Przykład 10.3

Dla formuły α postaci

$$p(x, y) \Rightarrow \forall x \bullet \forall z \bullet (q(x) \wedge p(x, z))$$

zbiór zmiennych mających wystąpienia wolne jest następujący:

$$\begin{aligned} FV(\alpha) &= FV(p(x, y) \cup FV(\forall x \bullet \forall z \bullet (q(x) \wedge p(x, z))) \\ &= \{x, y\} \cup FV(q(x) \wedge p(x, z)) \setminus \{x, z\} \\ &= \{x, y\} \cup \emptyset \\ &= \{x, y\} \end{aligned}$$

a zbiór zmiennych mających wystąpienia związane jest następujący:

$$\begin{aligned} BV(\alpha) &= BV(p(x, y) \cup BV(\forall x \bullet \forall z \bullet (q(x) \wedge p(x, z))) \\ &= \emptyset \cup \{x, z\} \cup BV(q(x) \wedge p(x, z)) \\ &= \{x, z\} \end{aligned}$$

Formuła zawierająca wolne wystąpienia zmiennych nazywa się formułą *otwartą*. *Zamknięciem* formuły otwartej nazywa się formułę otrzymaną przez poprzedzenie danej formuły otwartej kwantyfikatorami ogólnymi, wiążącymi wszystkie jej zmienne wolne. Formuła zamknięta jest *zdaniem*, czyli ma jednoznacznie określoną wartość logiczną: prawda albo fałsz.

Przykład 10.4

Formułami otwartymi są:

$\forall x \bullet p(x, y)$

$\forall x \bullet \forall y \bullet (q(x) \Rightarrow p(y, z))$

Formułami zamkniętymi (zdaniami) są:

$\forall x \bullet q(x)$

$\forall x \bullet \forall y \bullet (q(x) \wedge p(x, y))$

10.4. Podstawianie termów

Podstawieniem tekstowym termu za zmienne – albo krótko – *podstawieniem* nazywa się funkcję

$\sigma: V \rightarrow TERM(F, V)$

taką, że zbiór

$\{x \in V \mid \sigma(x) \neq x\}$

jest skończony. Zbiór ten będzie nazywany dziedziną podstawienia i oznaczany przez $dom(\sigma)$.

Funkcja σ jest *podstawieniem tożsamościowym* jeżeli $dom(\sigma) = \emptyset$. Podstawienie takie będzie oznaczane symbolem ε. Podstawienie nazywa się *podstawieniem podstawowym* albo *stałym*, jeżeli przeciwdziedzina $ran(\sigma)$ funkcji podstawienia zawiera tylko termy stałe.

Jeżeli $dom(\sigma) = \{x_1, ..., x_n\}$ oraz $\sigma(x_i) = t_i$, dla $i = 1, 2, ..., n$, to funkcję σ zapisuje się w postaci

$\sigma =_{def} [x_1 ::= t_1, ..., x_n ::= t_n]$

Zapis $x_i ::= t_i$ czyta się: x_i *jest zastępowane przez* t_i. Element $x_i ::= t_i$ nazywa się *przypisaniem* albo *wiązaniem*, dlatego podstawienie określa się też jako skończony zbiór przypisań albo wiązań.

Podstawienie σ można rozszerzyć na zbiór formuł. Najpierw rozszerza się je na zbiór termów, to znaczy do odwzorowania

$$\sigma' : TERM(F, V) \to TERM(F, V)$$

przyjmując

$$\sigma'(x) =_{def} \sigma(x) \text{ dla } x \in V$$
$$\sigma'(f(t_1, ..., t_n)) =_{def} f(\sigma'(t_1), ..., \sigma'(t_n))$$

W kolejnym kroku rekursywnie rozszerza się odwzorowanie σ' na odwzorowanie σ'': $FORM(F, P, V) \to FORM(F, P, V)$ w następujący sposób:

1. $\sigma''(t_1, ..., t_n)) =_{def} p(\sigma'(t_1), ..., \sigma'(t_k))$
2. $\sigma''(\neg\alpha) =_{def} \neg\sigma''(\alpha)$
3. $\sigma''((\alpha \circ \beta)) =_{def} (\sigma''(\alpha) \circ \sigma''(\beta))$
   gdzie $\circ$ oznacza dowolny spójnik logiczny, czyli $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$
4. $\sigma''(Q\,x \bullet \alpha) =_{def} Q\,x \bullet \sigma_1''(\alpha)$
   gdzie: Q oznacza dowolny kwantyfikator, czyli $Q \in \{\forall, \exists\}$, a σ_1'' jest obcięciem funkcji σ'' do zbioru zmiennych wolnych w formule $Qx \bullet \alpha$, czyli $\sigma_1'' = \sigma''|_{FV(Qx \bullet \alpha)}$. Oznacza to, że w formule $Q\,x \bullet \alpha$ mogą nastąpić przypisania tylko za wolne wystąpienia zmiennych.

Dalej, w celu uproszczenia oznaczeń, wszystkie symbole podstawienia będą pisane bez ′ oraz ″. Ponadto, ze względu na wygodę, zastosowanie podstawienia σ do termu t będzie zapisywane w postaci $t\sigma$ oraz – podobnie – zastosowanie do formuły α w postaci $\alpha\sigma$. Formułę $\alpha\sigma$ będzie się nazywać *ukonkretnieniem* formuły α przez podstawienie σ.

Przykład 10.5

Niech $\sigma =_{def} [x ::= t_1, y ::= t_2]$ oraz $\alpha \equiv \forall z \bullet (p(x, y, z) \Rightarrow q(x, z))$

wówczas

$$\alpha\sigma \equiv (\forall z \bullet (p(x, y, z) \Rightarrow q(x, z)))[x ::= t_1, y ::= t_2] \equiv \forall z \bullet (p(t_1, t_2, z) \Rightarrow q(t_1, z)).$$

Niech σ oraz τ będą dwoma podstawieniami. Złożenie podstawień σ oraz τ jest definiowane tak samo jak składanie funkcji. Jest zatem podstawieniem oznaczanym przez $\sigma \circ \tau$ – albo krótko $\sigma\tau$ – i zdefiniowanym następująco:

$$(\sigma\tau)(x) =_{def} \tau(\sigma(x)) \text{ dla } x \in V.$$

Jeżeli podstawienie σ jest takie, że istnieje dla niego podstawienie odwrotne σ^{-1} takie, że

$$\sigma\sigma^{-1} = \sigma^{-1}\sigma = \varepsilon$$

to σ jest nazywane *przemianowaniem* zmiennych. Term t_1 nazywa się *wariantem* termu t_2, jeżeli istnieje takie przemianowanie σ, że $t_1 \equiv \sigma(t_2)$.

Podstawienie σ nazywa się podstawieniem *idempotentnym*, jeśli $\sigma(\sigma(x))=\sigma(x)$ dla dowolnego $x \in V$.

Term t jest wolny w formule α ze względu na zmienną x, gdy zachodzi jeden z warunków:

1. α jest formułą atomową,
2. $\alpha \equiv \neg\beta$ oraz term t jest wolny w β ze względu na x,
3. $\alpha \equiv \beta_1 \circ \beta_2$, gdzie $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$ oraz term t jest wolny w β_1 oraz w β_2 ze względu na x,
4. $\alpha \equiv Q\,x \bullet \beta$,
5. $\alpha \equiv Q\,y \bullet \beta, x \neq y, Q \in \{\forall, \exists\}, y \notin Var(t)$ oraz term t jest wolny w β ze względu na x.

Inaczej: term t jest wolny w formule α ze względu na x wtedy, gdy podstawienie t za x w formule α nie powoduje, że któraś ze zmiennych występujących w termie t stanie się zmienną związaną.

Przykład 10.6

Dana jest formuła

$$p(x, y) \Rightarrow \forall x \bullet \forall z \bullet (q(x) \wedge p(y, z))$$

Term $f(x)$ jest wolny w tej formule ze względu na x, ale nie jest wolny ze względu na y, gdyż po zastąpieniu y przez term $f(x)$ otrzymano by formułę postaci

$$p(f(y), y) \Rightarrow \forall x \bullet \forall z \bullet (q(x) \wedge p(f(\underline{x}), z))$$

w której podkreślone wystąpienie zmiennej x stałoby się wystąpieniem związanym. Term $g(y, w)$ natomiast jest wolny w formule zarówno ze względu na x, jak i na y.

W dalszym ciągu, dokonując podstawienia σ w formule α, będzie zawsze wymagane, by dla dowolnej zmiennej $x \in dom(\sigma)$ odpowiadający jej term $\sigma(x)$ był wolny w α ze względu na x.

10.5. Semantyka

Rachunek kwantyfikatorów, stanowiąc uogólnienie rachunku zdań, przejmuje znaczenie przypisywane spójnikom logicznym zgodne ze standardową interpretacją.

Sposób opisu semantyki rachunku kwantyfikatorów jest podobny do opisu semantyki rachunku zdań. Opis rozpoczyna się ustaleniem dziedzin semantycznych, w których

będzie wyrażane znaczenie elementów języka – termów i formuł, a następnie określa się interpretację symboli funkcyjnych i predykatywnych. Oba te elementy – dziedziny semantyczne i interpretacja symboli funkcyjnych i predykatywnych – stanowią model interpretacji. Po ustaleniu modelu interpretacji definiuje się znaczenie najpierw termów, a następnie formuł.

Dziedziną interpretacji dla formuł – tak jak w rachunku zdań – jest zbiór wartości logicznych *Logiczne*. Dziedzina interpretacji termów może się natomiast składać z wielu różnych zbiorów wartości $D_1, ..., D_m$ ($m > 0$) – mówi się, że dziedzina jest *wielorodzajowa*. W dalszych rozważaniach, oprócz niektórych przykładów, dziedziny szczegółowe nie będą rozróżniane. W celu uproszczenia prezentacji zakłada się, że istnieje jedna *wspólna dziedzina* $\boldsymbol{D}$ stanowiącą mnogościową sumę dziedzin szczegółowych. Dziedzina ta jest rozłączna ze zbiorem wartości logicznych.

Uwaga

Rozróżnienia dziedzin interpretacji dokonuje się w *rachunku kwantyfikatorów z typami*.

Nowymi elementami, które wymagają dodatkowej interpretacji, są symbole funkcyjne oraz predykatywne. Symbolom funkcyjnym będą odpowiadały pewne funkcje, symbolom predykatywnym – pewne relacje w ustalonej dziedzinie interpretacji. Ponieważ dowolną relację można w równoważny sposób przedstawić za pomocą jej funkcji charakterystycznej, dlatego symbolom predykatywnym będą również przyporządkowywane funkcje, ale o wartościach w zbiorze *Logiczne*.

Uwaga

Przypomnijmy, że dla relacji $R \subseteq D_1 \times ... \times D_n$ ($n > 1$) jej *funkcja charakterystyczna*

$f_R : D_1 \times ... \times D_n \rightarrow Logiczne$

jest zdefiniowana następująco:

$f_R(d_1, ..., d_n) = \mathbf{P}$ wtedy i tylko wtedy, gdy $<d_1, ..., d_n> \in R$.

Wprowadza się funkcję *interpretacji symboli funkcyjnych i predykatywnych* $\boldsymbol{I}$, określoną na zbiorze symboli $F \cup P$ taką, że:

- jeżeli $f \in F_n$, dla $n \in Nat$, to $\boldsymbol{I}(f) : \boldsymbol{D}^n \rightarrow \boldsymbol{D}$ jest n-argumentową funkcją, w szczególnym przypadku, gdy f jest stałą, czyli $f \in F_0$, $\boldsymbol{I}(f) \in \boldsymbol{D}$,
- jeżeli $p \in P_n$, dla $n \in Nat$, to $\boldsymbol{I}(p) : \boldsymbol{D}^n \rightarrow Logiczne$ jest n-argumentową funkcją o wartościach logicznych.

Ponieważ została wprowadzona jedna wspólna dziedzina interpretacji $\boldsymbol{D}$, funkcje definiowane przez $\boldsymbol{I}$ są więc najczęściej funkcjami częściowymi.

Para

$\boldsymbol{M} = <\boldsymbol{D}, \boldsymbol{I}>$

gdzie: $\boldsymbol{D}$ jest niepustą dziedziną interpretacji, a $\boldsymbol{I}$ jest interpretacją symboli funkcyjnych i predykatywnych, będzie nazywana *modelem* dla formalnego języka rachunku kwantyfikatorów o sygnaturze $Sig = <F, P>$.

Następny etap definiowania semantyki rachunku kwantyfikatorów polega na nadaniu interpretacji dowolnym termom. Pomocniczym pojęciem – podobnie jak w rachunku zdań – jest funkcja wartościowania zmiennych indywiduowych.

Wartościowaniem zmiennych jest funkcja v o następującej sygnaturze:

$$v : V \rightarrow \boldsymbol{D}$$

Niech $\boldsymbol{M} = <\boldsymbol{D}, \boldsymbol{I}>$ będzie modelem języka o sygnaturze $Sig = (F, P)$. Każdemu termowi t, przy ustalonym wartościowaniu v, przyporządkuje się pewną wartość z dziedziny interpretacji $\boldsymbol{D}$. Wartość tę wyznacza funkcja *interpretacji termów przy wartościowaniu* v

$$INT_v : TERM(F, V) \rightarrow \boldsymbol{D}$$

zdefiniowana rekursywnie względem struktury składniowej zbioru termów w sposób następujący:

- jeżeli term jest zmienną indywiduową $x \in V$, to
  $INT_v(x) =_{def} v(x)$
- jeżeli term jest postaci $f(t_1, ..., t_n)$, gdzie $f \in F_n$, oraz $t_1, ..., t_n \in TERM(F, V)$, to
  $INT_v(f(t_1, ..., t_n)) =_{def} I(f)(INT_v(t_1), ..., INT_v(t_n))$

Należy zauważyć, że interpretacja zmiennych indywiduowych nie zależy od interpretacji $\boldsymbol{I}$. Wartość termów stałych, przy ustalonej interpretacji $\boldsymbol{I}$, nie zależy od wartościowania v.

Niech v będzie wartościowaniem, x – zmienną indywiduową oraz niech $a \in \boldsymbol{D}$. Wartościowanie $v[x := a]$ definiuje się jako

$$v[x := a](y) = \begin{cases} a & \text{gdy } y = x \\ v(y) & \text{w przypadku przeciwnym} \end{cases}$$

Wartościowanie $v\ [x := a]$ jest więc modyfikacją wartościowania v, polegającą na przypisaniu wartości a ustalonej zmiennej x i pozostawieniu niezmienionych wartości przypisanych do pozostałych zmiennym.

Uwaga

Należy odróżniać dwa podobne oznaczenia: $[x := a]$ oraz $[x ::= t]$. Pierwsze określa modyfikację definicji pewnej funkcji; powyżej odnosi się do modyfikacji funkcji wartościowania v. Drugie określa tekstową modyfikacją pewnego napisu, na przykład termu lub formuły.

Jeżeli INT_ν jest interpretacją przy pewnym wartościowaniu ν, to niech $INT_{\nu[x := a]}$ będzie interpretacją przy wartościowaniu $\nu[x := a]$.

Rozszerza się teraz funkcję INT_ν na zbiór formuł jako funkcję

$$INT_\nu : FORM(F, P, V) \to \boldsymbol{D}$$

W celu uniknięcia zbyt wielu oznaczeń, funkcja INT_ν będzie oznaczać interpretację zarówno termów, jak i formuł. *Interpretację formuł w modelu M przy wartościowaniu* ν, definiuje się rekursywnie względem struktury składniowej zbioru formuł:

a) $INT_\nu(p(t_1, ..., t_n)) =_{def} I(p)(INT_\nu(t_1), ..., INT_\nu(t_n))$

b) $INT_\nu(\neg\alpha) =_{def} \neg INT_\nu(\alpha)$

c) $INT_\nu(\alpha \wedge \beta) =_{def} INT_\nu(\alpha) \wedge INT_\nu(\beta)$

d) $INT_\nu(\alpha \vee \beta) =_{def} INT_\nu(\alpha) \vee INT_\nu(\beta)$

e) $INT_\nu(\alpha \Rightarrow \beta) =_{def} INT_\nu(\alpha) \Rightarrow INT_\nu(\beta)$

f) $INT_\nu(\alpha \Leftrightarrow \beta) =_{def} INT_\nu(\alpha) \Leftrightarrow INT_\nu(\beta)$

g) $INT_\nu(\forall x \bullet \alpha) =_{def}$ $\begin{cases} \mathbf{P} & \text{gdy dla dowolnego } d \in D \text{ zachodzi } INT_{\nu[x:=d]}(\alpha) = \mathbf{P} \\ \mathbf{F} & \text{w przypadku przeciwnym} \end{cases}$

h) $INT_\nu(\exists x \bullet \alpha) =_{def}$ $\begin{cases} \mathbf{P} & \text{gdy dla pewnego } d \in D \text{ zachodzi } INT_{\nu[x:=d]}(\alpha) = \mathbf{P} \\ \mathbf{F} & \text{w przypadku przeciwnym} \end{cases}$

Uwaga

Symbole **P** oraz **F** są skrótami wartości logicznych *prawda*, *fałsz*. Spójniki logiczne, występujące po prawej stronie w powyższej definicji, są rozumiane zgodnie z ich standardową interpretacją przyjętą dla rachunku zdań. Język rachunku zdań jest tutaj fragmentem metajęzyka służącego do definiowania języka rachunku kwantyfikatorów. Elementami metajęzyka są również pojęcia *dla dowolnego* i *dla pewnego*, użyte w punktach g) oraz h). Należą one do języka teorii mnogości.

10.6. Spełnialność formuł

Formuła α jest *spełniona w modelu M dla wartościowania* ν, gdy $INT_\nu(\alpha) = \mathbf{P}$. Fakt ten będzie zapisywany również w postaci

$$INT_\nu \models \alpha$$

Formuła α *jest spełniona w modelu M*, co oznacza się

$$M \models \alpha$$

gdy jest spełniona w tym modelu dla dowolnego wartościowania.

Jeżeli formuła α nie jest spełnialna w modelu M, będzie to zapisywane w postaci

$$M \not\models \alpha$$

Formuła α *jest spełnialna*, gdy istnieje model, w którym jest spełniona.

Formuła jest *tautologią*, co oznacza się

$$\models \alpha$$

gdy jest spełnialna w dowolnym modelu. Tautologia jest zatem schematem wypowiedzi zawsze prawdziwej, niezależnie od interpretacji przyjętej dla symboli funkcyjnych i predykatywnych. Tautologia zakłada, oczywiście, standardową interpretację spójników logicznych.

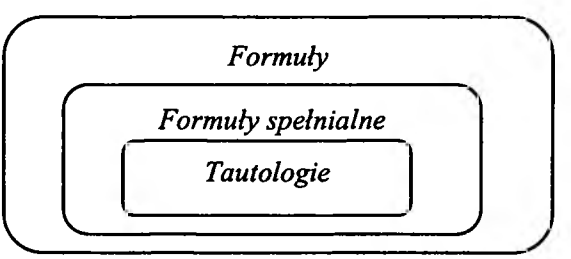
Wprowadzone pojęcia spełnialności uogólnia się na zbiory formuł.

Zbiór formuł Φ jest *spełniony w modelu M dla wartościowania v*, gdy wszystkie formuły zbioru Φ są spełnione w tym modelu M przy wartościowaniu v.

Zbiór formuł Φ jest *spełniony w modelu M*, gdy każda formuła tego zbioru jest spełniona w tym modelu.

Zbiór formuł Φ jest *spełnialny*, gdy istnieje model, w którym zbiór ten jest spełniony.

Ilustracją związków pomiędzy różnymi rodzajami formuł przedstawiono na rysunku 10.1: zbiór formuł spełnialnych jest oczywiście podzbiorem formuł, a zbiór tautologii jest podzbiorem zbioru formuł spełnialnych.



Rys. 10.1. Związek pomiędzy formułami spełnianymi i tautologiami

Niech Φ będzie zbiorem formuł oraz α – pojedynczą formułą. Pisze się

$$\Phi \models \alpha$$

co czyta się: *formuła* α *wynika semantycznie ze zbioru formuł* Φ, albo inaczej: *formuła* α *jest semantyczną konsekwencją zbioru formuł* Φ, co oznacza, że każdy model, w którym są spełnione formuły zbioru Φ jest również modelem, w którym spełniona jest formuła α.

Pisze się

$\beta \models \alpha$ zamiast $\{\beta\} \models \alpha$

oraz

$\models \alpha$ zamiast $\varnothing \models \alpha$

Pusty zbiór formuł po lewej stronie symbolu $\models$ jest oczywiście prawdziwy w każdym modelu. Zapis $\models \alpha$ oznacza zatem, że α jest tautologią.

Uwaga

Należy zwrócić uwagę na dwie role symbolu $\models$. Po jego prawej stronie występuje zawsze formuła, na przykład α, natomiast po lewej – może wystąpić model M lub interpretacja INT_v albo zbiór formuł Φ. W pierwszym przypadku symbol $\models$ oznacza, że formuła α jest spełnialna w modelu M lub w interpretacji INT_v, w drugim – że jest semantyczną konsekwencją zbioru formuł Φ.

Mówi się, że dwie formuły α, β są *semantycznie równoważne*, co pisze się

$\alpha = \beta$

wtedy i tylko wtedy, gdy

$\alpha \models \beta$ oraz $\beta \models \alpha$

Lemat 10.3

Niech Φ będzie zbiorem formuł oraz α – pojedynczą formułą, wówczas

$\Phi \models \alpha$

wtedy i tylko wtedy, gdy zbiór $\Phi \cup \{\neg\alpha\}$ jest niespełnialny.

Dowód

Jeżeli zachodzi $\Phi \models \alpha$, to oznacza, że każdy model, w którym jest spełniony zbiór formuł Φ, jest również modelem, w którym jest spełniona formuła α. Zachodzi to wtedy i tylko wtedy, gdy nie istnieje model, w którym są spełnione formuły Φ, a w którym nie jest spełniona formuła α. Ale to z kolei oznacza, że nie istnieje model, w którym są spełnione formuły zbioru $\Phi \cup \{\neg\alpha\}$, czyli gdy zbiór ten nie jest spełnialny. $\Phi \models \alpha$ zachodzi zatem wtedy i tylko wtedy, gdy zbiór formuł $\Phi \cup \{\neg\alpha\}$ jest niespełnialny. ■

Twierdzenie 10.6 (*Twierdzenie o dedukcji*)

Niech $\Phi =_{def} \{\alpha_1, ..., \alpha_n\}$ będzie niepustym zbiorem formuł oraz β – pojedynczą formułą, wówczas

$\Phi \models \beta$

wtedy i tylko wtedy, gdy

$\models \alpha_1 \wedge ... \wedge \alpha_n \Rightarrow \beta$

Dowód

$\Phi \models \beta$ zachodzi wtedy i tylko wtedy, gdy w każdym modelu, w którym spełnione są formuły $\alpha_1, ..., \alpha_n$, spełniona jest również formuła β. To dzieje się dokładnie wtedy, gdy w każdym modelu, w którym jest spełniona formuła $\alpha_1 \wedge ... \wedge \alpha_n$, spełniona jest również formuła β, co z kolei zachodzi wtedy i tylko wtedy, gdy

$$\models \alpha_1 \wedge ... \wedge \alpha_n \Rightarrow \beta$$ ■

Twierdzenie o dedukcji ma ważne znaczenie w sytuacjach, gdy tezy badanych twierdzeń mają schemat postaci $\Phi \models \beta$. Bezpośrednie sprawdzenie czy formuła β jest semantyczną konsekwencją zbioru formuł Φ nie jest możliwe, gdyż ogólnie oznacza to przebadanie nieskończonej liczby modeli. Twierdzenie o dedukcji umożliwia zastąpienie takiego sprawdzania zbadaniem, czy formuła postaci $\alpha_1 \wedge ... \wedge \alpha_n \Rightarrow \beta$ jest tautologią. Badanie, czy formuła jest tautologią, można przeprowadzić, konstruując odpowiedni dowód według pewnego systemu dowodowego. W konstrukcji dowodu – co będzie pokazane w następnych rozdziałach – wykorzystuje się wyłącznie przekształcenia tekstowe badanej formuły.

Użyteczną konsekwencją twierdzenia o dedukcji jest równoważność stwierdzeń:

$\Phi \cup \{\alpha\} \models \beta$ wtedy i tylko wtedy, gdy $\Phi \models \alpha \Rightarrow \beta$.

Fakt ten wskazuje na pewne podobieństwo symboli $\models$ oraz $\Rightarrow$, nie oznacza jednak, że symbole te mają takie samo znaczenie. Należy zwrócić uwagę na to, że symbol konsekwencji $\models$ należy do metajęzyka, a symbol implikacji $\Rightarrow$ do formalnego języka rachunku kwantyfikatorów. Podobna uwaga odnosi się do symboli spójnika równoważności $\Leftrightarrow$ oraz symbolu równoważności semantycznej $=$.

10.7. Wybrane prawa rachunku kwantyfikatorów

Korzystając bezpośrednio z definicji interpretacji formuł rachunku kwantyfikatorów, można sprawdzić, że zachodzą podane poniżej równości semantyczne, nazywane też *prawami rachunku kwantyfikatorów*. Niektóre z nich mają też tradycyjne nazwy.

Jeżeli α oraz β są dowolnymi formułami, to tautologiami są formuły:

$$\models \quad (\forall x \bullet \alpha) \Rightarrow \alpha$$

$$\models \quad \alpha \Rightarrow (\exists x \bullet \alpha)$$

Pierwsze prawo wyraża to, że jeżeli dowolna formuła jest spełniona dla wszystkich wartościowań, to jest również spełniona dla dowolnie wybranego wartościowania. Drugi natomiast wyraża to, że jeżeli dowolna formuła jest spełniona dla pewnego wartościowania, to znaczy, że istnieje wartościowanie, przy którym formuła ta jest spełniona.

Prawa de Morgana

$\models \quad (\neg\forall x \bullet \alpha) \Leftrightarrow (\exists x \bullet \neg\alpha)$

$\models \quad (\neg\exists x \bullet \alpha) \Leftrightarrow (\forall x \bullet \neg\alpha)$

Prawa de Morgana wskazują na związki semantyczne pomiędzy kwantyfikatorem ogólnym i szczegółowym. Oznaczają one, że w rachunku kwantyfikatorów, bez utraty siły ekspresji języka, można się ograniczyć do posługiwania się tylko jednym kwantyfikatorem. Jest to analogia do rachunku zdań, w którym – bez utraty ogólności – zbiór wykorzystywanych spójników logicznych można ograniczyć do funkcjonalnie pełnego zbioru spójników logicznych.

Prawa rozdzielności kwantyfikatorów

$\models \quad (\forall x \bullet \alpha) \wedge (\forall x \bullet \beta) \Leftrightarrow (\forall x \bullet \alpha \wedge \beta)$

$\models \quad (\forall x \bullet \alpha) \vee (\forall x \bullet \beta) \Rightarrow (\forall x \bullet \alpha \vee \beta)$

$\models \quad (\exists x \bullet \alpha) \vee (\exists x \bullet \beta) \Leftrightarrow (\exists x \bullet \alpha \vee \beta)$

$\models \quad (\exists x \bullet \alpha \wedge \beta) \Rightarrow (\exists x \bullet \alpha) \wedge (\exists x \bullet \beta)$

Prawa przemianowania kwantyfikatorów

Jeżeli $y \notin FV(\alpha)\backslash\{x\}$ oraz y jest zmienną wolną w α ze względu na x, to

$\models \quad (\forall x \bullet \alpha) \Leftrightarrow (\forall y \bullet \alpha[x ::= y])$

$\models \quad (\exists x \bullet \alpha) \Leftrightarrow (\exists y \bullet \alpha[x ::= y])$

Prawo to pozwala na przemianowanie nazwy zmiennej wiązanej przez kwantyfikator. Zmienną taką można zastąpić dowolną inną zmienną, która nie jest wolna w formule będącej w zasięgu kwantyfikatora. Na przykład formuła

$$(\forall x \bullet p(x, y) \Rightarrow q(x)) \vee (\forall y \bullet r(x, y) \wedge \neg p(x, y))$$

jest równoważna formule

$$(\forall z \bullet p(z, y) \Rightarrow q(z)) \vee (\forall w \bullet r(x, w) \wedge \neg p(x, w))$$

Prawa przestawiania kwantyfikatorów

$\models \quad (\forall x \bullet \forall y \bullet \alpha) \Leftrightarrow (\forall y \bullet \forall x \bullet \alpha)$

$\models \quad (\exists x \bullet \exists y \bullet \alpha) \Leftrightarrow (\exists y \bullet \exists x \bullet \alpha)$

$\models \quad (\exists x \bullet \forall y \bullet \alpha) \Rightarrow (\forall y \bullet \exists x \bullet \alpha)$

Prawa włączania i wyłączania dla kwantyfikatorów

Jeżeli $x \notin FV(\beta)$, to

$\models \quad ((Q\,x \bullet \alpha) \circ \beta) \Leftrightarrow (Q\,x \bullet \alpha \circ \beta)$

$\models \quad$ dla $\circ \in \{\wedge, \vee, \Rightarrow\}$, oraz $Q \in \{\forall, \exists\}$

Prawa rozkładu kwantyfikatorów

$\models (\forall x \bullet \alpha \Rightarrow \beta) \Rightarrow ((\forall x \bullet \alpha) \Rightarrow (\forall x \bullet \beta))$

$\models (\forall x \bullet \alpha \Rightarrow \beta) \Rightarrow ((\exists x \bullet \alpha) \Rightarrow (\exists x \bullet \beta))$

$\models (\forall x \bullet \alpha \wedge \beta) \Leftrightarrow ((\forall x \bullet \alpha) \wedge (\forall x \bullet \beta))$

$\models (\exists x \bullet \alpha \wedge \beta) \Rightarrow ((\exists x \bullet \alpha) \wedge (\exists x \bullet \beta))$

$\models (\forall x \bullet \alpha \vee \forall x \bullet \beta) \Rightarrow (\forall x \bullet \alpha \vee \beta)$

$\models (\exists x \bullet \alpha \vee \beta) \Leftrightarrow ((\exists x \bullet \alpha) \vee (\exists x \bullet \beta))$

$\models (\forall x \bullet \alpha \Leftrightarrow \beta) \Rightarrow ((\forall x \bullet \alpha) \Leftrightarrow (\forall x \bullet \beta))$

$\models (\forall x \bullet \alpha \Leftrightarrow \beta) \Rightarrow ((\exists x \bullet \alpha) \Leftrightarrow (\exists x \bullet \beta))$

10.8. Przedrostkowa postać normalna

Niech α będzie formułą rachunku kwantyfikatorów.

Definicja 10.1

Formuła α znajduje się w postaci *przedrostkowej normalnej* lub w postaci *PNF* (*Prenex Normal Form*) wtedy i tylko wtedy, gdy jest ona w postaci

$$Q_1 x_1 \bullet Q_2 x_2 \bullet \ldots Q_n x_n \bullet \beta$$

gdzie: $Q_1, Q_2, \ldots, Q_n \in \{\forall, \exists\}$, a formuła β nie zawiera kwantyfikatorów.

Część $Q_1 x_1 \bullet Q_2 x_2 \bullet \ldots Q_n x_n \bullet$ nazywa się *przedrostkiem* formuły α, a β nazywa się *matrycą* formuły α.

Dla dowolnej formuły rachunku kwantyfikatorów istnieje równoważnie semantyczna formuła, która jest w przedrostkowej postaci normalnej. Jeżeli α będzie pewną formułą, to przez $PNF(\alpha)$ będzie się oznaczać formułę, która jest w przedrostkowej postaci normalnej, i która jest semantycznie równoważna α.

Poniżej jest przedstawiany algorytm sprowadzania dowolnej formuły do przedrostkowej postaci normalnej. Algorytm ten dokonuje jeszcze dodatkowego przekształcenia, polegającego na eliminacji z matrycy formuły spójników równoważności i implikacji, a pozostawieniu tylko negacji, dysjunkcji i koniunkcji.

Algorytm sprowadzania do przedrostkowej postaci normalnej

Dane: formuła α.
Wynik: formuła $PNF(\alpha)$.
Procedura: procedura postępowania polega na etapowym, tekstowym przekształcaniu formuły α. Formuła pośrednia jest oznaczana przez β.

1. Początkowo przyjmuje się, że formuła β jest tekstowo identyczna z α.
2. Eliminuje się z formuły β spójnik równoważności, zastępując tekstowo podformuły postaci $\gamma \Leftrightarrow \delta$ formułami postaci $(\gamma \Rightarrow \delta) \wedge (\delta \Rightarrow \gamma)$.
3. Eliminuje się z formuły β spójnik implikacji, zastępując tekstowo podformuły postaci $\gamma \Rightarrow \delta$ formułami postaci $\neg\gamma \vee \delta$.
4. Wprowadza się znak negacji bezpośrednio przed symbole predykatów, zastępując (dopóki można) podformuły zgodnie z poniższą tablicą:

Lp.	Podformuła zastępowana	Formuła zastępująca
1	$\neg(\neg\delta)$	δ
2	$\neg(\delta \vee \gamma)$	$\neg\delta \wedge \neg\gamma$
3	$\neg(\delta \wedge \gamma)$	$\neg\delta \vee \neg\gamma$
4	$\neg\forall x \bullet \delta$	$\exists x \bullet \neg\delta$
5	$\neg\exists x \bullet \delta$	$\forall x \bullet \neg\delta$

5. Dopóki formuła β nie jest w przedrostkowej postaci normalnej, przekształca się ją zgodnie z poniższą tablicą:

Lp.	Podformuła zastępowana	Formuła zastępująca	Warunek zastąpienia
1	$(Q\,x \bullet \delta) \vee \gamma$	$Q\,x \bullet (\delta \vee \gamma)$	$x \notin FV(\gamma)$
2	$(Q\,x \bullet \delta) \wedge \gamma$	$Q\,x \bullet (\delta \wedge \gamma)$	$x \notin FV(\gamma)$
3	$(\forall x \bullet \delta) \wedge (\forall x \bullet \gamma)$	$\forall x \bullet (\delta \wedge \gamma)$	
4	$(\exists x \bullet \delta) \vee (\exists x \bullet \gamma)$	$\exists x \bullet (\delta \vee \gamma)$	
5	$(Q_1\,x \bullet \delta) \vee (Q_2\,y \bullet \gamma)$	$Q_1\,x \bullet Q_2\,y \bullet (\delta \vee \gamma)$	$x \notin FV(\gamma), y \notin FV(\delta)$
6	$(Q_1\,x \bullet \delta) \wedge (Q_2\,y \bullet \gamma)$	$Q_1\,x \bullet Q_2\,y \bullet (\delta \wedge \gamma)$	$x \notin FV(\gamma), y \notin FV(\delta)$

gdzie $Q, Q_1, Q_2 \in \{\forall, \exists\}$.

6. Formułę $PNF(\alpha)$ definiuje się jako formułę β.

Uwaga

Jeżeli warunki zastąpienia bezpośrednio nie są spełnione, to można je spełnić przez przemianowanie zmiennych wiązanych przez kwantyfikator. Podstawą przemianowania jest równoważność semantyczna: jeżeli $x \notin FV(\alpha) \setminus \{x\}$ oraz y jest zmienną wolną w α ze względu na x, to

$$Qx \bullet \alpha = Qy \bullet \alpha[x ::= y] \text{ dla } Q \in \{\forall, \exists\}$$

Przykład 10.7

Rozpatruje się sprowadzenie do przedrostkowej postaci normalnej formuły

$$(\exists x \bullet \alpha) \Rightarrow (\exists y \bullet \beta)$$

Na podstawie kroku 3. algorytmu, eliminując implikację, otrzymuje się

$$\neg(\exists x \bullet \alpha) \vee (\exists y \bullet \beta)$$

Na podstawie kroku 4. algorytmu, przypadek 4., otrzymuje się

$(\forall x \bullet \neg\alpha) \vee (\exists y \bullet \beta)$

Na podstawie kroku 5. algorytmu, przypadek 5., otrzymuje się

$\forall x \bullet \exists y \bullet (\neg \alpha \vee \beta)$

przy czym zakłada się, że $x \notin FV(\beta)$ oraz $y \notin FV(\alpha)$.

10.9. Przykład języka rachunku kwantyfikatorów

Omawia się przykład prostego języka kwantyfikatorów, który występuje w wielu językach programowania. Język ma służyć do przedstawiania prostych wyrażeń arytmetycznych, o wartościach ze skończonego podzbioru liczb całkowitych, i predykatów określonych na tych wyrażeniach. Jego interpretacja jest również zgodna z interpretacją przyjmowaną w językach programowania.

Alfabet języka składa się z:

1. zbioru zmiennych indywiduowych, reprezentowanych przez identyfikatory,
2. zbioru symboli funkcyjnych zawierającego:
   - dwie stałe: *ZERO*, *ONE*
   - jedną operację jednoargumentową: *alt*
   - cztery operacje dwuargumentowe: *_add_*, *_sub_*, *_mult_*, *_div_*
3. dwóch dwuargumentowych predykatów: *_eq_*, *_less_*
4. symboli spójników logicznych: **_and_**, **_or_**, **not_**
5. trzech symboli pomocniczych: ( ),

Sygnaturą języka jest więc

$Sig = <\{ ZERO, ONE, \_add\_, \_sub\_, \_mult\_, \_div\_\}, \{\_eq\_, less\_\}>$

Wyznaczony zgodnie z sygnaturą *Sig* zbiór termów zawiera:

1. zmienne indywiduowe oraz stałe,
2. napisy postaci:

   $alt\ t_1 \quad (t_1\ add\ t_2) \quad (t_1\ sub\ t_2) \quad (t_1\ mult\ t_2) \quad (t_1\ div\ t_2)$

   gdzie: t_1, t_2 są termami.

Zbiór formuł jest określony następująco:

1. jeżeli t_1, t_2 są termami, to formułami są:

   $(t_1\ eq\ t_2) \quad (t_1\ less\ t_2)$

2. jeżeli α, β są formułami, to formułami są także:

   $(\alpha\ \mathbf{and}\ \beta) \quad (\alpha\ \mathbf{or}\ \beta) \quad \mathbf{not}\ \alpha$

Zbiór formuł jest uboższy od pełnego języka kwantyfikatorów, gdyż nie zawiera kwantyfikatorów.

Dziedziną interpretacji termów niech będzie zbiór:

$$\boldsymbol{D} =_{\text{def}} Integer \cup \{error\}$$

gdzie

$$Integer =_{\text{def}} \{-N, \ldots, 0, \ldots, N\} \qquad N \in Nat \setminus \{0\}$$

Zbiór *Integer* reprezentuje typowy skończony zbiór wartości reprezentowany przez odpowiednik typu całkowitego w językach programowania. Element *error* ma natomiast reprezentować tak zwany *błąd abstrakcyjny*, powstający podczas obliczania wartości termów. Odzwierciedla to typową sytuację, która powstaje na przykład podczas obliczeń arytmetycznych w programie, gdy obliczona wartość wykracza poza zakres wartości dopuszczalnych. Będzie używane też oznaczenie na zbiór

$$Integer_{error} =_{\text{def}} Integer \cup \{error\}$$

Interpretacja $\boldsymbol{I}$ ustala przyporządkowania:

1. symbolom funkcyjnym funkcje typu:

$$\boldsymbol{I}(ZERO) : \rightarrow Integer$$
$$\boldsymbol{I}(ONE) : \rightarrow Integer$$
$$\boldsymbol{I}(alt) : Integer_{error} \rightarrow Integer_{error}$$
$$\boldsymbol{I}(add), \boldsymbol{I}(sub), \boldsymbol{I}(mult), \boldsymbol{I}(div) : Integer_{error}^{2} \rightarrow Integer_{error}$$

2. symbolom predykatów funkcje typu:

$$\boldsymbol{I}(eq) : Integer^2 \rightarrow Logiczne$$
$$\boldsymbol{I}(less) : Integer^2 \rightarrow Logiczne$$

Stałe są zdefiniowane jako:

$$\boldsymbol{I}(ZERO) =_{\text{def}} 0$$
$$\boldsymbol{I}(ONE) =_{\text{def}} 1$$

Jeżeli wartością któregokolwiek argumentu pozostałych operacji jest *error*, to wynikiem operacji jest również *error*. W podawanych niżej definicjach zakłada się, że argumenty a oraz b są elementami zbioru *Integer*:

$$\boldsymbol{I}(alt)\, a =_{\text{def}} \begin{cases} -a & a \in Integer \\ error & \text{w przeciwnym przypadku} \end{cases}$$

$$(a\ \boldsymbol{I}(add)\ b) =_{\text{def}} \begin{cases} a+b & a+b \in Integer \\ error & \text{w przeciwnym przypadku} \end{cases}$$

$$(a\ \boldsymbol{I}(sub)\ b) =_{\text{def}} \begin{cases} a-b & a-b \in Integer \\ error & \text{w przeciwnym przypadku} \end{cases}$$

$$(a\ I(mult)\ b) =_{def} \begin{cases} a*b & a*b \in Integer \\ error & \text{w przeciwnym przypadku} \end{cases}$$

$$(a\ I(div)\ b) =_{def} \begin{cases} a/b & a/b \in Integer \\ error & \text{w przeciwnym przypadku} \end{cases}$$

$(a\ I(eq)\ b) =_{def} a = b$

$(a\ I(less)\ b) =_{def} a < b$

Po prawej stronie powyższych definicji występują symbole znanych operacji arytmetycznych i operacji porównań w dziedzinie liczb całkowitych. Symbole te, podobnie jak symbole *a*, *b*, należą do metajęzyka opisującego semantykę wprowadzonego języka formalnego.

Zdefiniowany język rachunku kwantyfikatorów nie wprowadza symboli kwantyfikatorów. Formuły tego języka to odpowiednik napisów, które w językach programowania określa się jako wyrażenia logiczne albo wyrażenia boolowskie. Oczywiście, wyrażenia logiczne w praktycznie stosowanych językach programowania są na ogół bogatsze, gdyż operują szerszym zbiorem termów oraz symboli predykatów.

Obliczenie wartości termu

$(x\ mult\ (ONE\ add\ y))$

przy wartościowaniu

$v = \{<x, 4>, <y, 5>\}$

przebiega następująco: Zgodnie z definicją funkcji interpretacji termów $INT_v(t)$, przy założeniu, że wartość *N* w zbiorze *Integer* wynosi 10, otrzymuje się:

$INT_v(\ (x\ mult\ (ONE\ add\ y))\ ) =$

$INT_v(x)\ I(mult)\ INT_v(\ (ONE\ add\ y)\ ) =$

$v(x) * (INT_v(ONE)\ I(add)\ INT_v(y)) =$

$4 * (I(ONE) + v(y)) =$

$4 * (1 + 5) = 4 * 6 = error$

Zgodnie z definicją funkcji interpretacji formuł $INT_v(\alpha)$ wartość formuły

$(x\ less\ (ONE\ add\ y))$

przy wartościowaniu *v* oblicza się następująco:

$INT_v(\ (x\ less\ (ONE\ add\ y))\ ) =$

$INT_v(x)\ I(less)\ INT_v(\ (ONE\ add\ y)\ ) =$

$v(x) < (INT_v(ONE)\ I(add)\ INT_v(y)) =$

$4 < (I(ONE) + v(y)) =$

$4 < (1 + 5) = 4 < 6 = \mathbf{P}$

10.10. Rachunek kwantyfikatorów z równością

Język rachunku predykatów może być wykorzystywany w różnych konkretnych celach. W takich przypadkach wprowadzanym symbolom funkcji i predykatów nadaje się specyficzną interpretację, na przykład tak, jak przedstawiono to w poprzednim podrozdziale. Ważnym przykładem często spotykanego symbolu predykatu jest *predykat równości* lub *identyczności*. Zawierający ten predykat język rachunku predykatów nazywa się *rachunkiem predykatów z równością* lub *identycznością*. Dwuargumentowy predykat identyczności, reprezentowany poniżej przez symbol $\approx$, ma wyrażać równość wartości termów. Dla wprowadzanego predykatu równości celowo przyjęto symbol $\approx$, aby odróżniać go od symbolu $=$, występującego w metajęzyku definiującym semantykę języka zawierającego symbol $\approx$.

Semantyka predykatu $\approx$ jest zdefiniowana następująco:

$INT_v(t_1 \approx t_2) = \mathbf{P}$ wtedy i tylko wtedy, gdy $INT_v(t_1) = INT_v(t_2)$

Z podanej definicji interpretacji identyczności wynika, że ma ona własności zwrotności, symetryczności i przechodniości, to znaczy dla dowolnych termów t_1, t_2, $t_3 \in TERM(F, V)$ tautologiami są formuły:

$t_1 \approx t_1$ – zwrotność
$t_1 \approx t_2 \Rightarrow t_2 \approx t_1$ – symetria
$t_1 \approx t_2 \wedge t_2 \approx t_3 \Rightarrow t_1 \approx t_3$ – przechodniość

Identyczność ma ponadto własność *ekstensjonalności*, wyrażoną przez tautologie:

$t_1 \approx t_2 \Rightarrow (t[x ::= t_1] \Leftrightarrow t[x ::= t_2])$ – ekstensjonalność względem termów
$t_1 \approx t_2 \Rightarrow (\alpha[x ::= t_1] \Leftrightarrow \alpha[x ::= t_2])$ – ekstensjonalność względem formuł

gdzie: t jest dowolnym termem, a α jest dowolną formułą.

Własności zwrotności, symetryczności i przechodniości określa się mianem specyficznych aksjomatów teorii opisującej identyczność. Przez pojęcie teorii rozumie się język formalny wraz z systemem dowodzenia, czyli tekstowego wyprowadzania nowych formuł na podstawie formuł-aksjomatów.

10.11. Teorie elementarne

Ustalenie konkretnego języka rachunku kwantyfikatorów wiąże się zwykle z zamiarem opisu pewnego fragmentu interesującej rzeczywistości (realnej lub abstrakcyjnej). Język ma z jednej strony opisywać te zjawiska czy własności, które są przedmiotem zainteresowania, a z drugiej strony powinien umożliwiać wyprowadzanie pewnych wniosków.

Konkretność języka oznacza ustalenie jego sygnatury, czyli składni, oraz ustalenie jego modelu interpretacji, czyli semantyki. Wybrany fragment rzeczywistości ma zwykle specyficzne własności, które można wyrazić w postaci pewnych formuł w ustalonym języku. Formuły takie nazywa się *aksjomatami* i są to formuły spełnione w ustalonej interpretacji języka.

Wnioski, jakich wyprowadzenia się oczekuje, mają być formułami stanowiącymi logiczne konsekwencje przyjętych aksjomatów. Pojęcie zbioru konsekwencji jest definiowane następująco:

Jeżeli Φ jest zbiorem formuł, to jego *zbiorem konsekwencji semantycznych* (*logicznych*) jest zbiór formuł:

$$Con(\Phi) =_{def} \{\alpha \in FORM(F, P, V) \mid \Phi \models \alpha\}$$

Zbiór konsekwencji jest zatem zbiorem formuł spełnionych w interpretacji, w której spełnione są aksjomaty.

Zbiór konsekwencji logicznych ma następujące własności:

- $\Phi \subseteq Con(\Phi)$
- Jeżeli $\Phi_1 \subseteq \Phi_2$, to $Con(\Phi_1) \subseteq Con(\Phi_2)$.
- $Con(Con(\Phi)) = Con(\Phi)$

Uwaga

Druga z własności wyraża monotoniczność konsekwencji logicznej. Jest to ważna własność, której nie mają niektóre logiki nieklasyczne, mające zastosowanie między innymi w budowie systemów ekspertowych. Oznacza to, że w przypadku dołączenia do zbioru aksjomatów dodatkowego aksjomatu może się okazać, że nie wszystkie wcześniej wyprowadzone konsekwencje pozostaną konsekwencjami rozszerzonego zbioru aksjomatów.

W szczególności zbiór wszystkich tautologii to $Con(\varnothing)$, co oznacza, że zbiór tautologii jest podzbiorem $Con(\Phi)$ dla dowolnego Φ.

Do efektywnego wyprowadzania nowych formuł na podstawie formuł-aksjomatów służy pewien system dowodowy. Istotą tego systemu jest to, że wyprowadzenia nowych formuł dokonuje się na podstawie tekstowego przetwarzania formuł, bez analizy semantycznej. Na system dowodowy składają się dwa elementy – pewien zbiór formuł, nazywanych aksjomatami, oraz zbiór reguł wnioskowania. Systemy dowodowe będą omawiane w następnych rozdziałach.

Definicja pewnej teorii polega na wprowadzeniu jej *aksjomatów specyficznych*. Przykładem takiej teorii jest teoria relacji mniejszości.

Przykład 10.8

Teoria jest oparta na dwóch predykatach: *równości* i *mniejszości*, reprezentowanych przez symbole $\approx$ oraz $\prec$. Symbole te przyjęto tylko na użytek rozważanego przykładu, aby podkreślić ich ogólność i nie kojarzyć wyłącznie z konkretną dziedziną, na przykład z równością i mniejszością w dziedzinie liczb. Teoria jest w pełni zdefiniowana przez podane niżej grupy specyficznych aksjomatów.

Pierwsza grupa aksjomatów jest powtórzeniem wyżej sformułowanych własności i oznacza, że równość jest relacją równoważności, czyli dla dowolnych zmiennych indywiduowych x, y, z zachodzi:

$$\forall x \bullet x \approx x$$

$$\forall x \bullet \forall y \bullet x \approx y \Rightarrow y \approx x$$

$$\forall x \bullet \forall y \bullet \forall z \bullet x \approx y \wedge y \approx z \Rightarrow x \approx y$$

Druga grupa określa, że relacja równości nie zmienia innych predykatów, w tym przypadku zachowuje relację mniejszości:

$$\forall x \bullet \forall y \bullet \forall z \bullet x \approx y \wedge x \prec z \Rightarrow y \prec z$$

$$\forall x \bullet \forall y \bullet \forall z \bullet x \approx y \wedge z \prec x \Rightarrow z \prec y$$

Aksjomaty następnej grupy oznaczają, że relacja mniejszości jest relacją ścisłego porządku, to znaczy jest antysymetryczna, przechodnia i spójna:

$$\forall x \bullet \forall y \bullet x \prec y \Rightarrow \neg(y \prec x)$$

$$\forall x \bullet \forall y \bullet x \prec y \wedge y \prec z \Rightarrow x \prec z$$

$$\forall x \bullet \forall y \bullet \forall z \bullet x \approx y \vee x \prec y \vee y \prec x$$

Ponadto jest porządkiem *gęstym*, to znaczy że między dwoma elementami x, y takimi, że $x \prec y$, istnieje jeszcze trzeci element z taki, że $x \prec z$ oraz $z \prec y$:

$$\forall x \bullet \forall y \bullet x \prec y \Rightarrow \exists z \bullet x \prec z \wedge z \prec y$$

Nie istnieje dla tej relacji element najmniejszy, ani największy:

$$\neg\exists x \bullet \forall y \bullet x \approx y \vee x \prec y$$

$$\neg\exists x \bullet \forall y \bullet x \approx y \vee y \prec x$$

Rozpatruje się jeszcze inny przykład teorii wprowadzającej specyficzne aksjomaty. Jest to teoria Peana[19] opisująca liczby naturalne.

[19] Giuseppe Peano (1958–1932).

Przykład 10.9

Teoria liczb naturalnych Peana jest rozszerzeniem teorii następnika. *Teoria następnika* jest oparta na symbolu jednej stałej 0, jednego symbolu funkcyjnego jednoargumentowego *succ* oraz symbolu poprzednio zdefiniowanego predykatu równości $\approx$. Lista aksjomatów teorii, oprócz aksjomatów definiujących równość, jest następująca:

$$\forall x \bullet \exists y \bullet y \approx x$$

$$\forall x \bullet \neg(0 \approx succ(x))$$

$$\forall x \bullet \forall y \bullet succ(x) \approx succ(y) \Rightarrow x \approx y$$

$$(\alpha[x ::= 0] \wedge \forall x \bullet \alpha \Rightarrow \alpha[x ::= succ(x)]) \Rightarrow \forall x \bullet \alpha$$

Ostatnia formuła nie jest aksjomatem, lecz *schematem aksjomatu*, gdyż występująca w niej formuła α może być dowolną formułą rachunku kwantyfikatorów. Jest to – omówiony już wcześniej – *schemat indukcji*.

Teorią liczb naturalnych Peana jest *system arytmetyki naturalnej*, wprowadzający dodatkowy zestaw aksjomatów charakteryzujących działania dodawania i mnożenia, reprezentowanych symbolami + oraz $*$.

$$\forall x \bullet x + 0 \approx x$$

$$\forall x \bullet \forall y \bullet x + succ(y) \approx succ(x + y)$$

$$\forall x \bullet x * 0 \approx 0$$

$$\forall x \bullet \forall y \bullet x * succ(y) \approx x * y + x$$

W teorii z dodatkowymi aksjomatami specyficznymi nabiera właściwego sensu pojęcie konsekwencji semantycznej. Jeżeli α jest pewną formułą, a Φ jest zbiorem aksjomatów, to można pytać czy $\Phi \models \alpha$. Formuła α może być konsekwencją aksjomatów teorii, ale nie musi być tautologią. W takim przypadku oznacza to, że α nie jest spełniona we wszystkich modelach rachunku kwantyfikatorów, ale tylko w tych modelach, które akceptują szczególną interpretację pewnych symboli predykatów lub funkcji wyrażoną przez aksjomaty. Na przykład w teorii relacji mniejszości konsekwencją semantyczną zbioru jej aksjomatów jest formuła

$$\forall x \bullet \exists y \bullet y \prec x$$

10.12. Teorie nieelementarne

Przedstawione wyżej przykłady są przykładami *teorii elementarnych*. Za elementarną uważa się teorię, która powstaje przez dołączenie do języka rachunku kwantyfikato-

rów specyficznych aksjomatów charakteryzujących specyficzne symbole funkcji i predykatów, ale która nie zawiera pojęcia przynależności elementu do zbioru oraz w której nie można mówić o dowolnych zbiorach rozważanych elementów. Przykładem teorii nieelementarnej jest arytmetyka liczb naturalnych.

Przykład 10.10

Teoria ta wprowadza – tak samo jak elementarna arytmetyka Peana – symbole stałej 0, funkcji następnika *succ* oraz symbol predykatu równości $\approx$. Ponadto wprowadza symbol zbioru liczb naturalnych *Nat*, symbol jednoargumentowego predykatu *IsSet*(z), który stwierdza czy z jest zbiorem, oraz symbol dwuargumentowego predykatu przynależności elementu do zbioru $x \in z$. Aksjomatami tej teorii są wszystkie aksjomaty teorii zbiorów zdefiniowane w podrozdziale 4.1 oraz formuły:

$0 \in Nat$

$\forall x \bullet x \in Nat \Rightarrow succ(x) \in Nat$

$\forall x \bullet x \in Nat \Rightarrow \neg(0 \approx succ(x))$

$\forall x \bullet \exists y \bullet succ(x) \approx succ(y) \Rightarrow x \approx y$

$IsSet(Nat)$

$\forall z \bullet (IsSet(z) \wedge 0 \in z \wedge \forall u \bullet u \in z \Rightarrow succ(u) \in z) \Rightarrow \forall x \bullet x \in Nat \Rightarrow x \in z)$

Warto zwrócić uwagę, że wartościami zmiennej indywiduowej z, która występuje w ostatnim aksjomacie, mogą być dowolne zbiory.

Uwaga

W dalszym ciągu predykat równości, zamiast symbolem $\approx$, będzie oznaczany powszechnie używanym symbolem $=$.

W praktyce często korzysta się z rozszerzonego języka kwantyfikatorów, w którym zbiory występują jawnie w powiązaniu z kwantyfikatorami. Oprócz dotychczas omawianych kwantyfikatorów zwykłych, używa się *kwantyfikatorów o ograniczonym zakresie*. Kwantyfikatory te mają postać:

$\forall x \in X \bullet \alpha$ oraz $\exists x \in X \bullet \alpha$

gdzie: X jest pewnym ustalonym zbiorem, a α jest dowolną formułą.

Przykład 10.11

Kwantyfikatory o ograniczonym zakresie są wygodne w wyrażaniu wielu własności związanych z konkretną dziedziną interpretacji:

Dla każdej liczby naturalnej n istnieje liczba rzeczywista x taka, że $x^2 = n$:

$\forall n \in Nat \bullet \exists x \in Rzeczywiste \bullet x^2 = n$

Dla każdej liczby całkowitej a istnieje liczba wymierna x taka, że $a < x < a + 1$

$\forall n \in$ *Całkowite* $\bullet \exists x \in$ *Wymierne* $\bullet\ a < x < a + 1$

Rozszerzona notacja jest przydatna do opisu sytuacji związanych z pewnym konkretnym obszarem zainteresowania. Wyraża się to przez wprowadzone zbiory – dziedziny interpretacji, a także przez interpretację symboli funkcyjnych i predykatywnych występujących w formułach z kwantyfikatorami o ograniczonym zasięgu. Tak właśnie jest w przedstawionym powyżej przykładzie, gdzie wyrażenia x^2, $a + 1$, $a < x$ mają znaną interpretację arytmetyczną.

Należy pamiętać, że z rozszerzoną notacją wiąże się zawężenie semantyki. Wprowadzenie konkretnych zbiorów narzuca mianowicie dziedzinę interpretacji. Oznacza to, że formuły spełnione przy założeniu konkretnych zbiorów nie muszą być spełnione w innych dziedzinach interpretacji.

Ćwiczenia

1. W układzie współrzędnych *Oxy* zaznaczyć obszary, w których prawdziwe są następujące funkcje zdaniowe:
   a) $|x * y| = 1$
   b) $x \geq |y|$
   c) $|x * y| < 0 \Rightarrow x*x + y*y \geq 1$
   d) $x \geq |y| \Rightarrow x*x + y*y = 1$
   e) $sin(x) > |y|$

   Zakłada się, że występujące w zadaniu symbole funkcyjne i predykatowe mają standardową interpretację.

2. Dana jest sygnatura $Sig = <F, P>$ języka rachunku kwantyfikatorów, w której:

   $F =_{def} \{f_0\} \cup \{f_1, g_1\} \cup \{f_2, g_2, h_2\}$ jest zbiorem symboli funkcyjnych,
   $P =_{def} \{p_0\} \cup \{p_1, q_1\} \cup \{p_2, q_2\}$ jest zbiorem symboli predykatów,

   dolny indeks zaś wskazuje liczbę argumentów. Podać gramatykę definiującą zbiór termów i gramatykę definiującą zbiór formuł języka o podanej sygnaturze.

3. Niech f, g, h będą symbolami funkcyjnymi, p, q – symbolami predykatów, x, y, z – zmiennymi indywiduowymi. Wskazać wolne i związane wystąpienia zmiennych indywiduowych w formułach:
   a) $\forall x \bullet \forall y \bullet p(f(x, y), z) \wedge \forall x \bullet q(x, z, h(x, y))$
   b) $(\forall x \bullet \exists y \bullet q(x, z) \vee p(h(x, y)) \Rightarrow p(f(x, y), z)$
   c) $\forall x \bullet p(h(x), z) \Rightarrow (\exists z \bullet (\exists y \bullet q(f(h(x), z) \wedge \forall z \bullet p(z, y) \Leftrightarrow q(x, y))$

4. Zdefiniować funkcję, która dla dowolnej formuły α rachunku kwantyfikatorów określa zbiór wszystkich zmiennych indywiduowych, które w formule α mają:
   a) parzystą liczbę wystąpień wolnych,
   b) jednocześnie wystąpienia wolne i wystąpienia związane,
   c) dokładnie taką samą liczbę wystąpień wolnych jak liczbę wystąpień związanych.

5. Niech $\{=, \leq, <\}$ będzie zbiorem symboli predykatów oraz $\{+, *, /\}$ – zbiorem symboli funkcyjnych określonych na liczbach naturalnych. Dla symboli tych przyjmujemy standardową interpretacją arytmetyczną. Korzystając z tego zestawu symboli oraz z symboli stałych liczbowych, zapisać formuły reprezentujące następujące wypowiedzi:
   a) x jest liczbą podzielną przez ustaloną liczbę $n > 0$,
   b) x jest sumą kwadratów dwóch liczb naturalnych,
   c) x jest liczbą pierwszą,
   d) x nie jest liczbą pierwszą,
   e) x jest najmniejszą wspólną wielokrotnością liczb y i z,
   f) x przy dzieleniu przez 4 daje resztę 1 lub 2,
   g) każda liczba przy dzieleniu przez inną liczbę daje resztę 0 lub 1,
   h) każda liczba parzysta większa od 3 jest sumą dwóch liczb pierwszych,
   i) każde trzy liczby mają największy wspólny dzielnik,
   j) nie istnieje największa liczba naturalna.

   Zapisać zaprzeczenia powyższych formuł w taki sposób, aby nie zaczynały się od negacji.

6. Zakładając, że są do dyspozycji symbole predykatów określone w poprzednim zadaniu oraz symbol przynależności elementu do zbioru, wyrazić w postaci formuł następujące wypowiedzi:
   a) zbiór X ma dokładnie jeden element,
   b) zbiór X ma dokładnie trzy elementy,
   c) zbiór X ma przynajmniej dwa elementy.

7. Podać formalną definicję sformułowania: *istnieje dokładnie jedno x takie, że spełniona jest formuła* α.

8. Podać przykłady predykatów $p(x)$, $q(x)$, dla $x \in$ *Rzeczywiste*, dla których podane niżej formuły są zawsze prawdziwe albo zawsze fałszywe:
   a) $\forall \bullet (p(x) \vee q(x))$
   b) $\forall \bullet (p(x) \Rightarrow q(x))$
   c) $\exists \bullet (p(x) \wedge q(x))$
   d) $\exists \bullet (p(x) \Rightarrow q(x))$

9. Które z poniższych stwierdzeń są prawdziwe? Jeżeli $INT_v(\alpha \vee \beta) = prawda$, to:
   a) $INT_v(\alpha) = prawda$ lub $INT_v(\beta) = prawda$,
   b) $INT_v(\alpha) = prawda$ oraz $INT_v(\beta) = prawda$,
   c) dla każdego v' różniącego się od v wartościowaniem zmiennej x, zachodzi $INT_{v'}(\alpha) = prawda$ oraz $INT_{v'}(\alpha) = prawda$.

10. Które z poniższych stwierdzeń są prawdziwe? Jeżeli $INT_v(\alpha \wedge \beta) = fałsz$, to:
    a) $INT_v(\alpha) = fałsz$ oraz $INT_v(\beta) = fałsz$,
    b) istnieje takie v' różniące się od v wartościowaniem pewnej zmiennej x, że $INT_{v'}(\alpha) = prawda$ lub $INT_{v'}(\alpha) = prawda$,
    c) dla każdego v' różniącego się od v wartościowaniem zmiennej x zachodzi $INT_{v'}(\alpha) = fałsz$ oraz $INT_{v'}(\alpha) = prawda$.

11. Dana jest formuła $\exists x \bullet p(x, y)$. Interpretacja symbolu predykatu p jest wyrażona przez system relacyjny $SR =_{df} <A_{SR}, R_{SR}>$, gdzie A_{SR} jest zbiorem, zwanym nośnikiem, a R_{SR} jest relacją binarną. Interpretacja przypisuje symbolowi p relację R_{SR}.

    Uwaga: Interpretacja n-argumentowego symbolu predykatu q jako pewnej n-elementowej relacji $R_q \subseteq A^n$ na zbiorze A jest równoważna interpretacji tego symbolu jako funkcji n-argumentowej $f_q : A^n \rightarrow \{prawda, fałsz\}$. Dlaczego?

    Które z poniższych stwierdzeń są prawdziwe. Jeżeli nośnik systemu relacyjnego $A_{SR} =_{def} \{a, b\}$ i relacja $R_{SR} =_{def} \{<a, a>, <b, b>, <a, b>\}$, to dla wartościowania v takiego, że:
    a) $v(x) = a$ i $v(y) = b$ formuła jest spełniona,
    b) $v(x) = a$ i $v(y) = a$ formuła nie jest spełniona,
    c) $v(x) = b$ i $v(y) = b$ formuła jest spełniona,
    d) $v(x) = b$ i $v(y) = a$ formuła nie jest spełniona.

12. Dla każdej z poniższych formuł podaj interpretacje, w których formuła jest (a) spełniona dla każdego wartościowania, (b) nie jest spełniona dla każdego wartościowania, (c) dla niektórych wartościowań jest spełniona, a dla pozostałych nie jest spełniona:
    a) $\forall x \bullet \exists y \bullet p(x, y)$
    b) $\exists x \bullet \forall y \bullet p(x, y, z)$
    c) $\exists x \bullet (q(x, y) \Rightarrow q(x, y))$

13. Następujące formuły sprowadzić do przedrostkowej postaci normalnej (*PNF*):
    a) $\forall x \bullet ((x > 2) \wedge (\exists y \bullet (x < y)))$
    b) $(x < y) \wedge \exists x \bullet \forall y \bullet (x > y)$
    c) $\forall x \bullet (\exists y \bullet (y > 2) \wedge (y < x))$

14. Sprowadzić do przedrostkowej postaci normalnej negacje formuł z poprzedniego zadania.

15. Przedstawić predykaty p oraz q określone na zbiorze liczb całkowitych, dla których poniższe zdania są fałszywe:
    a) $(\exists x \bullet \exists y \bullet p(x, y)) \Rightarrow (\forall y \bullet \exists x \bullet p(x, y))$
    b) $(\exists x \bullet \exists y \bullet p(x, y)) \Rightarrow (\exists x \bullet p(x, x))$
    c) $(\forall x \bullet \exists y \bullet p(x, y)) \Rightarrow (\forall x \bullet p(x, x))$
    d) $(\forall x \bullet p(x, x)) \Rightarrow (\forall x \bullet \forall y \bullet p(x, y))$
    e) $(\exists x \bullet \forall y \bullet p(x, y)) \Rightarrow (\forall x \bullet \forall y \bullet p(x, y))$
    f) $(\forall x \bullet p(x) \vee q(y)) \Rightarrow (\forall x \bullet p(x) \vee \forall y \bullet q(y))$
    g) $(\exists x \bullet p(x) \wedge \exists x \bullet q(x)) \Rightarrow (\exists x \bullet (p(x) \wedge q(x))$

11. Rachunek sekwentów Gentzena

11.1. Wstęp

W poprzednim rozdziale wprowadzono pojęcie semantycznej konsekwencji. Bezpośrednie sprawdzenie – na podstawie definicji – czy dana formuła rachunku kwantyfikatorów jest, czy nie jest semantyczną konsekwencją pewnego zbioru formuł rachunku kwantyfikatorów, nie jest możliwe, gdyż wymagałoby to sprawdzenia nieskończonej liczby modeli danego zbioru formuł. Praktyczne badanie, czy formuła jest konsekwencją semantyczną pewnego zbioru formuł, opiera się na pojęciu *konsekwencji składniowej*. Pojęcie konsekwencji składniowej jest pewnym odpowiednikiem pojęcia konsekwencji semantycznej. Pojęcie konsekwencji składniowej będzie przedstawione w ramach *rachunku sekwentów Gentzena*. Rachunek ten – opracowany przez Gentzena[20] w latach trzydziestych XX wieku – wyznacza jeden z efektywniejszych systemów automatycznego dowodzenia twierdzeń.

Istota podejścia opartego na pojęciu konsekwencji składniowej polega na zdefiniowaniu pewnego systemu generowania napisów. System taki, nazywany systemem dowodowym, zawiera dwa elementy – zbiór *aksjomatów* Ψ i zbiór *reguł wyprowadzania* (albo inaczej: *reguł wnioskowania* lub *inferencji*). Aksjomatami są pewne napisy, reguły zaś określają, w jaki sposób na podstawie pewnych napisów otrzymać nowe napisy, na przykład jak z pewnego zbioru formuł otrzymać nowe formuły.

W celu dowiedzenia, że – w danym systemie dowodowym – formuła α jest konsekwencją semantyczną zbioru formuł Φ postępuje się następująco: Stosując reguły wyprowadzania, wyprowadza się nowe formuły ze zbioru Φ oraz zbioru aksjomatów Ψ i powtarza się tę czynność tak długo, aż zostanie wyprowadzona formuła α. Tak skonstruowany dowód nazywa się dowodem *wprost* i służy do pokazania, że wynikanie logiczne $\Phi \models \alpha$ zachodzi w dowolnym modelu.

Możliwy jest również dowód *nie wprost* faktu, że $\Phi \models \alpha$. Dowód taki polega na doprowadzeniu do sprzeczności na podstawie założenia, że zbiór formuł $\Phi \cup \{\neg\alpha\}$ jest spełnialny, to znaczy na podstawie założenia, że formuła α nie jest spełniona dla pew-

[20] Gerhard Gentzen (1909–1945).

nego modelu zbioru formuł Φ. Stwierdzenie sprzeczności oznacza, że zbiór $\Phi \cup \{\neg\alpha\}$, wbrew założeniu, jest spełnialny, a to – na podstawie twierdzenia o dedukcji z poprzedniego rozdziału – daje podstawę do ostatecznego orzeczenia, że $\Phi \models \alpha$.

Jeżeli dany jest pewien system dowodzenia S, to fakt, że formuła α została wyprowadzona w tym systemie, na podstawie zbioru formuł Φ będzie oznaczany:

$$\Phi \vdash_S \alpha$$

Istnieje wiele systemów dowodzenia opartego na pojęciu konsekwencji składniowej. Niektóre spośród nich są związane z dowodzeniem wprost, inne z dowodzeniem nie wprost. System dowodzenia Gentzena może być rozważany zarówno jako system do dowodzenia wprost, jak i do dowodzenia nie wprost. Zwykle dowody tworzone w systemie Gentzena są oparte na idei budowy dowodu nie wprost, ale zbudowany dowód jest odczytywany jako dowód wprost. Przedstawiany tu system dowodzenia Gentzena opiera się na idei konstrukcji dowodów nie wprost.

Od każdego systemu dowodzenia wymaga się w pierwszej kolejności, aby nie prowadził on do fałszywych wniosków. Własność taką określa się mianem *semantycznej poprawności* (albo *niesprzeczności*) systemu dowodzenia. Oznacza to – przy dowodzeniu wprost – że jeżeli na podstawie zbioru formuł Φ wygeneruje się formułę α, to zachodzi $\Phi \models \alpha$, a przy dowodzeniu nie wprost – że jeżeli na podstawie założenia o spełnialności zbioru formuł $\Phi \cup \{\neg\alpha\}$ uzyska się sprzeczność, to również $\Phi \models \alpha$. Własność semantycznej poprawności systemu dowodzenia S można sformułować w postaci

jeżeli $\Phi \vdash_S \alpha$, to $\Phi \models \alpha$.

Drugą oczekiwaną własnością systemu dowodzenia – odwrotną w stosunku do własności semantycznej poprawności – jest *zupełność semantyczna*. Własność ta oznacza – przy dowodzeniu wprost – że jeżeli zachodzi $\Phi \models \alpha$, to zawsze w systemie istnieje wyprowadzenie formuły α ze zbioru formuł Φ. Przy dowodzeniu nie wprost własność ta oznacza, że jeżeli zachodzi $\Phi \models \alpha$, to z założenia o spełnialności zbioru formuł $\Phi \cup \{\neg\alpha\}$ zawsze uzyska się sprzeczność. Własność semantycznej zupełności systemu dowodzenia S można sformułować w postaci

jeżeli $\Phi \models \alpha$, to $\Phi \vdash_S \alpha$.

Z nazwiskiem Gentzena wiążą się dwa systemy dowodzenia: jeden jest nazywany systemem dedukcji naturalnej, drugi – rachunkiem sekwentów. W tym rozdziale przedstawiono tylko rachunek sekwentów. Rachunek ten odzwierciedla w znacznym stopniu sposób postępowania stosowany w praktyce matematycznej. Jak wspomniano, konstrukcja dowodów opiera się na idei dowodzenia nie wprost, a ponadto opiera się na obserwacji, że konstruując dowód pewnego twierdzenia, próbuje się zdekomponować go na zestaw prostszych dowodów (poddowodów lub dowodów podporządkowanych). W przypadku rachunku kwantyfikatorów konstrukcja poddowodów wynika ze składni formuł.

11.2. Lemat o podstawieniu

Poniżej przedstawiany lemat o podstawieniu będzie wykorzystywany w następnych podrozdziałach. Wskazuje on na rolę, jaką odgrywają wartościowania w interpretacji termów i formuł.

Lemat 11.1

Niech t będzie termem nad sygnaturą *Sig* oraz niech $\mathcal{M} = \langle D, I \rangle$ będzie modelem interpretacji, a v_1, v_2 niech będą dwoma wartościowaniami. Jeżeli zachodzi równość wartościowań $v_1(x) = v_2(x)$ dla wszystkich zmiennych $x \in Var(t)$, to

$$INT_{v_1}(t) = INT_{v_2}(t).$$

Dowód

Dowód prowadzi się metodą indukcji strukturalnej względem termów. Jeżeli term t jest postaci x, to $INT_{v_1}(x) = v_1(x) = v_2(x) = INT_{v_2}(x)$. Jeżeli term t jest postaci $f(t_1, ..., t_n)$, to na mocy założenia indukcyjnego

$$INT_{v_1}(t_i) = INT_{v_2}(t_i) \quad \text{dla } i = 1, ..., n$$

stąd

$$INT_{v_1}(f(t_1,...,t_n)) = I(F)(INT_{v_1}(t_1), ..., INT_{v_1}(t_n))$$

$$= I(f)(INT_{v_2}(t_1), ..., INT_{v_2}(t_n))$$

$$= INT_{v_2}(f(t_1, ..., t_n)) \qquad \blacksquare$$

Lemat o podstawieniu wskazuje na związek, jaki zachodzi pomiędzy zmianą wartościowania zmiennych indywiduowych przez podstawienie za wskazaną zmienną wartość termu a tekstowym zastąpieniem tej zmiennej termem w innym termie lub w formule.

Lemat 11.2 (*Lemat o podstawieniu*)

Niech $\mathcal{M} = \langle D, I \rangle$ będzie modelem, v – pewnym wartościowaniem, t – termem, oraz niech $v' = v[x := INT_v(t)]$. Dla dowolnego termu t' oraz dla dowolnej formuły α zachodzą wówczas następujące własności:

1. $INT_{v'}(t') = INT_v(t'[x ::= t])$
2. $INT_{v'}(\alpha) = INT_v(\alpha[x ::= t])$ pod warunkiem, że $x \in FV(\alpha)$.

Dowód

Dowód prowadzi się metodą indukcji strukturalnej najpierw względem składni termów, a następnie formuł.

1. Niech term $t' \equiv y$, gdzie zmienna y jest różna od zmiennej x, wówczas

   $$INT_{v'}(y) = v'(y) = v(y) = v(y[x ::= t]) = INT_v(y[x ::= t])$$

   Dla $t' \equiv x$, zachodzi

   $$INT_{v'}(x) = v'(x) = INT_v(t) = INT_v(x[x ::= t])$$

   Dla $t' \equiv f(t_1, ..., t_n)$, na mocy założenia indukcyjnego, zachodzi

   $$INT_{v'}(t_i) = INT_v(t_i[x ::= t]) \text{ dla } i = 1, ..., n,$$

   stąd

   $$\begin{aligned} INT_{v'}(f(t_1, ..., t_n)) &= I(f)\,(INT_{v'}(t_1), ..., INT_{v'}(t_n)) \\ &= I(f)\,(INT_v(t_1[x ::= t]), ..., INT_v(t_n[x ::= t])) \\ &= INT_v\,(f(t_1, ..., t_n)[x ::= t]) \end{aligned}$$

2. Niech formuła α będzie postaci $\forall y \bullet \beta$ oraz niech term t będzie wolny w α ze względu na x. Należy rozpatrzeć dwa przypadki: $x \neq y$ oraz $x = y$.

   W pierwszym przypadku, gdy $x \neq y$, dla każdego $d \in \boldsymbol{D}$ zachodzi

   $$INT_{v'}(\forall y \bullet \beta) = INT_{v'[y := d]}(\beta)$$

   Należy zauważyć, że

   $$v'[y := d] = v[x := INT_v(t)][y := d] = v[y := d][x := INT_v(t)]$$

   Na mocy poprzedniego lematu zachodzi

   $$INT_v(t) = INT_{v[y := d]}(t)$$

   ponieważ $y \notin Var(t)$. Dalej niech $a = INT_v(t)$, stąd i z założenia indukcyjnego wynika

   $$\begin{aligned} INT_{v[y := d][x := a]}(\beta) &= INT_{v[y := d]}(\beta[x ::= t]) \\ &= INT_v(\forall y \bullet \beta[x ::= t]) \\ &= INT_v\,(\forall y \bullet \beta)[x ::= t] \end{aligned}$$

   W drugim przypadku, gdy $x = y$, dla każdego $d \in \boldsymbol{D}$ zachodzi

   $$INT_{v'}(\forall y \bullet \beta) = INT_{v'[x := d]}(\beta)$$

   Ponieważ

   $$v[x := INT_v(t)][x := d] = v[x := d]$$

   oraz

   $$\forall y \bullet \beta = (\forall y \bullet \beta)[x ::= t])$$

   dla każdego $d \in \boldsymbol{D}$ zachodzi

$$INT_{v'}(\forall y \bullet \beta) = INT_{v[x := d]}(\beta)$$
$$= INT_{v}(\forall y \bullet \beta)$$
$$= INT_{v}(\forall y \bullet \beta)[x ::= t])$$

Pozostałe przypadki, gdy formuła α ma inne postaci, pozostawia się do udowodnienia Czytelnikowi. ■

Przykład 11.1

Niech sygnatura *Sig* składa się z jednej stałej 0, dwóch jednoargumentowych operacji *nast*, *pop* oraz jednego symbolu predykatu dwuargumentowego $\_\approx\_$. Dziedziną interpretacji $\boldsymbol{D}$ niech będzie zbiór liczb całkowitych *Całkowite*. Interpretacja $\boldsymbol{I}$ stałej 0 przyporządkowuje liczbę zero, operacjom *nast* i *pop* przyporządkowuje dodawanie i odejmowanie jedynki, to znaczy:

$\boldsymbol{I}(nast)(n) =_{def} n + 1$,
$\boldsymbol{I}(pop)(n) =_{def} n - 1$,

a predykatowi $\_\approx\_$ przyporządkowuje równość liczb.

Niech dana będzie formuła

$pop(pop(x)) \approx 0$

oraz term

$nast(nast(0))$

Dla dowolnego wartościowania v i wartościowania $v' = v[x := INT_v(nast(nast(0)))]$ zachodzi:

$$INT_{v'}(pop(pop(x)) \approx 0) = INT_{v}(pop(pop(x)) \approx 0[x ::= nast(nast(0))])$$
$$= INT_{v}(pop(pop(nast(nast(0)))) \approx 0) = \mathbf{P}$$

Ostatnia równość zachodzi, ponieważ:

$INT_{v}(nast(nast(0))) = 2$ oraz $INT_{v}(pop(pop(nast(nast(0))))) = 0$.

Wniosek 11.1

Niech α będzie formułą oraz niech term t będzie wolny ze względu na x w α. Na podstawie lematu i definicji semantyki kwantyfikatorów zachodzi:

1. $\forall x \bullet \alpha \models \alpha[x ::= t]$
2. $\alpha[x ::= t] \models \exists x \bullet \alpha$

Punkt 1. wniosku, z faktu spełnialności formuły $\forall x \bullet \alpha$, pozwala na wyprowadzanie wniosku o spełnialności dowolnej formuły postaci $\alpha[x ::= t]$ dla dowolnego termu t. Inaczej, aby pokazać, że formuła $\forall x \bullet \alpha$ nie jest spełnialna, wystarczy pokazać, że formuła $\alpha[x ::= t]$ nie jest spełnialna dla pewnego t.

Punkt 2. wniosku, z faktu spełnialności formuły $\alpha[x ::= t]$, pozwala na wyprowadzanie wniosku o spełnialności formuły $\exists x \bullet \alpha$. Inaczej, aby pokazać, że formuła $\exists x \bullet \alpha$ nie jest spełnialna, należy pokazać, że formuła $\alpha[x ::= t]$ nie jest spełnialna dla dowolnego t.

11.3. Przykłady wprowadzające

Przed formalnym przedstawieniem rachunku sekwentów Gentzena, przykłady wprowadzające pozwolą na poznanie głównych idei, na których opiera się rachunek.

Przykład 11.2

Rozpatruje się następującą formułę rachunku zdań

$$((a \Rightarrow b) \wedge (c \Rightarrow d) \wedge (a \wedge c)) \Rightarrow (b \wedge d) \tag{1}$$

Formuła ta – jak można sprawdzić, na przykład metodą zero-jedynkową – jest tautologią. Istota dowodu nie wprost polega na przyjęciu założenia, że formuła (1) nie jest tautologią, czyli wartość formuły jest fałszem dla pewnego wartościowania zbioru zmiennych zdaniowych $\{a, b, c, d\}$. Okaże się, że poszukiwanie takiego wartościowania doprowadzi do sprzeczności. W tym celu zadanie rozbija się na dwa, prostsze, podzadania. Zgodnie z twierdzeniem o rozbiorze, przedstawionym w poprzednim rozdziale, każdą formułę można jednoznacznie zdekomponować na podformuły składowe. Spójnik łączący te podformuły jest nazywany spójnikiem głównym. Głównym spójnikiem formuły (1) jest ostatni po prawej symbol implikacji. Całą formułę (1) można przedstawić w postaci

$$\alpha \Rightarrow \beta \tag{2}$$

gdzie:

$$\alpha \equiv (a \Rightarrow b) \wedge (c \Rightarrow d) \wedge (a \wedge c) \tag{3}$$

$$\beta \equiv b \wedge d \tag{4}$$

Aby pokazać, że formuła (2) jest fałszywa dla pewnego wartościowania, wystarczy pokazać, że dla tego wartościowania podformuła α jest prawdziwa, a podformuła β jest fałszywa – stwierdzenie to wynika ze standardowej interpretacji spójnika implikacji. Formuła (1) została zatem rozbita na dwie podformuły: (3) oraz (4), z zadaniem pokazania prawdziwości podformuły (3) i fałszywości podformuły (4) dla pewnego wartościowania zmiennych logicznych.

Ogólnie, daną formułę można rozbijać na dwa zbiory jej podformuł z zadaniem pokazania, że dla pewnego wartościowania podformuły z jednego zbioru są prawdziwe, a podformuły z drugiego zbioru są fałszywe. Przyjmuje się oznaczenie: jeżeli

$$\{\alpha_1, ..., \alpha_n\}$$

jest zbiorem formuł, które mają być prawdziwe dla pewnego wartościowania, a

$\{\beta_1, ..., \beta_m\}$

jest zbiorem formuł, które mają być fałszywe dla tego samego wartościowania, to takie żądanie będzie zapisywane w postaci

$\alpha_1, ..., \alpha_n \rightarrow \beta_1, ..., \beta_m$

a napis taki będzie nazywany *sekwentem*.

Sekwent jest więc parą dwóch zbiorów formuł. Poszczególne zbiory formuł są zapisywane w postaci list, a symbol $\rightarrow$ jest separatorem oddzielającym dwie listy. W tym zapisie sekwent jest traktowany jako umowna forma zapisu pary zbiorów.

W rozpatrywanym przykładzie początkowe założenie można zatem zapisać w postaci sekwentu

$$\rightarrow ((a \Rightarrow b) \wedge (c \Rightarrow d) \wedge (a \wedge c)) \Rightarrow (b \wedge d) \tag{5}$$

natomiast to, co należy pokazać, jako konsekwencję tego założenie, można zapisać w postaci sekwentu

$$(a \Rightarrow b) \wedge (c \Rightarrow d) \wedge (a \wedge c) \rightarrow b \wedge d \tag{6}$$

Należy zwrócić uwagę na to, że przejściu od zadania (5) do zadania (6) towarzyszy eliminacja jednego spójnika logicznego.

Aby pokazać, że prawdziwa jest lewa stronę sekwentu (6), należy pokazać, że prawdziwe są wszystkie jej składowe połączone symbolem koniunkcji, czyli

$$a \Rightarrow b, c \Rightarrow d, a \wedge c \rightarrow b \wedge d \tag{7}$$

W ten sam sposób, kierując się znaczeniem koniunkcji, zadanie sprowadza się do pokazania

$$a \Rightarrow b, c \Rightarrow d, a, c \rightarrow b \wedge d \tag{8}$$

Po prawej stronie sekwentu (8) jest również spójnik koniunkcji, lecz należy pokazać, że formuła po tej stronie jest fałszywa. Wystarczy więc pokazać tylko jeden z przypadków, że fałszywe jest b albo że fałszywe jest d. Rozwiązanie zadania (8) sprowadza się zatem do rozwiązania jednego z dwóch podzadań:

$$a \Rightarrow b, c \Rightarrow d, a, c \rightarrow b \tag{9a}$$

$$a \Rightarrow b, c \Rightarrow d, a, c \rightarrow d \tag{9b}$$

Rozpatruje się zadanie (9a). Można je sprowadzić do zadania prostszego, eliminując pierwszy z lewej spójnik implikacji. Aby pokazać, że dla pewnego wartościowania prawdziwa jest implikacja $a \Rightarrow b$, wystarczy pokazać jeden z dwóch przypadków, że fałszywe jest a albo że prawdziwe jest b. Zadanie (9a) rozbija się zatem na dwa podzadania:

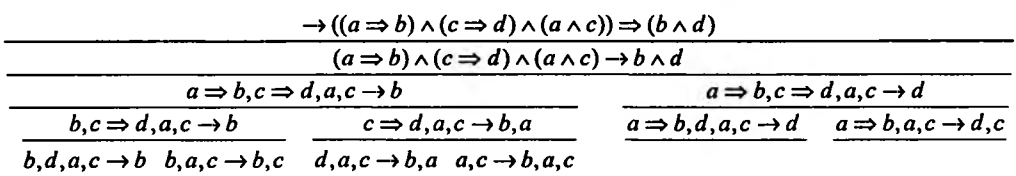
$$c \Rightarrow d, a, c \rightarrow b, a \tag{10a}$$

$$b, c \Rightarrow d, a, c \rightarrow b \tag{10b}$$

Zadanie (10a) prowadzi do sprzeczności. Wynika to z tego, że od zmiennej zdaniowej a wymaga się jednocześnie, by dla tego samego wartościowania była prawdziwa (wystąpienie a lewej stronie) i fałszywa (wystąpienie a po prawej stronie). Podobnie do sprzeczności prowadzi zadanie (10b). Ponieważ zadania (10a) i (10b) były dekompozycją zadania (9a), oznacza to, że również zadanie (9a) prowadzi do sprzeczności.

W analogiczny sposób przeprowadzone rozumowanie w stosunku do zadania (9b) prowadzi także do sprzeczności. Każda zatem próba znalezienia odpowiedniego wartościowania, które potwierdzałoby, że formuła (1) nie jest tautologią, prowadzi do sprzeczności, co daje ostateczny wniosek, że (1) jest tautologią.

Przeprowadzone wnioskowanie można zapisać w postaci graficznej. Na rysunku 11.1 przedstawiono graf-drzewo. Wierzchołkami drzewa są sekwenty, czyli napisy postaci $\Gamma \rightarrow \Delta$, gdzie Γ, Δ są zbiorami formuł zapisywanymi w postaci list. Łuki drzewa są reprezentowane przez poziome linie – przyjmuje się, że są one skierowane z góry w dół. Przejście między dwoma sąsiadującymi wierzchołkami drzewa odpowiada eliminacji jednego spójnika logicznego.



Rys. 11.1. Drzewo dowodowe dla formuły (1)

Korzeniem drzewa jest sekwent postaci

$$\rightarrow \alpha$$

gdzie α jest dowodzoną formułą. W przykładzie

$$\alpha \equiv ((a \Rightarrow b) \wedge (c \Rightarrow d) \wedge (a \wedge c)) \Rightarrow (b \wedge d)$$

Liśćmi drzewa są sekwenty postaci

$$\Gamma, \beta \rightarrow \Delta, \beta$$

gdzie: – jak poprzednio – Γ, Δ są ciągami formuł, a β jest pewną formułą, która występuje po obu stronach sekwentu.

Drzewo, którego liśćmi są wyłącznie sekwenty o tej postaci jest *drzewem dowodowym* tautologii.

Przykład 11.3

Rozpatruje się teraz przykład konstrukcji drzewa dowodu dla formuły

$$((a \Rightarrow b) \wedge (c \Rightarrow d) \wedge a) \Rightarrow (b \wedge c) \tag{11}$$

która nie jest tautologią. Postępując jak poprzednio, można wnioskowanie zapisać w postaci drzewa (rys. 11.2).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\to ((a \Rightarrow b) \wedge (c \Rightarrow d) \wedge a) \Rightarrow (b \wedge c)}}{(a \Rightarrow b) \wedge (c \Rightarrow d) \wedge a \to b \wedge c}}{a \Rightarrow b, c \Rightarrow d, a \to b \wedge c}}{a \Rightarrow b, c \Rightarrow d, a \to c}}{c \Rightarrow d, a \to c, a}\ (1) \qquad \dfrac{b, c \Rightarrow d, a \to c}{\dfrac{b, d \to c}{(2)} \quad \dfrac{b, d, a \to c}{(3)}}}{a \Rightarrow b, c \Rightarrow d, a \to b}\ \dfrac{\ldots\ldots}{(4)}$$

Rys. 11.2. Drzewo dowodowe dla formuły (11)

W drzewie dowodu liść (1) jest sekwentem, który po prawej i lewej stronie ma tę samą formułę – tu zmienną zdaniową. Sekwent ten oznacza sprzeczność.

Liście (2), (3) są natomiast sekwentami postaci

$$b, d \to c \text{ oraz } b, d, a \to c \tag{12}$$

Nie mają one takiej własności jak sekwent (1), ponadto nie zawierają spójników logicznych. Wyznaczają one takie wartościowania zmiennych a, b, c, d, przy których wartość dowodzonej formuły (11) jest fałszem.

Znalezienie takich wartościowań oznacza, że formuła (11) nie jest tautologią. Znalezienie takich wartościowań kończy dowód i nie wymaga rozwijania pozostałych gałęzi drzewa, tu w przykładzie – gałęzi (4).

Oba przykłady można traktować jako systematyczne poszukiwanie kontrprzykładu w celu pokazania, że wyjściowa formuła nie jest tautologią. W pierwszym przykładzie każda możliwa próba znalezienia kontrprzykładu prowadziła do sprzeczności, co dało podstawę do ostatecznego stwierdzenia, że formuła (1) jest tautologią. W drugim przykładzie kontrprzykład taki znaleziono, co dało podstawę do ostatecznego stwierdzenia, że formuła (11) nie jest tautologią.

Przykład 11.4

Rozpatruje się teraz formułę rachunku kwantyfikatorów

$$\forall x \bullet p(x) \Rightarrow \exists y \bullet p(y) \tag{13}$$

gdzie p jest symbolem pewnego jednoargumentowego predykatu. Formuła ta jest oczywiście tautologią. Dokonuje się transformacji formuły polegającej na zastą-

pieniu kwantyfikatora szczegółowego kwantyfikatorem ogólnym. Z praw de Morgana wynika, że zachodzi równoważność semantyczna

$$\exists y \bullet \alpha = \neg\forall y \bullet \neg\alpha$$

Formułę (13) można przedstawić w postaci

$$\forall x \bullet p(x) \Rightarrow \neg\forall y \bullet \neg p(y) \tag{14}$$

Jak w przykładach poprzednich, zakłada się, że formuła nie jest tautologią, co oznacza, że istnieje pewna interpretacja predykatu I, w której formuła (14) – przy pewnym wartościowaniu zmiennych indywiduowych – jest fałszywa. Zgodnie z poprzednim rozumowaniem, poszukiwanie to sprowadza się do znalezienia takiej interpretacji I i takiego wartościowania, dla których formuła $\forall x \bullet p(x)$ jest prawdziwa, a formuła $\neg\forall y \bullet \neg\alpha$ jest fałszywa, czyli formuła $\forall y \bullet \neg\alpha$ jest prawdziwa. Jeżeli interpretacja I jest taka, że formuła $\forall x \bullet p(x)$ jest w niej spełnialna, to na podstawie własności pokazanej w poprzednim punkcie

$$\forall x \bullet \alpha \vDash \alpha[x ::= z]$$

można stwierdzić, że I spełnia również formułę $p(z)$, gdyż z jest zmienną wolną w $p(x)$.

Podobnie, stosując analogiczne rozumowanie w stosunku do formuły $\forall y \bullet \neg\alpha$, można stwierdzić, że I spełnia formułę $\neg p(z)$, czyli że I nie spełnia formuły $p(z)$.

Od $p(z)$ oczekuje się zatem, że w interpretacji I jest jednocześnie prawdziwe i fałszywe, co kończy dowód.

Graficzna reprezentacja tego dowodu jest pokazana na rysunku 11.3.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\rightarrow \forall x \bullet p(x) \Rightarrow \neg\forall y \bullet \neg p(y)}{\forall x \bullet p(x) \rightarrow \neg\forall y \bullet \neg p(y)}}{\forall x \bullet p(x), \forall y \bullet \neg p(y) \rightarrow}}{p(x)[x ::= z], \neg p(y)[y ::= z] \rightarrow}}{p(z) \rightarrow p(z)}$$

Rys. 11.3. Drzewo dowodowe dla formuły (13)

Przykład 11.5

Rozpatrzmy formułę rachunku kwantyfikatorów

$$\exists y \bullet p(y) \Rightarrow \forall x \bullet p(x) \tag{15}$$

gdzie p jest symbolem pewnego jednoargumentowego predykatu. Formuła ta nie jest tautologią. Jak poprzednio, zastępuje się kwantyfikator szczegółowy kwantyfikatorem ogólnym, otrzymując formułę

$$\neg\forall y \bullet \neg p(y) \Rightarrow \forall x \bullet p(x) \tag{16}$$

Zgodnie z poprzednim rozumowaniem, poszukuje się takiej interpretacji I i takiego wartościowania, dla których formuły $\forall x \bullet p(x)$ oraz $\forall y \bullet \neg p(y)$ są fałszywe. Jeżeli w interpretacji I formuła $\forall x \bullet p(x)$ nie jest spełnialna, to formuła $p(x)$ nie jest spełnialna. Podobnie, z tego, że formuła $\forall y \bullet \neg p(y)$ nie jest spełnialna, wynika, że formuła $\neg p(y)$ nie jest spełnialna, czyli formuła $p(y)$ jest spełnialna. Oczekuje się zatem, że dla pewnej interpretacji I i dla pewnego wartościowania zmiennych x, y formuła $p(x)$ jest fałszywa, a formuła $p(y)$ jest prawdziwa. Oczywiście taka interpretacja istnieje, wystarczy na przykład przyjąć, że dziedziną interpretacji jest zbiór liczb całkowitych, a interpretacja predykatu p jest następująca

$$I(p)(x) =_{\text{def}} x > 0$$

wówczas dla wartościowania $\nu =_{\text{def}} \{<x, -1>, <y, 1>\}$ $INT_\nu(p(x)) = \mathbf{F}$ oraz $INT_\nu(p(y)) = \mathbf{P}$. Formuła (16) nie jest zatem tautologią.

Graficzna reprezentacja tego dowodu jest pokazana na rysunku 11.4.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\to \neg\forall y \bullet \neg p(y) \Rightarrow \forall x \bullet p(x)}{\neg\forall y \bullet \neg p(y) \to \forall x \bullet p(x)}}{\to \forall x \bullet p(x), \forall y \bullet \neg p(y)}}{\to p(x), \forall y \bullet \neg p(y)}}{\to p(x), \neg p(y)}}{p(y) \to p(x)}$$

Rys. 11.4. Drzewo dowodowe dla formuły (15)

11.4. Język sekwentów – składnia i semantyka

Niech $FORM(F, P, V)$ będzie zbiorem formuł rachunku kwantyfikatorów nad alfabetem o sygnaturze $Sig = (F, P)$. Język rachunku kwantyfikatorów rozszerza się o dodatkową kategorię napisów nazywanych sekwentami.

Sekwentem nad alfabetem o sygnaturze Sig jest dowolny napis postaci

$$\Phi \to \Gamma \tag{1}$$

gdzie: Φ oraz Γ są dowolnymi skończonymi, być może pustymi, zbiorami formuł, tzn. $\Phi, \Gamma \subset FORM(F, P, V)$. Zbiór Φ jest nazywany *poprzednikiem*, a Γ – *następnikiem* sekwentu.

Zbiór wszystkich napisów postaci (1) będzie nazywany zbiorem sekwentów nad alfabetem o sygnaturze Sig i będzie oznaczany symbolem $SEKW(F, P, V)$.

Przyjmie się notację: jeżeli dane są zbiory formuł:

$\Phi =_{def} \{\alpha_1, ..., \alpha_n\}$ dla $n \in Nat$
$\Gamma =_{def} \{\beta_1, ..., \beta_m\}$ dla $m \in Nat$

to sekwent (1) będzie zapisywany w postaci

$$\alpha_1, ..., \alpha_n \rightarrow \beta_1, ..., \beta_m$$

Nawiasy będą więc opuszczane, a elementy zbiorów Φ oraz Γ będą przedstawiane w postaci list. Elementy obu zbiorów można na liście przedstawiać w dowolnej kolejności, a dwukrotne wystąpienie tego samego elementu na liście można pomijać.

Jeżeli Φ, Γ są zbiorami formuł, a α jest pojedynczą formułą, to zbiór formuł

$$\Phi \cup \{\alpha\} \cup \Gamma$$

będzie zapisywany w postaci listy

$$\Phi, \alpha, \Gamma$$

Szczególne przypadki sekwentów zachodzą wtedy, gdy zbiory Φ oraz Γ są puste:

$\rightarrow \beta_1, ..., \beta_m$	gdy Φ jest pusty,
$\alpha_1, ..., \alpha_n \rightarrow$	gdy Γ jest pusty,
$\rightarrow$	gdy Φ oraz Γ są puste.

Pierwszy sekwent ma pusty poprzednik, drugi – pusty następnik, a ostatni zapis oznacza sekwent pusty.

Semantykę sekwentów dla modelu $M = <D, I>$ definiuje funkcja

$$INT_v : SEKW(F, P, V) \rightarrow Logiczne$$

określana jako rozszerzenie funkcji interpretacji formuł

$$INT_v : FORM(F, P, V) \rightarrow Logiczne$$

nad tym modelem. W celu zdefiniowania tej funkcji wprowadza się najpierw oznaczenia pomocnicze: jeżeli

$$\Phi = \{\alpha_1, ..., \alpha_n\}$$

to

$\wedge\Phi =_{def} \alpha_1 \wedge ... \wedge \alpha_n$
$\vee\Phi =_{def} \alpha_1 \vee ... \vee \alpha_n$

Symbole $\wedge\Phi$ oraz $\vee\Phi$ są więc uogólnionymi spójnikami iloczynu i sumy logicznej, obejmującymi wszystkie formuły zbioru Φ. Jeżeli Φ jest zbiorem pustym, to z definicji:

$\wedge\Phi =_{def}$ **true**
$\vee\Phi =_{def}$ **false**

gdzie: **true** oraz **false** są stałymi logicznymi, interpretowanymi standardowo jako wartości **P** (*prawda*) i **F** (*fałsz*).

Funkcja interpretacji sekwentu $\Phi \rightarrow \Gamma$ przy wartościowaniu v zmiennych indywiduowych jest określona następująco:

$INT_v(\Phi \rightarrow \Gamma) = \mathbf{P}$ wtedy i tylko wtedy, gdy $INT_v(\wedge\Phi \Rightarrow \vee\Gamma) = \mathbf{P}$

gdzie symbol implikacji $\Rightarrow$ ma standardową interpretację.

Sekwent $\Phi \rightarrow \Gamma$ *jest spełniony w modelu* M, gdy funkcja interpretacji $INT_v(\Phi \rightarrow \Gamma)$ przyjmuje wartość prawdy dla dowolnego wartościowania v.

Sekwent $\Phi \rightarrow \Gamma$ jest *spełnialny uniwersalnie*, gdy jest spełnialny w dowolnym modelu.

Z podanych określeń wynika, że sekwent

$$\Phi \rightarrow \Gamma$$

jest semantycznie równoważny formule

$$\wedge\Phi \Rightarrow \vee\Gamma$$

Symbol $\rightarrow$ można więc traktować jako pewnego rodzaju uogólnienie spójnika implikacji $\Rightarrow$. Podczas gdy argumentami implikacji są dwie formuły, argumentami symbolu $\rightarrow$ są dwa zbiory formuł. W szczególnym przypadku, gdy zbiory te są jednoelementowe, znaczenie sekwentu jest takie same jak znaczenie implikacji.

Przykład 11.6

Sekwent

$$(\alpha \Rightarrow \beta) \rightarrow (\neg\beta \Rightarrow \neg\alpha)$$

jest równoważny semantycznie formule

$$(\alpha \Rightarrow \beta) \Rightarrow (\neg\beta \Rightarrow \neg\alpha)$$

11.5. System dowodzenia

System dowodzenia G w rachunku sekwentów Gentzena składa się z jednego aksjomatu oraz z reguł eliminacji spójników logicznych i kwantyfikatorów. Każdy ze spójników i kwantyfikatorów może wystąpić po lewej i prawej stronie sekwentu, dlatego liczba reguł jest równa dwukrotnej liczbie używanych spójników. Każda z reguł jest oznaczana literami l albo p – które oznaczają lewą albo prawą stronę sekwentu, po której występuje eliminowany spójnik – oraz symbolem spójnika. Na przykład $(l\forall)$ będzie oznaczać regułę eliminacji kwantyfikatora ogólnego stojącego po lewej stronie sekwentu. Przedstawiany dalej zestaw reguł dotyczy tylko spójników negacji, ko-

niunkcji i dysjunkcji, ponieważ stanowią one zbiór funkcjonalnie pełny, oraz tylko kwantyfikatora ogólnego, ponieważ za jego pomocą można również wyrazić kwantyfikator szczegółowy.

Aksjomat

Aksjomatem – a dokładniej schematem aksjomatu – jest dowolny sekwent:

$\Phi \rightarrow \Gamma$

dla którego $\Phi \cap \Gamma \neq \emptyset$, czyli gdy istnieje co najmniej jedna taka formuła, która występuje po lewej i po prawej stronie sekwentu.

Uwaga

Schemat aksjomatu jest tylko jeden, generuje on natomiast nieskończenie wiele aksjomatów, czyli konkretnych sekwentów. Podobna uwaga odnosi się do reguł wnioskowania. Należy zauważyć, że szczególnymi postaciami aksjomatu są sekwenty:

false $\rightarrow$

$\rightarrow$ **true**

Reguły

Dla negacji:

$$(l\neg)\ \frac{\neg\alpha, \Phi \rightarrow \Gamma}{\Phi \rightarrow \alpha, \Gamma}$$

$$(p\neg)\ \frac{\Phi \rightarrow \neg\alpha, \Gamma}{\Phi, \alpha \rightarrow \Gamma}$$

Dla koniunkcji:

$$(l\wedge)\ \frac{\alpha \wedge \beta, \Phi \rightarrow \Gamma}{\alpha, \beta, \Phi \rightarrow \alpha, \Gamma}$$

$$(p\wedge)\ \frac{\Phi \rightarrow \alpha \wedge \beta, \Gamma}{\Phi \rightarrow \Gamma, \alpha \quad \Phi \rightarrow \Gamma, \beta}$$

Dla alternatywy:

$$(l\vee)\ \frac{\alpha \vee \beta, \Phi \rightarrow \Gamma}{\Phi, \alpha \rightarrow \Gamma \quad \Phi, \beta \rightarrow \Gamma}$$

$$(p\vee)\ \frac{\Phi \rightarrow \alpha \vee \beta, \Gamma}{\Phi \rightarrow \alpha, \beta, \Gamma}$$

Dla kwantyfikatora ogólnego:

$$(l\,\forall)\quad \frac{\forall x \bullet \alpha, \Phi \rightarrow \Gamma}{\alpha[x ::= t], \forall x \bullet \alpha, \Phi \rightarrow \Gamma}$$

gdzie t jest dowolnym termem, który w formule α jest wolny ze względu na x

$$(p\forall)\quad \frac{\Phi \rightarrow \Gamma, \forall x \bullet \alpha}{\Phi \rightarrow \Gamma, \alpha} \quad \text{pod warunkiem, że } x \notin FV(\Phi) \cup FV(\Gamma)$$

Reguły eliminacji kwantyfikatora mają specyficzne własności.

Po pierwsze – reguła eliminacji kwantyfikatora ogólnego ($l\,\forall$) w istocie nie eliminuje kwantyfikatora, lecz dodatkowo wprowadza nową formułę. Występująca w przesłance formuła $\forall x \bullet \alpha$ zostaje zastąpiona formułami $\alpha[x ::= t]$ oraz $\forall x \bullet \alpha$ we wniosku reguły. Ponieważ zachodzi implikacja

$$(\forall x \bullet \alpha) \Rightarrow \alpha[x ::= t]$$

formułę $\alpha[x ::= t]$ można uważać za szczególny przypadek formuły $\forall x \bullet \alpha$.

Po drugie – nasuwa się pytanie, jak wyznaczyć term t. Każdorazowe wykorzystanie reguły ($l\,\forall$) powinno być związane z wprowadzaniem nowego termu, gdyż nie ma sensu tworzenie tych samych kopii formuły $\alpha[x ::= t]$ przy ustalonym t. Zbiór termów jest oczywiście nieskończony – zakłada się, że jest to zbiór $\{t_0, t_1, ..., t_n, ...\}$, dlatego podczas tworzenia drzewa dowodu jest potrzebny pewien pomocniczy mechanizm, który przy kolejnym k-tym użyciu reguły ($l\,\forall$), związanym z eliminacją kwantyfikatora przy formule $\forall x \bullet \alpha$, będzie wyznaczać term t_k.

Z uwagi na to, że reguła ($l\,\forall$) nie eliminuje kwantyfikatora, zaleca się, aby tę regułę stosować w ostatniej kolejności, po wyczerpaniu możliwości stosowania pozostałych reguł. Zalecenie takie może wyrażać zmodyfikowana postać reguły ($l\,\forall$)

$$(l\,\forall)'\quad \frac{\forall x \bullet \alpha, \Phi \rightarrow \Gamma}{\alpha[x ::= t], \Phi \rightarrow \Gamma, \neg\forall x \bullet \alpha}$$

Formuła $\forall x \bullet \alpha$ znalazła się po prawej stronie sekwentu ze znakiem negacji, co – przy tekstowym porządku eliminacji spójników – oznacza, że ponowne zastosowanie reguły eliminacji kwantyfikatora w odniesieniu do tej formuły zostanie odłożone na koniec, to znaczy po wyczerpaniu możliwości stosowania innych reguł eliminacji.

Dowodzenie, że formuła α jest tautologią, polega na budowie drzewa dowodu, którego wierzchołki są etykietowane sekwentami. Korzeniem drzewa, od którego rozpoczyna się budowę drzewa, jest sekwent postaci

$$\rightarrow \alpha$$

Następne kroki dowodu polegają na rozwijaniu drzewa przez wyznaczanie kolejnych wierzchołków-następników. Jeżeli wcześniej został wyznaczony pewien wierzchołek

etykietowany pewnym sekwentem S, to jego następnikami będą wierzchołki etykietowane sekwentami S_1, ..., S_k wtedy i tylko wtedy, gdy została zastosowana reguła pozwalająca sekwent S rozłożyć na sekwenty S_1, ..., S_k. Sekwent S oddziela się od jego następników kreską poziomą.

Budowę drzewa prowadzi się tak długo, aż osiągnie się przynajmniej jeden liść, który nie daje się dalej rozwijać i nie jest etykietowany aksjomatem – co oznacza, że badana formuła nie jest tautologią, albo – gdy wszystkie liście drzewa są etykietowane aksjomatami – co oznacza, że badana formuła jest tautologią.

Przedstawiony system dowodzenia bezpośrednio nie wyznacza algorytmu budowy drzewa dowodu. Jest to spowodowane tym, że system zawiera dwa źródła niedeterminizmu. Pierwszym źródłem jest brak ustalenia kolejności stosowania reguł tego systemu, a drugim źródłem – brak określenia postaci termów podstawianych za zmienne w wyniku stosowania reguł eliminacji kwantyfikatorów. Dokonanie ustaleń w tym zakresie pozwala już na zalgorytmizowanie postępowania dowodowego i ma wpływ na efektywność procesu dowodowego.

W niżej przedstawionym algorytmie wnioskowania przyjęto, że:

- kolejność stosowania reguł jest wyznaczona przez porządek tekstowy sekwentu – jako pierwszą wybiera się regułę, która się odnosi do pierwszego od lewej strony tekstu dającego wyeliminować się spójnika. Zamiast reguły ($l\ \forall$) stosuje się ponadto regułę ($l\ \forall$)′, co oznacza, że ponowne stosowanie reguły eliminacji kwantyfikatora ogólnego po lewej stronie sekwentu odbywa się po wyczerpaniu możliwości eliminacji innych spójników;
- zbiór wykorzystywanych termów jest uporządkowany w dowolny, ale ustalony ciąg t_0, t_1, ... Intuicja podpowiada oczywiście, aby termy uporządkować w kolejności od najprostszych po coraz bardziej złożone. Miarą złożoności może być na przykład długość termu, mierzona liczbą występujących w nim symboli języka.

Algorytm zakłada ponadto, że dowodzone formuły zawierają tylko wybrane spójniki logiczne (negację i koniunkcję) oraz kwantyfikator ogólny. Założenie to nie narusza ogólności, gdyż każda formuła daje się sprowadzić do równoważnej semantycznie formuły zawierającej wyłącznie te spójniki i jeden rodzaj kwantyfikatora. Algorytm jest przedstawiony w konwencji pseudoprogramowej. Jedyną konstrukcją, która może wymagać wyjaśnienia, jest użyta instrukcja pętli postaci

while *warunek* **do** *ciąg instrukcji* **od**

Konstrukcja ta oznacza następujący ciąg czynności: obliczenie wartości logicznej *warunku*, a następnie – jeżeli warunek jest prawdziwy – wykonanie *ciągu instrukcji*, po czym ponownie oblicza się *warunek* i – gdy jest prawdziwy – powtarza się obliczenie *ciągu instrukcji*; jeżeli po obliczaniu *warunku* okaże się, że jest on fałszywy, pętla się kończy.

Algorytm badania tautologii

Dane: formuła α.

Wynik: odpowiedź *tak*, jeżeli formuła α jest tautologią rachunku kwantyfikatorów, oraz *nie* w przypadku przeciwnym.

Procedura:

1. W formule α wyeliminuj spójniki logiczne: $\vee$, $\Rightarrow$, $\Leftrightarrow$ oraz kwantyfikator szczegółowy $\exists$, zastępując je tekstowo, zgodnie z poniższymi regułami:

Formuła zastępowana	Formuła zastępująca
$\alpha \vee \beta$	$\neg(\neg\alpha \wedge \neg\beta)$
$\alpha \Rightarrow \beta$	$\neg\alpha \vee \beta$
$\alpha \Leftrightarrow \beta$	$(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
$\exists x \bullet \alpha$	$\neg\forall x \bullet \neg\alpha$

2. Niech β będzie przekształconą formułą α. W formule β ponumeruj występujące w niej kwantyfikatory, powiedzmy od 1 do k. Wprowadź zmienne pomocnicze $i_1, \ldots, i_k$ i nadaj im wartości początkowe 0. Zmienna i_j odnosi się do kwantyfikatora o numerze j, a wartość tej zmiennej będzie oznaczać liczbę zastosowań reguły $(l\ \forall)'$ do j-tego kwantyfikatora.

3. Niech D będzie początkowym drzewem dowodu o jednym wierzchołku, etykietowanym sekwentem $\rightarrow \beta$.

4. **while**

   do liści drzewa D niebędących aksjomatami daje się zastosować reguły eliminacji spójników logicznych

   do

   modyfikuj D, stosując do jego liści regułę eliminacji, usuwającą pierwszy tekstowo (licząc od lewej do prawej strony) dający się wyeliminować spójnik logiczny;

   w przypadku zastosowania reguły $(l\ \forall)'$ w celu eliminacji kwantyfikatora o numerze j bierz pod uwagę term o numerze i_j ze zbioru wszystkich termów $\{t_0, t_1, \ldots, t_n, \ldots\}$, a następnie zwiększ wartość zmiennej i_j o jeden.

   od

5. Jeżeli wszystkie liście drzewa są aksjomatami, odpowiedz *tak*, w przypadku przeciwnym odpowiedz *nie*.

Krok 1. algorytmu ma znaczenie przygotowawcze – sprowadza daną formułę α do standardowej postaci β, która jest równoważna semantycznie formule α. Kroki 2. i 3. ustalają warunki początkowe dla zasadniczej części algorytmu, którą jest krok 4.

W tym kroku iteracyjnie powtarza się eliminację spójników logicznych i kwantyfikatora ogólnego. W iteracji tej może nastąpić zapętlenie tylko wtedy, gdy nieskończenie wiele razy stosuje się regułę $(l \forall)'$.

Algorytm ma następujące własności:

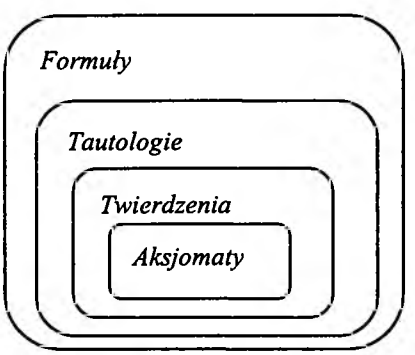
- Algorytm daje odpowiedź *tak* wtedy i tylko wtedy, gdy formuła dana na wejściu jest tautologią.
- Gdy formuła dana na wejściu nie jest tautologią, algorytm daje odpowiedź *nie* lub się zapętla.

Pierwsza własność oznacza poprawność i zupełność semantyczną przedstawionej metody dowodzenia. Druga własność oznacza częściową rozstrzygalność metody.

11.6. Semantyczna poprawność

Semantyczna poprawność systemu dowodzenia G dla rachunku sekwentów Gentzena oznacza, że zachodzi implikacja: jeżeli $\Phi \vdash_G \alpha$, to $\Phi \models \alpha$.

Graficzną ilustracją poprawności systemu dowodzenia jest rysunek 11.5: zbiór twierdzeń ma być podzbiorem zbioru tautologii.



Rys. 11.5. Ilustracja związku pomiędzy zbiorami twierdzeń i tautologii dla poprawnego systemu dowodzenia

Poprawność systemu G zostanie wykazana przez udowodnienie kilku lematów.

Lemat 11.3

Aksjomat rachunku sekwentów jest spełnialny uniwersalnie.

Dowód

Każdy aksjomat jest sekwentem postaci $\Phi, \alpha \rightarrow \Gamma, \alpha$. Zakłada się, że sekwent ten nie jest spełnialny uniwersalnie. Niech M będzie modelem, dla którego sekwent nie jest spełniony. Oznacza to, że jednocześnie $M \models \alpha$ oraz że $M \not\models \alpha$, co z kolei oznacza, że M nie jest modelem. ■

Lemat 11.4

Niech $y \notin FV(\alpha) \setminus \{x\}$ oraz niech y będzie zmienną wolną w formule α ze względu na x, wówczas

$$INT_{v[x := d]} \models \alpha \text{ wtedy i tylko wtedy, gdy } INT_{v[y := d]} \models \alpha[x ::= y]$$

Dowód – ćwiczenie.

Lemat 11.5

Dla każdej reguły dowodzenia jej wniosek jest sekwentem spełnialnym uniwersalnie wtedy i tylko wtedy, gdy uniwersalnie jest spełnialny każdy sekwent stanowiący jej przesłankę.

Dowód

Dowód prowadzi się metodą nie wprost. Dla każdej reguły pokaże się, że pewna interpretacja INT_v nie spełnia jej wniosku wtedy i tylko wtedy, gdy dla pewnej interpretacji $INT_{v'}$ nie są spełnione jej przesłanki. Teza twierdzenia wynika bezpośrednio z tego stwierdzenia.

Rozważania przeprowadza się tylko dla reguł $(l \wedge)$, $(p \wedge)$, $(l \forall)$, $(p \forall)$. Dowód dla pozostałych reguł prowadzi się podobnie.

Dla reguły $(l \wedge)$ zachodzi

$$INT_v \not\models \Phi, \alpha, \beta \rightarrow \Gamma$$

wtedy i tylko wtedy, gdy

$$INT_v \models \Phi \wedge \alpha \wedge \beta \rightarrow \vee\Gamma$$

wtedy i tylko wtedy, gdy

$$INT_v \not\models \Phi, \alpha \wedge \beta \rightarrow \vee\Gamma$$

W interpretacji INT_v przesłanki są zatem fałszywe wtedy i tylko wtedy, gdy fałszywy jest wniosek reguły $(l \wedge)$.

Dla reguły $(p \wedge)$ zachodzi

$$INT_v \not\models \Phi \rightarrow \Gamma, \alpha \wedge \beta$$

wtedy i tylko wtedy, gdy

$$INT_v \models \wedge\Phi \text{ oraz } INT_v \not\models \Gamma \text{ oraz } INT_v \not\models \alpha \wedge \beta$$

wtedy i tylko wtedy, gdy

$$INT_v \models \Phi \text{ oraz } INT_v \not\models \vee\Gamma \text{ oraz } (INT_v \not\models \alpha \text{ lub } INT_v \not\models \beta)$$

wtedy i tylko wtedy, gdy

$$INT_v \not\models \Phi \rightarrow \Gamma, \alpha \qquad \text{lub } INT_v \not\models \Phi \rightarrow \Gamma, \beta$$

W interpretacji INT_v przesłanki są zatem fałszywe wtedy i tylko wtedy, gdy fałszywy jest wniosek reguły ($p \wedge$).

Dla reguły ($l \forall$) zachodzi:

Jeżeli założyć, że dla INT_v nie jest spełniona przesłanka reguły, czyli

$$INT_v \not\models \Phi, \forall x \bullet \alpha \rightarrow \Gamma$$

to oznacza, że

$$INT_v \models \wedge\Phi \wedge \forall x \bullet \alpha \text{ oraz } INT_v \not\models \Gamma$$

Ponieważ $INT_v \models \forall x \bullet \alpha$, więc – na mocy wniosku z podrozdziału 11.2 – zachodzi $INT_v \models \alpha[x ::= t]$, pod warunkiem, że t jest termem wolnym w α ze względu na x. Z tego wynika, że dla INT_v nie jest spełniony sekwent stanowiący wniosek, czyli

$$INT_v \not\models \Phi, \alpha[x ::= t], \forall x \bullet \alpha \rightarrow \Gamma$$

Odwrotnie, jeżeli założyć, że dla INT_v nie jest spełniony wniosek reguły, to wynika z tego, że $INT_v \models \wedge\Phi \wedge \alpha[x ::= t] \wedge \forall x \bullet \alpha$ oraz $INT_v \not\models \Gamma$. Z tego z kolei wynika, że $INT_v \models \wedge\Phi \wedge \forall x \bullet \alpha$ oraz $INT_v \models \vee \Gamma$, co oznacza, że dla INT_v nie jest spełniona przesłanka reguły.

Dla reguły ($p \forall$) zachodzi:

Jeżeli założyć, że dla INT_v nie jest spełniona przesłanka reguły, czyli

$$INT_v \not\models \Phi \rightarrow \Gamma, \forall x \bullet \alpha$$

to oznacza, że

$$INT_v \models \wedge\Phi \text{ oraz } INT_v \not\models \vee \Gamma \text{ oraz } INT_v \not\models \forall x \bullet \alpha$$

Z ostatniego faktu wynika, że istnieje $d \in \boldsymbol{D}$ takie, że $INT_{v[x := d]} \not\models \alpha$. Niech $INT'_v \models INT_{v[y := d]}$, gdzie $y \notin FV(\Phi) \cup FV(\Gamma) \cup FV(\alpha)$ jest zmienną różną od x i wolną w α ze względu na x. Na mocy lematu 11.5 $INT'_v \not\models \alpha[x ::= y]$. Ponadto $INT'_v \models \wedge\Phi$ oraz $INT'_v \not\models \vee\Gamma$, ponieważ $y \notin FV(\Phi) \cup FV(\Gamma)$. Dlatego $INT'_v \not\models \Phi \rightarrow \Gamma, \forall x \bullet \alpha, \alpha[x ::= y]$. Oznacza to, że jeżeli przesłanka reguły jest niespełnialna przez pewną interpretację, to wniosek reguły jest też niespełnialny przez pewną interpretację.

Odwrotnie, jeżeli założy się, że dla INT_v nie jest spełniony wniosek reguły, to $INT_v \models \wedge\Phi$ oraz $INT_v \not\models \vee\Gamma$, $INT_v \not\models \alpha[x ::= y]$. Z lematu o podstawieniu wynika,

że $INT_{v[x:=d]} \not\models \alpha$, gdzie $d = INT_v(y)$. Dla INT_v nie jest więc spełniona formuła $\forall x \bullet \alpha$, co oznacza, że nie jest również spełniona przesłanka reguły. ■

Twierdzenie 11.1

Każdy sekwent, który ma dowód w systemie Gentzena, jest sekwentem uniwersalnie prawdziwym.

Dowód

Dowód wynika z wyżej udowodnionych lematów przez zastosowanie indukcji na strukturze drzewa dowodowego. Punktem wyjścia jest fakt, że sekwenty-liście jako aksjomaty są uniwersalnie prawdziwe, a każde przejście od sekwentów-wniosków do sekwentu-przesłanki w drzewie dowodu gwarantuje zachowanie uniwersalnej prawdziwości przesłanki. ■

Wniosek 11.2

Dla każdej interpretacji INT_v, jeżeli interpretacja ta nie spełnia sekwentu etykietującego n-ty wierzchołek drzewa dowodu, to nie spełnia ona również żadnego sekwentu etykietującego wierzchołek leżący na ścieżce prowadzącej od korzenia do wierzchołka n.

Lemat 11.5, oprócz tego, że pozwala dowieść poprawności semantycznej systemu dowodzenia, wskazuje na jeszcze jedną jego własność. Pokazuje mianowicie, że reguły prowadzą od prawdziwych wniosków do prawdziwych założeń, to znaczy: jeżeli reguła pozwala z sekwentu S otrzymać dwa sekwenty S_1, S_2 – jak na przykład w regule dla koniunkcji – to prawdziwość S_1 oraz S_2 implikuje prawdziwość sekwentu S. Oznacza to odwracalność wprowadzonych reguł. Reguły otrzymane na podstawie takiego odwrócenia są regułami wprowadzania spójników logicznych. Każdej regule eliminacji spójnika logicznego odpowiada więc reguła wprowadzania tego spójnika. Na przykład regułom eliminacji spójnika koniunkcji będą odpowiadać reguły wprowadzania spójnika koniunkcji odpowiednio po lewej ($l^+ \wedge$) i po prawej stronie sekwentu ($p^+ \wedge$):

$$(l^+ \wedge) \quad \frac{\alpha, \beta, \Phi \to \Gamma}{\alpha \wedge \beta, \Phi \to \Gamma}$$

$$(p^+ \wedge) \quad \frac{\Phi \to \Gamma, \alpha \quad \Phi \to \Gamma, \beta}{\Phi \to \Gamma, \alpha \wedge \beta}$$

Zestaw reguł wprowadzania spójników logicznych i kwantyfikatora ogólnego pozwala na konstrukcję takich samych drzew dowodu jak dla systemu z regułami eliminacji, z tą różnicą, że konstrukcja drzewa odbywa się w odwrotnej kolejności – od liści do korzenia.

11.7. Semantyczna zupełność

Semantyczna zupełność systemu dowodzenia G dla rachunku sekwentów Gentzena oznacza, że zachodzi implikacja: jeżeli $\Phi \models \alpha$, to $\Phi \vdash_G \alpha$. Odwołując się do rysunku 11.5, oznacza to, że zbiór twierdzeń i zbiór tautologii są identyczne.

Zupełność systemu G zostanie wykazana przez udowodnienie kilku lematów.

Lemat 11.6

Jeżeli formuła α jest tautologią oraz drzewo dowodowe, uzyskane w wyniku stosowania podanego algorytmu, jest skończone, to wszystkie jego liście są etykietowane aksjomatami.

Dowód

Dowód prowadzi się metodą nie wprost. Zakłada się, że D jest pewnym drzewem skończonym zbudowanym dla formuły α takim, że pewien liść jest etykietowany sekwentem postaci $\Phi \rightarrow \Gamma$, gdzie $\Phi \cap \Gamma = \emptyset$. Jeżeli da się pokazać, że istnieje pewna interpretacja INT_v, dla której sekwent ten nie jest spełniony, to z wniosku 11.2 wynika, że nie jest również spełniony sekwent stanowiący korzeń drzewa dowodu. Oznacza to zatem – wbrew założeniu lematu – że formuła α nie jest tautologią.

Poszukiwaną interpretację skonstruuje się w sposób następujący: Jako dziedzinę $\boldsymbol{D}$ interpretacji przyjmuje się zbiór wszystkich termów $TERM(F, P)$. Interpretacja $\boldsymbol{I}$ symboli funkcyjnych i symboli predykatów jest następująca:

- wynikiem zastosowana funkcji $f \in F_n$, dla $n \in Nat$, do termów $t_1, ..., t_n$ jest term $f(t_1, ..., t_n)$,
- dla dowolnych termów $t_1, ..., t_n$ oraz dowolnego predykatu $p \in P_n$, dla $n \in Nat$, definiuje się $p(t_1, ..., t_n) = prawda$ wtedy i tylko wtedy, gdy formuła $p(t_1, ..., t_n) \in \Phi$, czyli gdy występuje ona po lewej stronie sekwentu $\Phi \rightarrow \Gamma$.

Rozpatruje się takie wartościowanie v, które dowolnej zmiennej indywiduowej x przypisuje term x, czyli $v(x) =_{def} x$. Z definicji $\boldsymbol{I}$ oraz v wynika, że

$$INT_v \not\models \wedge\Phi \Rightarrow \vee\Gamma$$

bowiem wszystkie formuły w sekwencie $\Phi \rightarrow \Gamma$ są nierozkładalne, a na mocy definicji wszystkie formuły z Φ są prawdziwe, wszystkie zaś formuły z Γ są fałszywe. ■

Lemat 11.7

Jeżeli drzewo dowodowe uzyskane w wyniku stosowania podanego algorytmu dla formuły α jest nieskończone, to formuła α nie jest tautologią.

Dowód

Jeżeli drzewo jest nieskończone, to – na podstawie lematu Königa – oznacza to istnienie nieskończenie długiej ścieżki, zaczynającej się od korzenia. Niech $\Phi_i \rightarrow \Gamma_i$ będą sekwentami etykietującymi i-te wierzchołki na nieskończonej ścieżce. Dalej niech $\Phi = \bigcup_{i \in Nat} \Phi_i$ oraz $\Gamma = \bigcup_{i \in Nat} \Gamma_i$.

Wprowadza się teraz następującą interpretację: Dziedziną interpretacji – podobnie jak w poprzednim lemacie – jest zbiór wszystkich termów. Również interpretacja symboli funkcyjnych jest taka sama, jak w poprzednim lemacie, interpretacja zaś symboli predykatów jest następująca:

dla dowolnych termów $t_1, ..., t_n$ oraz dowolnego predykatu $p \in P_n$, z definicji $p(t_1, ..., t_n) = prawda$ wtedy i tylko wtedy, gdy formuła $p(t_1, ..., t_n) \in \Phi$, czyli gdy występuje ona po lewej stronie jednego z sekwentów $\Phi_i \rightarrow \Gamma_i$.

Niech v będzie wartościowaniem takim, że $v(x) =_{def} x$. Z pokazanego dalej lematu 11.8 wynika, że dla dowolnej formuły α:

jeżeli $\alpha \in \Phi$, to $INT_v \models \alpha$

jeżeli $\alpha \in \Gamma$, to $INT_v \models \neg\alpha$

Stwierdzenie tego faktu kończy dowód lematu, gdyż oznacza, że formuła α nie jest tautologią, bowiem korzeń drzewa dowodu jest etykietowany sekwentem $\rightarrow\alpha$, czyli formuła $\alpha \in \Gamma$. ■

Lemat 11.8

Niech $\Phi = \bigcup_{i \in Nat} \Phi_i$ oraz $\Gamma = \bigcup_{i \in Nat} \Gamma_i$, gdzie $\Phi_i \rightarrow \Gamma_i$ są sekwentami etykietującymi i-te wierzchołki na nieskończonej ścieżce w drzewie dowodu. Dla dowolnej formuły α:

jeżeli $\alpha \in \Phi$, to $INT_v \models \alpha$

jeżeli $\alpha \in \Gamma$, to $INT_v \models \neg\alpha$

Dowód

Zdefiniuje się najpierw relację $\prec$, określoną na zbiorze wszystkich formuł w sposób następujący:

1. Elementami minimalnymi relacji są formuły elementarne postaci $p(t_1, ..., t_n)$, gdzie: $p \in P_n$ jest symbolem n-argumentowego predykatu, a $t_1, ..., t_n$ są dowolnymi termami. Oznacza to, że dla formuły postaci $p(t_1, ..., t_n)$ nie istnieje formuła α taka, że $\alpha \prec p(t_1, ..., t_n)$.
2. $\alpha \prec \neg\alpha$

3. $\alpha \prec (\alpha \wedge \beta)$ oraz $\beta \prec (\alpha \wedge \beta)$

4. Dla dowolnego termu t zachodzi $\alpha[x := t] \prec \forall x \bullet \alpha$

Niech $\prec^+$ będzie przechodnim domknięciem relacji $\prec$. Jeśli α jest dowolną formułą, to nie istnieje nieskończony ciąg formuł $\alpha_1, \alpha_2, \ldots$ spełniający warunek

$$\ldots \prec^+ \alpha_2 \prec^+ \alpha_1 \prec^+ \alpha$$

Wynika to z obserwacji, że po lewej stronie relacji znajduje się zawsze formuła składniowo prostsza od formuły po prawej stronie relacji. Dla danej formuły liczba formuł składniowo od niej prostszych jest oczywiście skończona.

Dowód lematu będzie prowadzony indukcyjnie względem relacji $\prec^+$.

W kroku bazowym pokazuje się, że teza lematu zachodzi dla elementów minimalnych względem relacji $\prec^+$. Zakłada się, że $p(t_1, \ldots, t_n) \in \Phi \cup \Gamma$. Istnieje zatem $k \in Nat$ takie, że $p(t_1, \ldots, t_n) \in \Phi_k \cup \Gamma_k$ oraz k jest pierwszym takim wierzchołkiem na rozpatrywanej ścieżce. Oczywiście $\Phi_k \cap \Gamma_k = \varnothing$, gdyż w przeciwnym razie wierzchołek byłby aksjomatem i ścieżka kończyłaby się w tym wierzchołku, wbrew założeniu o jej nieskończoności, zatem albo $p(t_1, \ldots, t_n) \in \Phi_k$, albo $p(t_1, \ldots, t_n) \in \Gamma_k$.

Niech $p(t_1, \ldots, t_n) \in \Phi_k$. Ponieważ do formuł postaci $p(t_1, \ldots, t_n)$ nie da się już zastosować żadnej z reguł rozkładu, więc $p(t_1, \ldots, t_n) \in \Phi_m$ dla $m \geq k$, a tym samym $p(t_1, \ldots, t_n) \notin \Gamma_i$ dla dowolnego $i \in Nat$, zatem $p(t_1, \ldots, t_n) \in \Phi$ oraz $p(t_1, \ldots, t_n) \notin \Gamma$. Na mocy definicji interpretacji INT_v zachodzi więc $INT_v \models p(t_1, \ldots, t_n)$.

Analogicznie dla drugiego przypadku, gdy $p(t_1, \ldots, t_n) \in \Gamma_k$, można pokazać, że

$$INT_v \models \neg p(t_1, \ldots, t_n).$$

W kroku indukcyjnym dowodu rozpatruje się kolejne przypadki postaci formuły α.

1. Niech α będzie postaci $\neg\beta$. Na mocy reguł eliminacji spójnika negacji zachodzi:

   jeżeli $\neg\beta \in \Phi$, to $\beta \in \Gamma$ oraz

   jeżeli $\neg\beta \in \Gamma$, to $\beta \in \Phi$.

   Wystarczy teraz skorzystać z założenia indukcyjnego. Na przykład, gdy $\neg\beta \in \Phi$, to $\beta \in \Gamma$. Ponieważ $\beta \prec^+ \neg\beta$, zatem – korzystając z założenia indukcyjnego z $\beta \in \Gamma$ – wynika, że $INT_v \models \neg\beta$.

2. Niech α będzie postaci $\beta \wedge \gamma$. Z postaci reguł eliminacji koniunkcji wynika, że formuły β oraz γ występują razem w Φ albo jedna z nich występuje w Γ. Oczywiście $\beta \prec^+ (\beta \wedge \gamma)$ oraz $\gamma \prec^+ (\beta \wedge \gamma)$. Z założenia indukcyjnego wynika, że

$INT_v \models \beta$ oraz $INT_v \models \gamma$

albo

$INT_v \models \neg\beta$ lub $INT_v \models \neg\gamma$

zatem

$INT_v \models (\beta \wedge \gamma)$ gdy $(\beta \wedge \gamma) \in \Phi$

albo

$INT_v \models \neg(\beta \wedge \gamma)$ gdy $(\beta \wedge \gamma) \in \Gamma$

3. Niech α będzie postaci $\forall x \bullet \beta$. Dla przypadku zastosowania reguły eliminacji kwantyfikatora ($l\ \forall$) zakłada się nie wprost, że nie zachodzi $INT_v \models \forall x \bullet \beta$. Oznacza to, że dla pewnego elementu z dziedziny interpretacji, czyli dla pewnego termu t, formuła β nie jest spełniona, to znaczy, że $INT_v \models \neg\beta[x ::= t]$.

   Ponieważ $\beta[x := t] \prec^+ \forall x \bullet \beta$, oznacza to więc, że $\beta[x ::= t]$ musi wystąpić po lewej stronie sekwentu po skończonej liczbie zastosowań reguły ($l\ \forall$), czyli $\beta[x ::= t] \in \Phi$. Z założenia indukcyjnego wynika, że $INT_v \models \beta[x ::= t]$, co oznacza sprzeczność z wyprowadzonym wnioskiem, że $INT_v \models \neg\beta[x ::= t]$.

Przypadek zastosowania drugiej reguły eliminacji kwantyfikatora ($p\ \forall$) rozważa się podobnie. ■

Twierdzenie 11.2

System dowodzenia Gentzena jest semantycznie zupełny.

Dowód

Niech formuła α będzie tautologią. Wystarczy pokazać, że drzewo dowodowe dla tej formuły jest skończone. Z lematu 11.6 wynika bowiem, że wszystkie jego liście są etykietowane aksjomatami, co wskazuje na to, że α jest tautologią. Drzewo dowodowe dla α musi być jednak skończone, gdyż w przeciwnym przypadku – zgodnie z lematem 11.7 – formuła α nie byłaby tautologią. ■

Z przedstawionych rozważań wypływa dodatkowy wniosek:

Wniosek 11.3

System dowodzenia Gentzena jest częściowo rozstrzygalny.

Oznacza to, że istnieje procedura (na przykład algorytm przedstawiony w podrozdziale 11.5), która w skończonej liczbie kroków stwierdza, że badana formuła α jest tautologią albo stwierdza, że formuła nie jest tautologią, albo – w przypadku, gdy α nie jest tautologią – nie udziela żadnej odpowiedzi w skończonej liczbie kroków.

Ćwiczenia

1. Stosując metodę sekwentów Gentzena, wykazać, że następujące formuły nie są tautologiami rachunku kwantyfikatorów:
   a) $(\exists x \bullet p(x)) \Rightarrow p(x)$
   b) $(\forall x \bullet p(x)) \Rightarrow (\forall x \bullet \neg p(x))$
2. Czy formuły z zadania 1. są spełnialne?
3. Stosując metodę sekwentów Gentzena, wykazać, że następujące formuły są tautologiami rachunku kwantyfikatorów:
   a) $(\exists x \bullet p(x) \vee q(x)) \Leftrightarrow (\exists x \bullet p(x)) \vee (\exists x \bullet q(x))$
   b) $(\forall x \bullet p(x) \Leftrightarrow q(x)) \Rightarrow ((\forall x \bullet p(x)) \Leftrightarrow (\forall x \bullet q(x)))$
   c) $(\forall x \bullet p(x) \Leftrightarrow q(x)) \Rightarrow ((\exists x \bullet p(x)) \Leftrightarrow (\exists x \bullet q(x)))$
4. Stosując metodę sekwentów Gentzena, sprawdzić, które spośród następujących formuł są tautologiami rachunku kwantyfikatorów:
   a) $(\neg\forall x \bullet \forall y \bullet r(x, y)) \Leftrightarrow (\exists x \bullet \exists y \bullet \neg r(x, y))$
   b) $(\neg\exists x \bullet \exists y \bullet r(x, y)) \Leftrightarrow (\forall x \bullet \forall y \bullet \neg r(x, y))$
5. Podać dolne i górne oszacowanie liczby wierzchołków drzewa dowodu formuły rachunku zdań metodą sekwentów Gentzena, przy założeniu, że liczba wystąpień spójników logicznych w formule wynosi $n \in Nat$.
6. Przedstawić algorytm, który dla danej formuły wyznacza pierwszy (od lewej) tekstowo spójnik, który może być wyeliminowany przy dowodzeniu metodą sekwentów Gentzena.
7. Przedstawić algorytm, który sprawdza, czy term t jest wolny w α ze względu na zmienną indywiduową x.

12. Metoda rezolucji

12.1. Wstęp

Omówiony w rozdziale 11. system dowodzenia – oparty na rachunku sekwentów Gentzena – daje podstawę do tworzenia algorytmów automatycznego dowodzenia formuł rachunku kwantyfikatorów, ale jest związany z pewnymi niedogodnościami, które prowadzą do dużej złożoności obliczeniowej, a tym samym czasochłonności obliczeń. Poszukiwanie innych, bardziej efektywnych metod dowodzenia doprowadziło, w 1965 roku, do sformułowania przez J.A. Robinsona systemu dowodzenia opartego na tzw. *regule rezolucji*. Oparta na tej regule metoda, zwana metodą rezolucji, stanowi system dowodzenia spełnialności zbioru formuł. Istotną właściwością dowodzenia opartego na regule rezolucji jest konieczność sprowadzenia dowodzonej formuły do postaci znormalizowanej. Formuły muszą być w skolemowskiej postaci normalnej, a ich matryce – w koniunkcyjnej postaci normalnej.

Uwaga

Podejście zaproponowane przez Robinsona okazało się bardzo skuteczne i dało podwaliny pod wiele zastosowań, wśród których na pierwszym miejscu należy wymienić programowanie w logice. Programowanie w logice, zwłaszcza w języku *Prolog*, umożliwiło między innymi efektywną implementację systemów doradczych (ekspertowych), sterowanie robotami, automatyczne tłumaczenie tekstów.

Metoda Robinsona bazuje na wcześniejszych pracach, wśród których należy wymienić prace Herbranda[21] z 1930 roku oraz pracę Davida i Putnamana z początku lat sześćdziesiątych ubiegłego wieku.

Stosowanie metody rezolucji wiąże się z badaniem, czy dana formuła α jest logiczną (semantyczną) konsekwencją zbioru formuł Φ, czyli czy $\Phi \models \alpha$. Zgodnie z lematem 10.3, $\Phi \models \alpha$ wtedy i tylko wtedy, gdy zbiór $\Phi \cup \{\neg\alpha\}$ jest niespełnialny. Jeżeli $\Phi \models \{\alpha_1, ..., \alpha_n\}$, to niespełnialność zbioru formuł $\Phi \cup \{\neg\alpha\}$ oznacza, że formuła

$$\alpha_1 \wedge ... \wedge \alpha_n \wedge \neg\alpha$$

[21] Jaques Herbrand (1908–1931).

jest tożsamościowo fałszywa. Niespełnialność zbioru $\Phi \cup \{\neg\alpha\}$ oznacza zatem, że jedyną jego konsekwencją semantyczną jest formuła tożsamościowo fałszywa, czyli

$\Phi \cup \{\neg\alpha\} \models$ **false**

Stosowanie metody rezolucji polega na reprezentacji zbioru formuł $\Phi \cup \{\neg\alpha\}$ za pomocą zbioru klauzul, a następnie na próbie wyprowadzenia z tego zbioru klauzuli pustej, reprezentującej **false**. Jeżeli taka próba kończy się powodzeniem, to znaczy wyprowadzeniem klauzuli pustej, oznacza to, że $\Phi \models \alpha$. W przypadku przeciwnym, po wyczerpaniu wszystkich możliwości, oznacza to, że $\Phi \not\models \alpha$.

12.2. Zasada rezolucji dla rachunku zdań

Zakłada się, że badane formuły rachunku zdań są w koniunkcyjnej postaci normalnej, co oznacza, że są w postaci koniunkcji klauzul:

$\kappa_1 \wedge \kappa_2 \wedge \ldots \wedge \kappa_n$ dla $n \in Nat \setminus \{0\}$

Klauzula jest dysjunkcją literałów:

$\lambda_1 \vee \lambda_2 \vee \ldots \vee \lambda_m$ dla $m \in Nat$

W przypadku szczególnym, gdy $m = 0$, klauzula będzie nazywana klauzulą *pustą* i oznaczana symbolem □. Klauzula pusta oznacza formułę tożsamościowo fałszywą.

W przypadku rachunku zdań literałami są zmienne zdaniowe lub ich negacje. Dwa literały są komplementarne, gdy jeden jest negacją drugiego.

W przypadku znormalizowanej reprezentacji formuły można mówić, że formuła jest określona przez pewien zbiór klauzul, klauzula zaś jest określona przez pewien zbiór literałów. Uwaga ta wyjaśnia wprowadzaną poniżej konwencję oznaczeń.

Fakt, że pewien literał λ jest elementem klauzuli κ, będzie zapisywany w postaci $\lambda \in \kappa$. Jeżeli z klauzuli κ zostanie usunięty należący do niej literał λ, to otrzymana nowa, być może pusta, klauzula będzie oznaczana $\kappa \setminus \lambda$.

Definicja 12.1

Niech będą dane dwie klauzule κ_1, κ_2 oraz dwa literały komplementarne λ_1, λ_2 takie, że $\lambda_1 \in \kappa_1$, $\lambda_2 \in \kappa_2$. Klauzula postaci

$\kappa_1 \setminus \lambda_1 \cup \kappa_2 \setminus \lambda_2$

będzie nazywana *rezolwentą* klauzul κ_1 oraz κ_2 i oznaczana symbolicznie przez $rez(\kappa_1, \kappa_2)$. Literały λ_1, λ_2 nazywa się *literałami czynnymi*. O dwóch klauzulach mających rezolwentę będzie się mówić, że są klauzulami, które dają się *uzgodnić*. Oznaczenie $rez(\kappa_1, \kappa_2)$ ma sens tylko pod warunkiem, że klauzule κ_1, κ_2 dają się uzgodnić.

Badanie, czy dana formuła jest tautologią, polega na sprawdzeniu zbioru reprezentujących ją klauzul. Formuła α jest spełnialna wtedy i tylko wtedy, gdy jest spełnialny zbiór reprezentujących ją klauzul $\kappa_1, \kappa_2, ..., \kappa_n$. Zbiór klauzul nie jest spełnialny wtedy i tylko wtedy, gdy jego semantyczną konsekwencją jest klauzula pusta.

Dowodzenie polega na generowaniu, na podstawie zbioru klauzul $\kappa_1, \kappa_2, ..., \kappa_n$, nowych klauzul tak długo, aż zostanie utworzona klauzula pusta bądź, po wyczerpaniu wszystkich możliwości, klauzula pusta nie zostanie wygenerowana. Wygenerowanie klauzuli pustej będzie oznaczać, że badany zbiór klauzul jest niespełnialny, a przypadek przeciwny będzie oznaczać spełnialność badanego zbioru klauzul.

Generacja nowej klauzuli opiera się na zastosowaniu reguły rezolucji w postaci:

Definicja 12.2

Schemat reguły rezolucji ma postać

$$\frac{\kappa_1, \kappa_2}{\kappa_1 \setminus \lambda_1 \cup \kappa_2 \setminus \lambda_2} \quad \lambda_1 \in \kappa_1, \lambda_2 \in \kappa_2 \text{ oraz } \lambda_1, \lambda_2 \text{ są komplementarne}$$

Przesłankami reguły rezolucji są klauzule, a wnioskiem – rezolwenta tych klauzul.

Reguła rezolucji jest ogólną regułą wnioskowania, której szczególne postaci odzwierciedlają niektóre inne znane reguły wnioskowania.

Przykład 12.1

Niech będą dane dwie klauzule: p oraz $\neg p \vee q$. Spełniają one wymagania oczekiwane od przesłanek w regule rezolucji, literałami czynnymi są p oraz $\neg p$, a ich rezolwentą jest klauzula q, czyli

$$\frac{p, \neg p \vee q}{q}$$

Warto zauważyć, że klauzula $\neg p \vee q$ jest semantycznie równoważna formule $p \Rightarrow q$, co pozwala zapisać powyższą regułę w postaci

$$\frac{p, p \Rightarrow q}{q}$$

znanej jako reguła odrywania (*modus ponens*).

Przykład 12.2

Dla klauzul $\neg p \vee q$ oraz $\neg q \vee r$ reguła rezolucji ma postać

$$\frac{\neg p \vee q, \neg q \vee r}{\neg p \vee r}$$

co odpowiada regule wnioskowania łańcuchowego

$$\frac{p \Rightarrow q, q \Rightarrow r}{p \Rightarrow r}$$

Przykład 12.3

Prosta, ale szczególnie ważna, jest reguła rezolucji w postaci

$$\frac{\neg p, p}{\square}$$

Oznacza ona, że klauzula pusta wyraża sprzeczność przesłanek.

Należy zwrócić uwagę, że rezolwenta dwóch klauzul nie zawsze jest wyznaczona jednoznacznie, gdyż klauzule mogą zawierać więcej niż jedną parę literałów komplementarnych. W powstałej rezolwencie mogą się ponadto powtarzać pewne literały. Powtarzające się literały można usunąć bez naruszenia semantyki klauzuli. Proces eliminacji powtarzających się literałów określa się jako *faktoryzację*.

Przykład 12.4

Dla klauzul:

$\neg p \vee \neg q \vee \neg r \vee \neg s$ oraz $p \vee \neg q \vee r \vee \neg s \vee \neg t \vee \neg u$

rezolwentami są:

$\neg q \vee \neg r \vee \neg s \vee \neg q \vee r \vee \neg s \vee \neg t \vee \neg u$ (literałami aktywnymi są $\neg p$ i p)

oraz

$\neg p \vee \neg q \vee \neg s \vee p \vee \neg q \vee \neg s \vee \neg t \vee \neg u$ (literałami aktywnymi są $\neg r$ i r)

Po faktoryzacji rezolwenty przyjmują odpowiednio postać:

$\neg q \vee \neg r \vee \neg s \vee r \vee \neg t \vee \neg u$

oraz

$\neg p \vee \neg q \vee \neg s \vee p \vee \neg t \vee \neg u$

Reguła rezolucji jest semantycznie poprawna, co precyzuje twierdzenie.

Twierdzenie 12.1

Rezolwenta $rez(\kappa_1, \kappa_2)$ jest semantyczną konsekwencją klauzul κ_1, κ_2, czyli

$\kappa_1, \kappa_2 \square rez(\kappa_1, \kappa_2)$.

Dowód

Niech klauzule κ_1, κ_2 mają postać:

$\kappa_1 \equiv \lambda \vee \lambda_1 \vee ... \vee \lambda_n$

$\kappa_2 \equiv \neg\lambda \vee \lambda'_1 \vee ... \vee \lambda'_m$

gdzie λ, $\neg\lambda$ są literałami komplementarnymi. Niech klauzule te będą spełnione w pewnej interpretacji I. Oznacza to, że spełniony jest również jeden z literałów λ, $\neg\lambda$. Niech będzie to $\neg\lambda$. Wobec tego nie jest spełniony literał λ, ale skoro spełniona jest klauzula κ_1, to musi być spełniona formuła $\lambda_1 \vee ... \vee \lambda_n$. Formuła ta jest składnikiem rezolwenty $rez(\kappa_1,\kappa_2)$, a zatem rezolwenta jest również spełniona w interpretacji I. Rozumowanie przebiega podobnie, gdy założyć, że spełniony jest literał λ. ■

Na podstawie reguły rezolucji można sformułować algorytm badania spełnialności formuły rachunku zdań.

Algorytm badania spełnialności zbioru klauzul

Dane: formuła α rachunku zdań.

Wynik: odpowiedź *tak*, gdy formuła jest spełnialna, *nie* – w przypadku przeciwnym.

Procedura:

1. Dla formuły α wyznaczyć $CNF(\alpha)$ – algorytm z podrozdziału 9.6.
2. Wyznaczyć zbiór klauzul S reprezentujących $CNF(\alpha)$.
3. Powtarzać następujące czynności:

   while

   $\square \notin S$ i istnieją klauzule κ_1, $\kappa_2 \in S$ dające rezolwentę nienależącą do S

   do

   (a) znajdź klauzule κ_1, κ_2, które dają się uzgodnić i wylicz ich rezolwentę $rez(\kappa_1, \kappa_2)$,

   (b) zastąp S przez S', gdzie $S' = S \cup rez(\kappa_1, \kappa_2)$

   od

4. Jeżeli S zawiera klauzulę pustą, to odpowiedz *nie*, w przypadku przeciwnym – odpowiedz *tak*.

Przykład 12.5

Niech będzie dany następujący zbiór klauzul

$S =_{def} \{a \vee \neg b \vee \neg c, d, b, c \vee \neg a \vee \neg b, c \vee \neg d, \neg a \vee \neg d \vee \neg b\}$

Dla przejrzystości rozważań poszczególne klauzule zostaną zapisane w ponumerowanych wierszach. W następnych wierszach są zapisane klauzule uzyskane ze

zbioru S w wyniku stosowania reguły rezolucji. Po prawej stronie klauzuli są podane numery klauzul, które były przesłankami do jej uzyskania:

(1) $a \vee \neg b \vee \neg c$
(2) d
(3) b
(4) $c \vee \neg a \vee \neg b$
(5) $c \vee \neg d$
(6) $\neg a \vee \neg d \vee \neg b$

(7) $a \vee \neg c$	(1, 3)
(8) c	(2, 5)
(9) $\neg a \vee \neg b$	(2, 6)
(10) $c \vee \neg a$	(3, 4)
(11) $\neg a \vee \neg d$	(3, 6)
(12) $\neg a$	(2, 11)
(13) $\neg c$	(7, 12)
(14) $\square$	(8, 13)

Ostatnia wygenerowana klauzula jest klauzulą pustą, co dowodzi, że badany zbiór klauzul S nie jest spełnialny.

12.3. Skolemowska postać normalna

Badając spełnialność formuł rachunku kwantyfikatorów według metody rezolucji, zakłada się, że formuły są w postaci kanonicznej, zwanej skolemowską postacią normalną.

Niech $\alpha \in FORM(F, P, V)$ będzie formułą rachunku kwantyfikatorów nad sygnaturą $<F, P>$ i zbiorem zmiennych indywiduowych V.

Definicja 12.3

Formuła znajduje się w *postaci Skolema*, gdy jest w przedrostkowej postaci normalnej, a jej przedrostek nie zawiera kwantyfikatorów egzystencjalnych.

Przez $Skol(\alpha)$ będzie oznaczany skolemowski odpowiednik formuły α. Proces wyznaczania odpowiednika skolemowskiego dla danej formuły nazywa się *skolemizacją*.

W odróżnieniu od przypadku, gdy dla dowolnej formuły α istniała równoważna semantycznie postać przedrostkowa $PNF(\alpha)$, między formułą α a odpowiadającą jej postacią skolemowską $Skol(\alpha)$ równoważność taka może nie zachodzić. Zachodzi natomiast słabszy rodzaj równoważności oparty na związku *spełnialności*. Oznacza to, że formuła α jest spełnialna wtedy i tylko wtedy, gdy spełnialna jest formuła $Skol(\alpha)$. Formuła α jest spełnialna, gdy istnieje model, w którym jest spełniona.

Istotę skolemizacji wyjaśnia przykład.

Przykład 12.6

Dana jest formuła postaci

$$\forall x \bullet \forall y \bullet ((x \prec y) \Rightarrow \exists z \bullet (x \prec z) \wedge (z \prec y)) \tag{1}$$

lub po sprowadzeniu do przedrostkowej postaci normalnej

$$\forall x \bullet \forall y \bullet \exists z \bullet (\neg(x \prec y) \vee (x \prec z) \wedge (z \prec y)) \tag{2}$$

Jest to aksjomat elementarnej teorii relacji mniejszości, zdefiniowanej w rozdziale 10. Modelem dla tej teorii jest na przykład zbiór liczb wymiernych *Wymierne*, jako dziedzina interpretacji, oraz relacja mniejszości < w zbiorze liczb wymiernych, jako interpretacja symbolu predykatu $\prec$. W modelu tym aksjomat mówi, że dla dwóch dowolnych liczb wymiernych x, y istnieje pewna pośrednia liczba wymierna z, która jest zawarta w przedziale między liczbą x a liczbą y. Liczbę pośrednią można zawsze wskazywać bezpośrednio, biorąc na przykład średnią arytmetyczną liczb x, y, czyli określając, że wartościowanie zmiennej z jest funkcją wartościowania zmiennych x oraz y, zdefiniowaną wzorem $z =_{\text{def}} (x + y)/2$. Dla średniej arytmetycznej prawdziwa jest bowiem formuła

$$\forall x \bullet \forall y \bullet (\neg(x \prec y) \vee (x \prec (x + y)/2) \wedge (x + y)/2 \prec y) \tag{3}$$

Formuła ta jest w postaci skolemowskiej i różni się od poprzedniej formuły brakiem kwantyfikatora szczegółowego, który został zastąpiony dwuargumentową funkcją obliczającą średnią arytmetyczną. Ostatnią formułę można zapisać w ogólniej postaci

$$\forall x \bullet \forall y \bullet (\neg(x \prec y) \vee (x \prec f(x, y) \wedge f(x, y) \prec y) \tag{4}$$

gdzie f jest pewnym symbolem funkcyjnym, który nie występował w teorii relacji mniejszości.

Tekstowy związek między formułą (4) oraz formułą (2) można scharakteryzować następująco: formuła (4) powstała z formuły (2) przez eliminację kwantyfikatora szczegółowego oraz przez tekstowe zastąpienie każdego wystąpienia zmiennej z, wiązanej wyeliminowanym kwantyfikatorem, przez term $f(x, y)$, gdzie: x oraz y są zmiennymi wiązanymi przez kwantyfikatory ogólne, poprzedzające wyeliminowany kwantyfikator szczegółowy, a f jest nowym dwuargumentowym symbolem funkcyjnym.

Na przykładzie widać, że aksjomat (2) teorii relacji mniejszości można byłoby zastąpić aksjomatem postaci (4). Nowa aksjomatyka jest spełniona w każdym modelu, w którym jest spełniona dawna aksjomatyka.

Algorytm skolemizacji

Dane: formuła α nad sygnaturą $<F, P>$.

Wynik: formuła $Skol(\alpha)$ w postaci skolemowskiej, równoważna w sensie *spełnialności* formule α.

Procedura: procedura postępowania polega na etapowym, tekstowym przekształcaniu formuły α. Formuła pośrednia jest oznaczana przez β.

1. Niech α będzie formułą w przedrostkowej postaci normalnej (otrzymaną na przykład w wyniku stosowania algorytmu przedstawionego w p. 10.8).
2. **while**

   w przedrostku formuły β istnieje kwantyfikator egzystencjalny

   do

   (a) Jeżeli formuła β ma postać: $\exists x \bullet \gamma$, to zastępuje się ją formułą postaci $\gamma[x ::= c]$, gdzie c jest nowym symbolem stałej, to znaczy że c nie należy do zbioru symboli stałych sygnatury $<F, P>$, czyli $c \notin F_0$.

   (b) Jeżeli formuła β ma postać: $\forall x_1 \bullet \ldots \bullet \forall x_m \bullet \exists y \bullet \gamma$, dla $m > 0$, to zastępuje się ją formułą postaci: $\forall x_1 \bullet \ldots \bullet \forall x_m \bullet \gamma[y ::= f(x_1, \ldots, x_m)]$, gdzie f jest nowym m-argumentowym symbolem funkcyjnym, to znaczy że f nie należy do zbioru m-argumentowych symboli funkcyjnych sygnatury $<F, P>$, czyli $f \notin F_m$.

   od
3. Otrzymaną formułę β definiuje się jako $Skol(\alpha)$.

Przykład 12.7

Dla formuły postaci

$$\exists u \bullet \forall w \bullet \exists x \bullet \forall y \bullet \exists z \bullet (p(u, x) \Rightarrow q(w, y, h(z)))$$

jej skolemowskim odpowiednikiem jest formuła

$$\forall w \bullet \forall y \bullet (p(c, f(w)) \Rightarrow q(w, y, h(g(w, y))))$$

gdzie c, f, g są nowymi symbolami funkcyjnymi.

Twierdzenie 12.2

Formuła α jest spełnialna wtedy i tylko wtedy, gdy spełnialna jest formuła $Skol(\alpha)$ otrzymana w wyniku podanego wyżej algorytmu.

Dowód

Jeżeli formuła $\alpha \equiv \forall x_1 \bullet \ldots \bullet \forall x_m \bullet \exists y \bullet \gamma$ jest spełnialna, to oznacza, że jest spełniona w pewnym modelu $\boldsymbol{M} = <\boldsymbol{D}, \boldsymbol{I}>$ nad sygnaturą Sig, czyli

$$INT_v(\forall x_1 \bullet \ldots \bullet \forall x_m \bullet \exists y \bullet \gamma) = prawda$$

dla dowolnego wartościowania v. Z tego wynika, że również

$$INT_v(\exists y \bullet \gamma) = prawda$$

dla $v' = v[x_1 := d_1, ..., x_m := d_m]$, gdzie $d_1, ..., d_m \in \boldsymbol{D}$ są dowolnymi wartościami z dziedziny interpretacji. Oznacza to, że istnieje taka wartość $d \in \boldsymbol{D}$, że

$$INT_{v'[y := d]}(\gamma) = prawda$$

Niech Sig' będzie rozszerzeniem sygnatury Sig o m-argumentowy symbol funkcyjny f_y oraz niech $\boldsymbol{I}'$ będzie takim rozszerzeniem interpretacji $\boldsymbol{I}$, że $\boldsymbol{I}'(f_y) = f^d$ jest funkcją, która – z definicji – dla $d_1, ..., d_m \in \boldsymbol{D}$ przyjmuje wartość $d \in \boldsymbol{D}$, czyli

$$f^d(d_1, ..., d_m) = d$$

wówczas

$$\begin{aligned} v'[y := INT_{v'}(f_y(x_1, ..., x_m))] &= v'[y := f^d(INT_{v'}(x_1), ..., INT_{v'}(x_m))] \\ &= v'[y := f^d(d_1, ..., d_m)] \\ &= v'[y := d] \end{aligned}$$

Formuła γ nie zawiera symbolu f_y, dlatego

$$INT_{v'[y := d]}(\gamma) = prawda$$

Z lematu o podstawieniu wynika, że

$$INT_{v'}(\gamma) = INT_v(\gamma[y ::= f_y(x_1, ..., x_m)])$$

Ponieważ $d_1, ..., d_m$ były wybrane dowolnie, zatem

$$INT_v(\forall x_1 \bullet ... \bullet \forall x_m \bullet \gamma[y ::= f_y(x_1, ..., x_m)]) = prawda$$

dla dowolnego v.

Wynikanie odwrotne – spełnialność formuły α ze spełnialności formuły $Skol(\alpha)$ jest oczywiste.

Przedstawienie formuły α w postaci skolemowskiej

$$\forall x_1 \bullet ... \bullet \forall x_n \bullet (\kappa_1 \wedge ... \wedge \kappa_m)$$

gdzie $\kappa_1, ..., \kappa_n$ są klauzulami, pozwala na reprezentację formuły w postaci zbioru jej klauzul.

Reprezentacja ta jest jednoznaczna przy założeniu, że wszystkie zmienne występujące w klauzulach są związane kwantyfikatorami ogólnymi. Założenie to zawsze można przyjąć, gdy bada się spełnialność formuły. Wynika to z faktu, że jeśli na przykład jest spełnialna formuła $\forall y \bullet \forall x \bullet p(x, y)$, to również spełnialna jest formuła $\forall x \bullet p(x, y)$. ■

12.4. Unifikacja termów

Stosowanie metody rezolucji w rachunku kwantyfikatorów wymaga dodatkowego procesu, polegającego na sprowadzeniu literałów do pewnej ujednoliconej postaci. Proces ten nazywa się *unifikacją termów*. Zbiór klauzul $\{p(x), \neg p(y)\}$ jest oczywiście zbiorem niespełnialnym, ale literały $p(x)$, $\neg p(y)$ nie są komplementarne. Ogólnie literałami w rachunku kwantyfikatorów są formuły elementarne lub ich negacje, na przykład $p(t_1, ..., t_m)$, $\neg p(t_1, ..., t_m)$, gdzie: p jest symbolem predykatywnym, a $t_1, ..., t_m$ są termami. Sprowadzanie takich literałów do ujednoliconej postaci – jeżeli jest możliwe – polega na zastosowaniu do nich jednakowych podstawień tekstowych (zob. podrozdział 10.4).

Rozważane będą tylko takie podstawienia $\sigma =_{def} [x_1 ::= t_1, ..., x_n ::= t_n]$, które spełniają warunek

$$Var(t_i) \cap \{x_1, ..., x_n\} = \varnothing \text{ dla } i = 1, ..., n,$$

to znaczy, że w termach $t_1, ..., t_n$ nie występują zmienne $x_1, ..., x_n$.

Podstawienie σ nazywa się *unifikatorem* dla formuł $\alpha_1, ..., \alpha_n$, gdy

$$\alpha_1 \sigma \equiv ... \equiv \alpha_n \sigma.$$

Formuły muszą oczywiście być oparte na tym samym symbolu predykatywnym, mogą się różnić co najwyżej postacią termów, które są ich argumentami. Formuły $\alpha_1 \sigma, ..., \alpha_n \sigma$ nazywa się *formułami ukonkretnionymi* przez podstawienie σ.

Przykład 12.8

Unifikatorem dla formuł $p(x)$ oraz $p(y)$ jest $[x ::= z, y ::= z]$, gdyż

$$p(x)[x ::= z, y ::= z] \equiv p(y)[x ::= z, y ::= z] \equiv p(z)$$

ale unifikatorami są również na przykład:

$[x ::= w, y ::= w]$, gdzie w jest zmienną,

$[x ::= 5, y ::= 5]$, gdzie 5 jest stałą,

$[x ::= g(z, w), y ::= g(z, w)]$, gdzie g jest symbolem funkcyjnym.

Dla formuł $p(x)$, $p(6)$ jest tylko jeden unifikator $[x ::= 6]$, a dla formuł $p(5)$, $p(6)$ nie ma unifikatora.

Formułę α ukonkretnioną przez podstawienie σ można ukonkretniać ponownie innym podstawieniem τ. Dwukrotne ukonkretnienie formuły, najpierw podstawieniem σ, a następnie podstawieniem τ, jest równoważne jednokrotnemu ukonkretnieniu podstawieniem $\sigma\tau$, które jest złożeniem podstawień σ oraz τ. Oznacza to, że

$$(\alpha \sigma) \tau \equiv \alpha(\sigma \tau).$$

Przykład 12.9

Formułę $p(x, y)$ można kolejno ukonkretnić podstawieniem $[x ::= f(z)]$, co da formułę $p(f(z), y)$, a następnie podstawieniem $[y ::= g(u, w)]$, co da $p(f(z), g(u, w))$.

Złożeniem podstawień $[x ::= f(z)]$ oraz $[y ::= g(u, w)]$ jest podstawienie

$[x ::= f(z), y ::= g(u, w)]$.

Zastosowanie tego podstawienia do formuły $p(x, y)$ daje również $p(f(z), g(u, w))$.

Jeżeli dla podstawienia σ istnieje podstawienie odwrotne σ^{-1} takie, że $\sigma\,\sigma^{-1} = \sigma^{-1}\sigma = \varepsilon$, gdzie ε jest podstawieniem tożsamościowym, to σ jest nazywane *przemianowaniem* zmiennych.

Przykład 12.10

Postawienie $[x ::= z, y ::= w]$ jest przemianowaniem zmiennych, gdyż

$[x ::= z, y ::= w]\,[z ::= x, w ::= y] = [x ::= x, y ::= y]$

Definicja 12.4

Podstawienie σ_1 jest *bardziej ogólne* niż podstawienie σ_2, jeżeli dla pewnego niepustego podstawienia τ, różnego od przemianowania, zachodzi $\sigma_2 = \sigma_1\,\tau$.

Definicja 12.5

Podstawienie σ nazywa się *najbardziej ogólnym unifikatorem* formuł $\alpha_1, \ldots, \alpha_n$, gdy jest unifikatorem i jest bardziej ogólne od każdego innego unifikatora tych formuł.

Z definicji wynika, że najbardziej ogólny unifikator jest określony z dokładnością do nazw zmiennych. Najbardziej ogólny unifikator formuł $\alpha_1, \ldots, \alpha_n$ będzie oznaczany przez $NOU(\alpha_1, \ldots, \alpha_n)$.

Przykład 12.11

Najbardziej ogólnymi unifikatorami dla następujących par formuł są:

$NOU(p(10, 20), p(20, 10))$ nie istnieje
$NOU(p(10, 20), p(10, 20)) = \varepsilon$
$NOU(p(10, x), p(y, 20)) = [x ::= 20, y ::= 10]$
$NOU(p(10, x), p(10, y)) = [x ::= y]$ (a także $[y ::= x]$)
$NOU(p(x, x), p(10, y)) = [x ::= 10,\ y ::= 10]$
$NOU(p(f(10), 20), p(x, 20)) = [x ::= f(10)]$
$NOU(p(f(10), 20), p(10, 20))$ nie istnieje

Najbardziej ogólny unifikator można wyznaczyć w sposób algorytmiczny. Istnieje algorytm, który dla dowolnego zbioru formuł $\alpha_1, ..., \alpha_n$ w skończonej liczbie kroków orzeka, czy zbiór ten jest unifikowalny, a w przypadku, gdy zbiór ten jest unifikowalny, wyznacza $NOU(\alpha_1, ..., \alpha_n)$. Algorytm polega na tekstowym porównywaniu formuł, wykrywaniu i usuwaniu niezgodności, przez określanie odpowiednich podstawień, aż do uzyskania pełnej zgodności bądź do wyczerpania możliwości podstawień.

Definicja 12.6

Niech t i q będą termami. *Parą niezgodną* nazywa się takie podtermy t' i q' termów t i q, że:

- pierwsze symbole t' i q' są różne
- do miejsca wystąpienia podtermów t' i q' (licząc od lewej do prawej strony) termy t i q są identyczne.

Zbiorem niezgodności dla formuł $p_1(t_1, ..., t_n)$ i $p(q_1, ..., q_i)$ jest zbiór złożony z pary podtermów niezgodnych dla termów t_i, q_i dla najmniejszego $i \in \{1, ..., n\}$. Zbiorem niezgodności dla zbioru formuł opartych na tym samym symbolu predykatywnym jest zbiór niezgodności dla dowolnej pary formuł z tego zbioru.

Przykład 12.12

Dla podanych niżej par formuł zbiory niezgodności są następujące:

Zbiór formuł	Zbiór niezgodności
$\{p(x), p(y)\}$	$\{x, y\}$
$\{q(f(x), 20), q(10, 20)\}$	$\{f(x), 10\}$
$\{r(x, f(x, y), z), r(y, z, g(x, y))\}$	$\{x, y\}$
$\{r(x, f(x, y), z), r(x, z, g(x, y))\}$	$\{f(x, y\}, z\}$

Algorytm wyznaczania najbardziej ogólnego unifikatora

Dane: zbiór formuł $\{\alpha_1, ..., \alpha_n\}$, $n > 1$.

Wynik: $NOU(\alpha_1, ..., \alpha_n)$, gdy najbardziej ogólny unifikator istnieje, oraz odpowiedź *brak unifikatora* w przypadku przeciwnym.

Procedura: algorytm polega na cyklicznym wyliczaniu unifikatora w kolejnych iteracjach numerowanych przez zmienną k.

1. Wartości początkowe zmiennych algorytmu: $k = 0$, $\Phi_0 = \{\alpha_1, ..., \alpha_n\}$, $\sigma_0 = \varepsilon$.
2. Jeżeli $card(\Phi_k) = 1$, to algorytm się kończy i $NOU(\Phi_0) = \sigma_k$, w przypadku przeciwnym wylicz zbiór niezgodności N_k dla Φ_k i przejdź do następnego kroku.
3. Jeżeli w zbiorze niezgodności N_k występują zmienna x_k oraz term t_k takie, że x_k nie występuje w t_k, to przejdź do następnego kroku, w przypadku przeciwnym zbiór Φ_0 nie jest unifikowany i algorytm się kończy.

4. Oblicz nowe podstawienie $\sigma_{k+1} = \sigma_k\,[x_k ::= t_k]$, dokonaj unifikacji zbioru formuł Φ_k unifikatorem $[x_k ::= t_k]$, to znaczy $\Phi_{k+1} = \Phi_k\,[x_k ::= t_k]$, zwiększ k o jeden i przejdź do kroku 2.

Przykład 12.13

Niech

$$\Phi = \{p(10, x, f(g(y))), p(z, f(z), f(u))\}.$$

Obliczenia algorytmu unifikacji:

$\sigma_0 = \varepsilon$, $\Phi_0 = \Phi$, $N_0 = \{10, z\}$, $x_0 ::= z$, $t_0 = 10$

$\sigma_1 = \sigma_0\,[z ::= 10] = \varepsilon\,[z ::= 10] = [z ::= 10]$

$\Phi_1 = \Phi_0\,[z ::= 10] = \{p(10, x, f(g(y))), p(z, f(z), f(u))\}\,[z ::= 10] =$
$= \{p(10, x, f(g(y))), p(10, f(10), f(u))\}$

$N_1 = \{x, f(10)\}$, $x_1 ::= x$, $t_1 = f(10)$

$\sigma_2 = \sigma_1\,[x ::= f(10)] = [z ::= 10]\,[x ::= f(10)] = [z ::= 10, x ::= f(10)]$

$\Phi_2 = \Phi_1\,[x ::= f(10)] = \{p(10, x, f(g(y))), p(10, f(10), f(u))\}\,[x ::= f(10)] =$
$= \{p(10, f(10), f(g(y))), p(10, f(10), f(u))\}$

$N_2 = \{g(y), u)\}$, $x_2 ::= u$, $t_2 = g(y)$

$\sigma_3 = \sigma_2\,[u ::= g(y)] = [z ::= 10, x ::= f(10), u ::= g(y)]$

$\Phi_3 = \Phi_2\,[u ::= g(y)] = \{p(10, f(10), f(g(y))), p(10, f(10), f(u))\}\,[u ::= g(y)] =$
$= \{p(10, f(10), f(g(y))), p(10, f(10), f(g(y)))\} =$
$= \{p(10, f(10), f(g(y)))\}$

Zbiór Φ_3 jest singletonem, zatem $NOU(\Phi) = \sigma_3 = [z ::= 10, x ::= f(10), u ::= g(y)]$.

Przykład 12.14

Niech

$$\Phi = \{q(f(10), g(x))), q(y, y)\}.$$

Obliczenia algorytmu:

$\sigma_0 = \varepsilon$, $\Phi_0 = \Phi$, $N_0 = \{f(10), x\}$, $x_0 ::= y$, $t_0 = f(10)$

$\sigma_1 = \sigma_0\,[y ::= f(10)] = [y ::= f(10)]$

$\Phi_1 = \Phi_0\,[x ::= f(10)] = \{q(f(10), g(x))), q(x, x)\}\,[x ::= f(10)] =$
$= \{q(f(10), g(x))), q(f(10), f(10))\}$

$N_1 = \{g(x), f(10)\}$

W zbiorze niezgodności N_1 nie ma zmiennej, zbiór Φ nie jest zatem unifikowalny.

Twierdzenie 12.3

Przedstawiony algorytm zawsze kończy się po skończonej liczbie iteracji. Jeżeli zatrzyma się w kroku 2., to ostatnio obliczone podstawienie σ_k jest najbardziej ogólnym unifikatorem zbioru formuł Φ. Jeżeli zatrzyma się w kroku 3., to zbiór formuł Φ nie ma najbardziej ogólnego unifikatora.

Dowód jest zawarty na przykład w książce [Szałas 1991].

12.5. Zasada rezolucji dla rachunku kwantyfikatorów

Zasada rezolucji dla rachunku kwantyfikatorów zakłada, że formuły są w postaci skolemowskiej. Pozwala to na stwierdzenie, że – jak w przypadku rachunku zdań – formuła jest reprezentowana przez zbiór klauzul.

Literały $p(t_1, ..., t_n)$ oraz $\neg p(t'_1, ..., t'_n)$ dają się uzgodnić, gdy istnieje najbardziej ogólny unifikator $NOU\{p(t_1, ..., t_n), p(t'_1, ..., t'_n) = \sigma$. Możliwość uzgodnienia literałów oznacza, że formuły

$$p(t_1, ..., t_n)\sigma \text{ oraz } \neg p(t'_1, ..., t'_n)\,\sigma$$

są literałami komplementarnymi. σ będzie nazywane najbardziej ogólnym unifikatorem skojarzonym z literałami $p(t_1, ..., t_n)$ oraz $\neg p(t'_1, ..., t'_n)$.

Definicja 12.7

Schemat reguły rezolucji ma postać

$$\frac{\kappa_1, \kappa_2}{(\kappa_1 \setminus \lambda_1 \cup \kappa_2 \setminus \lambda_2)\,\sigma}$$

gdzie: $\lambda_1 \in \kappa_1$, $\lambda_2 \in \kappa_2$ są dającymi się uzgodnić literałami, a σ jest najbardziej ogólnym, skojarzonym z nimi, unifikatorem.

Klauzulę $(\kappa_1 \setminus \lambda_1 \cup \kappa_2 \setminus \lambda_2)\,\sigma$ nazywa się rezolwentą klauzul κ_1, κ_2 i oznacza symbolicznie $rez(\kappa_1, \kappa_2)$.

Dalej przyjmuje się, że rozpatrywane klauzule są sfaktoryzowane. W przypadku rachunku zdań oznacza to, że nie ma w nich powtarzających się literałów. W przypadku rachunku kwantyfikatorów sytuacja jest bardziej złożona. Na przykład – jak łatwo zauważyć – zbiór klauzul $\{p(x) \vee p(u), \neg p(y) \vee \neg p(v)\}$ jest niespełnialny, ale nie można tego wykazać za pomocą reguły rezolucji, dlatego wprowadza się pojęcie faktora klauzuli.

Definicja 12.8

Jeżeli σ jest najbardziej ogólnym unifikatorem pewnego podzbioru literałów klauzuli κ, to klauzulę κ', uzyskaną z κ przez zastosowanie do niej σ i usunięcie powtarzających się literałów, nazywa się *faktorem* klauzuli κ. Klauzula jest *sfaktoryzowana*, jeśli dowolny podzbiór jej literałów nie ma wspólnego unifikatora.

Przykład 12.15

Faktorem klauzuli $p(z, y) \vee p(x, g(x))$ jest $p(x, g(x))$, gdyż

$$NOU(p(z, y), p(x, g(x))) = [z ::= x, y ::= g(x)].$$

Zbiór klauzul $\{p(x) \vee p(u), \neg p(y) \vee \neg p(v)\}$, po faktoryzacji, przekształca się w zbiór $\{p(x), \neg p(y)\}$.

Algorytm badania spełnialności zbioru klauzul

Dane: formuła α rachunku kwantyfikatorów.

Wynik: odpowiedź *tak*, gdy formuła jest spełnialna, *nie* – w przypadku przeciwnym.

Procedura:

1. Dla formuły α wyznaczyć *Skol*(α) – algorytm z podrozdziału 12.3.
2. Wyznaczyć zbiór klauzul S reprezentujących *Skol*(α) i dokonać ich faktoryzacji.
3. Powtarzać następujące czynności:

   while

   $\square \notin S$ i istnieją klauzule κ_1, $\kappa_2 \in S$ dające rezolwentę nienależącą do S

   do

   (a) znajdź klauzule κ_1, κ_2, które dają się uzgodnić, znajdź dla nich najbardziej ogólny unifikator σ i wylicz ich rezolwentę $rez(\kappa_1, \kappa_2)$,

   (b) zastąp S przez S', gdzie $S' = S \cup rez(\kappa_1, \kappa_2)$

   od

4. Jeżeli S zawiera klauzulę pustą odpowiedz *nie*, w przypadku przeciwnym – odpowiedz *tak*.

Przykład 12.16

Niech będzie dany zbiór S klauzul:

$$\{p(x, g(x)),\ \neg p(u, v) \vee q(10),\ \neg p(w, g(10)) \vee \neg\, q(w)\}$$

Obliczenia algorytmu są przedstawione w podobnej konwencji jak dla rachunku zdań – poszczególne klauzule są zapisane w ponumerowanych wierszach. W następnych wierszach są zapisane klauzule uzyskane ze zbioru S w wyniku sto-

sowania reguły rezolucji. Po prawej stronie klauzuli są podane numery klauzul, które były przesłankami do jej uzyskania oraz zastosowane unifikatory:

(1) $p(x, g(x))$

(2) $\neg p(u, v) \vee q(10)$

(3) $\neg p(w, g(10)) \vee \neg q(w)$

(4) $\neg p(u, v) \vee \neg p(10, g(10))$	(2), (3)	$[w ::= 10]$
(5) $\neg p(10, g(10))$	(4)	faktoryzacja
(6) $\square$	(1), (5)	$[x ::= 10]$

Wyprowadzenie klauzuli pustej oznacza, że S jest niespełnialnym zbiorem klauzul.

Reguła rezolucji wyznacza specyficzny system dowodzenia R. Specyfika polega na tym, że system R nie ma aksjomatów i ma tylko jedną regułę wnioskowania – regułę rezolucji. System R jest systemem semantycznie poprawnym i zupełnym. Dokładnie precyzują to następujące twierdzenia:

Twierdzenie 12.4

Jeżeli istnieje wywód rezolucyjny klauzuli κ ze zbioru klauzul $\{\kappa_1, ..., \kappa_n\}$, to klauzula κ jest semantyczną konsekwencją zbioru $\{\kappa_1, ..., \kappa_n\}$, symbolicznie:

jeśli $\{\kappa_1, ..., \kappa_n\} \vdash_R \kappa$, to $\{\kappa_1, ..., \kappa_n\} \models \kappa$

Dowód

Prosty dowód twierdzenia sprowadza się do pokazania, że pojedynczy krok wnioskowania rezolucyjnego – wyliczenie rezolwenty dla klauzul-przesłanek – wyznacza klauzulę, która jest konsekwencją semantyczną klauzul-przesłanek (twierdzenie 12.1). ■

Twierdzenie 12.5

Jeżeli zbiór klauzul $\{\kappa_1, ..., \kappa_n\}$ jest niespełnialny, to istnieje wywód rezolucyjny klauzuli pustej ze zbioru $\{\kappa_1, ..., \kappa_n\}$, symbolicznie:

jeśli $\{\kappa_1, ..., \kappa_n\} \models$ **false**, to $\{\kappa_1, ..., \kappa_n\} \vdash_R \square$

Dowód twierdzenia, tu pominięty, jest zawarty na przykład w książce [Szałas 1991].

Należy wskazać na ograniczoność użytego tu pojęcia zupełności semantycznej w stosunku do pojęcia używanego w podrozdziale 11.7. Reguła rezolucji nie pozwala bowiem na wyprowadzenie wszystkich klauzul, które są semantyczną konse-

kwencją danego zbioru klauzul, pozwala natomiast na stwierdzanie niespełnialności dowolnego zbioru klauzul. Ograniczoność używanego pojęcia zupełności jest rekompensowana większą efektywnością obliczeniową algorytmu badania spełnialności zbioru klauzul w stosunku do algorytmu badania tautologii opartego na rachunku sekwentów Gentzena.

12.6. Klauzule Horna w programowaniu logicznym

Zasada rezolucji ma szczególne zastosowanie wtedy, gdy formuły są przedstawione w postaci zbioru klauzul Horna.

Definicja 12.9

Klauzulę nazywa się *klauzulą Horna*, gdy zawiera co najwyżej jeden literał pozytywny.

W dalszej części rozdziału pozytywne literały będą oznaczane symbolami: λ_1, λ_2, ..., a literały negatywne będą jawnie poprzedzane symbolem negacji: $\neg\lambda_1$, $\neg\lambda_2$, ... Klauzula Horna ma zatem postać

$$\lambda \vee \neg\lambda_1 \vee \ldots \vee \neg\lambda_n$$

gdzie: λ jest literałem opcjonalnym oraz $n \in Nat$.

Klauzule Horna są podstawą programowania w logice. Program logiczny jest zbiorem klauzul Horna $\{\kappa_1, \ldots, \kappa_n\}$ mających literał pozytywny. Obliczenie programu polega na udzielaniu odpowiedzi na pytanie: czy dana formuła w postaci koniunkcji literałów $\lambda_1 \wedge \ldots \wedge \lambda_n$ jest konsekwencją semantyczną klauzul stanowiących treść programu, czyli czy $\{\kappa_1, \ldots, \kappa_n\} \models \lambda_1 \wedge \ldots \wedge \lambda_n$?

Udzielenie odpowiedzi na zadane pytanie sprowadza się do zbadania spełnialności zbioru formuł $\{\kappa_1, \ldots, \kappa_n\} \cup \{\neg(\lambda_1 \wedge \ldots \wedge \lambda_n)\}$, czyli zbioru klauzul

$$\{\kappa_1, \ldots, \kappa_n\} \cup \{\neg\lambda_1 \vee \ldots \vee \neg\lambda_n\}.$$

W programowaniu w logice dla klauzul Horna używa się specyficznej notacji. Wynika ona z następujących równoważności semantycznych:

$$\lambda \vee \neg\lambda_1 \vee \ldots \vee \neg\lambda_n = \lambda \vee \neg(\lambda_1 \wedge \ldots \wedge \lambda_n) = (\lambda_1 \wedge \ldots \wedge \lambda_n) \Rightarrow \lambda$$

Ostatnią implikację zapisuje się w postaci odwróconej

$$\lambda \Leftarrow \lambda_1, \ldots, \lambda_n$$

z zastąpieniem przecinkami symboli koniunkcji. Szczególne postaci klauzuli Horna, zapisywane w przedstawionej konwencji, mają w programowaniu logicznym specyficzne nazwy:

$\lambda \Leftarrow \lambda_1, \ldots, \lambda_n$ – pełna postać klauzuli jest nazywana *regułą*,
$\lambda \Leftarrow$ – klauzula bez literałów negatywnych jest nazywana *faktem*,
$\Leftarrow \lambda_1, \ldots, \lambda_n$ – klauzula bez literału pozytywnego – *zanegowane pytanie*,
$\square$ – klauzula pusta – *sprzeczność*.

Klauzule-fakty i klauzule-reguły, jako klauzule zawierające literały pozytywne, stanowią treść programu logicznego. Zbiór tych klauzul określa się jako wiedzę, którą dysponuje program. Na podstawie posiadanej wiedzy program może udzielać odpowiedzi na kierowane do niego pytania.

W nowej konwencji zapisu reguła rezolucji dla rachunku kwantyfikatorów przyjmuje postać

$$\frac{\Leftarrow \lambda, \lambda_1, \ldots, \lambda_k \quad \lambda' \Leftarrow \lambda'_1, \ldots, \lambda'_l}{\Leftarrow \lambda\sigma, \ldots, \lambda'_1\sigma, \lambda_1\sigma, \ldots, \lambda_k\sigma} \quad \text{gdzie } \sigma = NOU(\lambda, \lambda') \qquad \text{dla } k, l \geq 0$$

Wyznaczenie rezolwenty klauzul:

$\Leftarrow \lambda, \lambda_1, \ldots, \lambda_n$
$\lambda' \Leftarrow \lambda'_1, \ldots, \lambda'_n$

sprowadza się do znalezienia najbardziej ogólnego unifikatora σ dla literałów λ oraz λ', a następnie do tekstowego zastąpienia literału λ w pierwszej z klauzul, przez prawą stronę drugiej z klauzul, ukonkretnioną podstawieniem σ, i ukonkretnienie pozostałych jej literałów, również podstawieniem σ.

Wprowadzone pojęcia i oznaczenia pozwalają na przedstawienie prostych programów logicznych.

Przykład 12.17

Treść prostego programu złożonego tylko z faktów przedstawia się następująco:

(1) *kocha*(*EWA*, *JAN*) $\Leftarrow$
(2) *kocha*(*EWA*, *JACEK*) $\Leftarrow$
(3) *kocha*(*JAN*, *KASIA*) $\Leftarrow$

Każdy z faktów składa się z dwuargumentowego predykatu *kocha*. Argumentami faktów są stałe reprezentowane napisami *JAN*, *EWA*, *JACEK*, *KASIA*.

Faktom tym można przypisywać pewną interpretację, na przykład *kocha*(*A*,*B*) można rozumieć, że pewien obiekt (osoba), reprezentowana przez stałą *A*, „kocha" inny obiekt (osobę), reprezentowaną przez stała *B*. Należy zwrócić uwagę, że zwrot „*A* kocha *B*" należy do dziedziny interpretacji.

W przypadku pytania:

Czy prawdą jest, że *kocha*(*JAN*, *KASIA*)?

program, na podstawie posiadanej wiedzy, odpowie oczywiście *tak*.

Odpowiedź wynika ze stwierdzenia niespełnialności zbioru złożonego z klauzul stanowiących treść programu i klauzuli

(4) $\Leftarrow$ *kocha*(*JAN*, *KASIA*)

stanowiącej negację pytania. Na postawie reguły rezolucji, z klauzul (3) i (4) wynika bowiem rezolwenta pusta

$$\frac{\Leftarrow kocha(JAN, KASIA) \quad kocha(JAN, KASIA) \Leftarrow}{\square}$$

Na pytanie natomiast:

Czy prawdą jest, że *kocha*(*KASIA*, *JAN*)?

ten sam program da oczywiście odpowiedź negatywną. Wynika to z tego, że ze zbioru zawierającego klauzule (1), (2), (3) oraz klauzulę (5) postaci

(5) $\Leftarrow$ *kocha*(*JAN*, *KASIA*)

nie daje się wyprowadzić żadnej nowej klauzuli, a zbiór ten nie zawiera klauzuli pustej.

Należy zwrócić uwagę na sens negatywnej odpowiedzi udzielanej przez program. Mechanizm odpowiedzi opiera się na tak zwanym założeniu o *zamkniętości świata*. Oznacza to, że program przyjmuje za fałszywe wszystko to, co nie da się udowodnić na gruncie posiadanej przez niego wiedzy. Odpowiedź negatywną należy ściśle rozumieć następująco: na gruncie posiadanej wiedzy nie daje się stwierdzić, że zdanie stanowiące pytanie jest logiczną konsekwencją wiedzy posiadanej przez program.

Przykład 12.18

Niech program złożony z faktów i jednej reguły przedstawia się następująco:

(1) *kocha*(*EWA*, *JAN*) $\Leftarrow$
(2) *kocha*(*EWA*, *JACEK*) $\Leftarrow$
(3) *kocha*(*JAN*, *KASIA*) $\Leftarrow$
(4) *kocha*(x, y) $\Leftarrow$ *kocha*(y, x)

W odpowiedzi na pytanie:

Czy prawdą jest, że *kocha*(*KASIA*, *JAN*)?

program dołączy do swojej treści klauzulę stanowiącą negację pytania:

(5) $\Leftarrow$ *kocha*(*KASIA*, *JAN*)

i może podjąć obliczenie:

(6) $\Leftarrow$ *kocha*(*JAN*, *KASIA*) z (4), (5), dla $\sigma = [x ::= KASIA, y ::= JAN]$
(7) $\square$ z (3), (6)

co daje podstawę do odpowiedzi *tak*.

Pytania, już w postaci zanegowanej, mogą mieć postać ogólniejszą, na przykład:

(5a) $\Leftarrow$ *kocha*(*KASIA*, *z*)

(5b) $\Leftarrow$ *kocha*(*z*, *JAN*)

Odpowiedź na takie pytania nie sprowadza się tylko do stwierdzenia *tak* albo *nie*. Polega ona na wskazaniu tych wszystkich obiektów, reprezentowanych przez zmienną *z*, dla których pytanie będzie prawdziwe.

W celu udzielenia odpowiedzi na pierwsze z tych pytań obliczenia programu mogą być następujące:

(6a) $\Leftarrow$ *kocha*(*z*, *KASIA*)	z (4), (5a), dla [x ::= *KASIA*, y ::= z]
(7a) $\square$	z (3), (6a), dla [z ::= *JAN*]

Obliczenie kończy się wygenerowaniem klauzuli pustej, przy ustalonym wartościowaniu zmiennej *z*. Informacja zawarta w ostatnim unifikatorze jest podstawą do odpowiedzi, wartość przypisywana zmiennej *z* wskazuje na poszukiwany obiekt. Odpowiedzią na pytanie będzie więc zbiór jednoelementowy {*JAN*}.

Odpowiedzi na pytanie (5b) można udzielić na podstawie dwóch różnych obliczeń:

(6b) $\square$	z (1), (5a), dla [z ::= *EWA*]

oraz

(6b′) $\Leftarrow$ *kocha*(*JAN*, *z*)	z (4), (5b), dla [x ::= z]
(7b) $\square$	z (3), (5a), dla [z ::= *KASIA*]

Obliczenia prowadzą do wskazania dwóch różnych obiektów, dlatego odpowiedzią na pytanie jest zbiór dwuelementowy {*EWA*, *KASIA*}.

Ćwiczenia

1. Następujące formuły sprowadzić do postaci skolemowskiej:
   a) $\exists y \bullet (y < 1)$
   b) $\forall x \bullet \exists y \bullet (x < y)$
   c) $\forall x \bullet \forall y \bullet \exists z \bullet ((x < y) \Rightarrow (x < z) \wedge (z < y))$
2. Pokazać przykład formuły, dla której odpowiednik w postaci skolemowskiej nie jest jej równoważny semantycznie.
3. Sprawdzić, które z podanych niżej zbiorów klauzul są zbiorami spełnialnymi:
   a) $\{a \vee \neg b, a \vee c, b \wedge \neg c\}$
   b) $\{\neg a \neg b, b \vee \neg c, b, a\}$
   c) $\{a \vee b, a, \neg b, \neg a \vee c\}$

4. Stosując metodę rezolucji, zbadać spełnialność niżej podanych formuł:
   a) $(p \vee q) \Leftrightarrow (\neg p \wedge \neg q)$
   b) $p \vee (q \vee r) \Leftrightarrow (p \vee q) \vee r$
   c) $(a \Rightarrow b) \wedge (\neg b \Rightarrow \neg a) \Rightarrow a$

5. Wyznaczyć najbardziej ogólny unifikator dla formuł:
   a) $p(y, 1)$ $p(x, 2)$
   b) $q(x, y)$ $q(y, x)$
   c) $p(x, y)$ $q(z, y)$
   d) $p(x, f(x))$ $p(y, f(y))$
   e) $r(h(x, y)), f(z))$ $r(h(g(x), y), f(f(x)))$

   gdzie p, q, r są symbolami predykatów, f, g, h – symbolami funkcji, x, y, z – symbolami zmiennych indywiduowych.

6. Dany jest zbiór klauzul:
   (1) *samochód*$(x) \Leftarrow$ pojazd(x), *ma_4_koła*(x)
   (2) *jeździ*$(x) \Leftarrow$ *samochód*(x)
   (3) *pojazd*$(x) \Leftarrow$ *polonez*(x)
   (4) *ma_4_koła*$(x) \Leftarrow$ *polonez*(x)
   (5) $\Leftarrow$ *polonez*(WCL_2222)

   Metodą rezolucji znajdź odpowiedź na pytanie czy *jeździ*(WCL_2222).

7. Zagadnienia przedstawione w niżej podanej postaci sprowadzić do programu logicznego. Sprawdzić, czy przedstawione wnioski są poprawne.
   a) *Wszyscy ludzie są śmiertelni.*
      Sokrates jest człowiekiem.
      Zatem: *Sokrates jest śmiertelny.*
   b) *Wszyscy wykładowcy są zdecydowani.*
      Każdy kto jest zdecydowany i inteligentny świadczy dobre usługi.
      Klara jest inteligentnym wykładowcą.
      Zatem: *Klara świetnie wykłada.*

13. Zagadnienia uzupełniające

13.1. Wstęp

Każdy sformalizowany system dedukcyjny (system dowodzenia) jest określony jako para $<A, R>$, gdzie: A jest zbiorem aksjomatów, R – zbiorem reguł dedukcyjnych (reguł wnioskowania). Wyróżnia się dwa rodzaje systemów dedukcyjnych logiki: systemy aksjomatyczne i systemy dedukcji naturalnej. Zasadniczą cechą systemów *dedukcji naturalnej* jest to, że mają dwa rodzaje reguł wnioskowania: reguły wprowadzania i reguły eliminacji spójników logicznych. Rodowód systemów aksjomatycznych sięga końca XIX wieku, a systemy dedukcji naturalnej powstały w latach trzydziestych XX wieku – ich inicjatorami byli Gentzen i Jaśkowski[22].

Do systemów aksjomatycznych zalicza się między innymi systemy Hilberta[23], systemy tablic analitycznych, najpowszechniej zaś stosowany system dedukcji naturalnej pochodzi od Gentzena. W tym rozdziale przedstawiono w zarysie system Hilberta, system dedukcji naturalnej Gentzena oraz metodę tablic analitycznych.

Systemy dowodzenia Hilberta uznaje się za tradycyjne. Mają one zarówno znaczenie historyczne, jak i powszechne zastosowanie w praktyce matematycznej. Na początku XX wieku Hilbert zainicjował w zakresie podstaw matematyki kierunek określany jako *formalizm*. Formalizm skupiał się na poszukiwaniu systemu, dzięki zastosowaniu którego dałoby się, w skończonym postępowaniu, udowodnić dowolne twierdzenia matematyki. Gödel[24], w latach trzydziestych XX wieku, zakończył te poszukiwania, pokazując, że budowa takiego systemu nie jest możliwa. Systemy dowodzenia Hilberta pozostały jednak użyteczne do dzisiaj.

System Hilberta był w zasadzie pierwszym formalnym systemem aksjomatyzacji. Jest to system uniwersalny, gdyż znajduje zastosowanie nie tylko w logice klasycznej, ale także w logikach nieklasycznych. W odróżnieniu od poprzednio omawianych systemów dowodzenia, które opierały się na dowodzeniu nie wprost, dowodzenie w systemach Hilberta polega na konstrukcji dowodów wprost.

---

[22] Stanisław Jaśkowski (1906–1965).

[23] David Hilbert (1862–1943).

[24] Kurt Gödel (1906–1978).

Gentzen opracował dwie różne metody dedukcji, każdą w dwóch wariantach – jeden dla logiki klasycznej i drugi dla logiki intuicjonistycznej. Jedna z tych metod to omówiony wcześniej rachunek sekwentów, a druga to metoda dedukcji naturalnej. Poniżej omawia się system dedukcji naturalnej tylko dla logiki klasycznej.

System dedukcji naturalnej dla rachunku zdań przypomina system dowodzenia dla rachunku zdań oparty na rachunku sekwencji. System dedukcji naturalnej ma te same reguły eliminacji spójników logicznych. W systemie są ponadto reguły wprowadzania spójników logicznych. Specyficzną właściwością systemu jest to, że w regułach mogą występować wyróżnione zdania, które traktuje się jako założenia (hipotezy robocze). Założenia takie są przydatne do wyprowadzania pewnych wniosków, po czym – po wyprowadzeniu takich wniosków – z założeń tych można zrezygnować. Oznacza to, że wyprowadzone wnioski są słuszne, niezależnie od poczynionych początkowo założeń. Ten sposób postępowania jest często stosowany w praktyce dowodowej i stąd bierze się termin dedukcji naturalnej.

Metoda tablic analitycznych jest pewnego rodzaju odpowiednikiem metody rezolucji. Główne różnice sprowadzają się do badania formuł zapisywanych w dysjunkcyjnej postaci normalnej.

13.2. Systemy dowodzenia Hilberta

Przedstawiany poniżej system Hilberta odnosi się tylko do klasycznego rachunku zdań i rachunku kwantyfikatorów.

System dowodzenia Hilberta H składa się z dwóch elementów: zbioru *aksjomatów* oraz zbioru *reguł inferencji* (lub *wnioskowania*), czyli zasad tekstowej transformacji jednych formuł w inne. Na podstawie pewnych formuł reguły wyprowadzają nowe formuły. Mówi się, że na podstawie reguł pewne formuły wynikają z innych.

Definicja 13.1

Dowodem w systemie H nazywa się skończony ciąg formuł $\alpha_1, ..., \alpha_n$ taki, że każda z formuł jest albo aksjomatem, albo wynika z poprzednich formuł w wyniku zastosowania jednej z reguł wnioskowania. Formułę α_n nazywa się *twierdzeniem* w systemie H.

Definicja 13.2

Derywacją ze zbioru formuł Φ w systemie H nazywa się skończony ciąg formuł $\alpha_1, ..., \alpha_n$ taki, że każda z formuł jest albo aksjomatem, albo jest jedną z formuł zbioru Φ, albo wynika z poprzednich formuł po zastosowaniu jednej z reguł wnioskowania. Formułę α_n nazywa się *konsekwencją składniową* ze zbioru Φ w systemie H.

Fakt, że formuła α jest konsekwencją składniową ze zbioru formuł Φ w systemie H, zapisuje się w postaci

$$\Phi \vdash_H \alpha$$

Zamiast $\varnothing \vdash_H \alpha$ pisze się $\vdash_H \alpha$, co oznacza, że α jest twierdzeniem. Symbol $\vdash$ jest nazywany symbolem *konsekwencji składniowej*.

Dla klasycznego rachunku zdań, opartego na funkcjonalnie zupełnym zbiorze spójników logicznych zawierającym negację $\neg$, implikację $\Rightarrow$ i stałe logiczne **false** i **true**, przykładowy system Hilberta składa się z następujących schematów aksjomatów:

Schematy aksjomatów

1. $\alpha \Rightarrow (\beta \Rightarrow \alpha)$ – prawo symplifikacji,
2. $(\alpha \Rightarrow (\beta \Rightarrow \gamma)) \Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \gamma))$ – prawo Fregego,
3. **false** $\Rightarrow \alpha$
4. $\alpha \Rightarrow$ **true**
5. $\neg\neg\alpha \Rightarrow \alpha$
6. $\alpha \Rightarrow (\neg\alpha \Rightarrow \beta)$
7. $(\alpha \wedge \beta) \Rightarrow \alpha$
8. $(\alpha \wedge \beta) \Rightarrow \beta$
9. $(\alpha \Rightarrow \gamma) \Rightarrow ((\beta \Rightarrow \gamma) \Rightarrow (\alpha \vee \beta \Rightarrow \gamma))$

Schemat aksjomatu oznacza faktycznie nieskończony zbiór formuł, które różnią się od formuły występującej w schemacie aksjomatu tym, że każde wystąpienie symbolu α, β, γ może być zastąpione dowolną formułą. Symbole α, β, γ są więc symbolami pomocniczymi, reprezentującymi dowolne formuły.

Jedyną regułą wnioskowania jest *reguła odrywania* (*modus ponens*)

$$\frac{\alpha, \alpha \Rightarrow \beta}{\beta}$$

Sens reguły jest następujący: jeżeli w trakcie pewnej derywacji wyprowadzono formuły α oraz $\alpha \Rightarrow \beta$, to niezależnie od interpretacji, jaką się przypisuje formułom α oraz β, dopuszczalnym wnioskiem jest β.

Czasem system Hilberta przedstawia się inaczej. Zamiast schematów aksjomatów wprowadza się *aksjomaty* i dodatkową *regułę podstawienia* (*zastąpienia*). Reguła podstawienia pozwala na zastąpienie zmiennych zdaniowych występujących w formule przez inne formuły. Formalnie reguła podstawienia ma postać

$$\frac{\alpha}{\alpha[a ::= \beta]}$$

gdzie: α, β są dowolnymi formułami, a a jest zmienną zdaniową. Zapis $\alpha[a ::= \beta]$ oznacza formułę, która powstaje z formuły α przez tekstowe zastąpienie każdego wystąpienia zmiennej a przez formułę β.

Przykład 13.1

Formuła $a \Rightarrow a$ jest twierdzeniem. Dowodem dla tej formuły jest ciąg formuł:

(1) $(a \Rightarrow ((a \Rightarrow a) \Rightarrow a)) \Rightarrow ((a \Rightarrow (a \Rightarrow a)) \Rightarrow (a \Rightarrow a))$
– aksjomat 2 z $[\alpha ::= a, \gamma ::= a, \beta ::= a \Rightarrow a]$

(2) $(a \Rightarrow ((a \Rightarrow a) \Rightarrow a)$ – aksjomat 1 z $[\alpha ::= a, \beta ::= a \Rightarrow a]$

(3) $(a \Rightarrow (a \Rightarrow a)) \Rightarrow (a \Rightarrow a)$ – reguła odrywania zastosowana do (1), (2)

(4) $a \Rightarrow (a \Rightarrow a)$ – aksjomat 1z $[\alpha ::= a, \beta ::= a]$

(5) $a \Rightarrow a$ – reguła odrywania zastosowana do (3), (4)

Przykład wskazuje na uciążliwość w prowadzeniu dowodów dla bardziej złożonych formuł. Ten przykład nie nasuwa ponadto wskazówek dotyczących taktyki prowadzenia dowodów. W stosunku do wcześniej przedstawionych systemów dowodzenia, system Hilberta jest trudniej algorytmizowalny.

Chociaż system Hilberta jest uciążliwy w stosowaniu do logiki klasycznej, to często jest on używany w logikach nieklasycznych, gdy zawodzą inne systemy. Poniżej przedstawia się zarys algorytmu postępowania przy dowodzeniu formuł z zastosowaniem systemu Hilberta $H =_{def} <A, R>$, złożonego ze zbioru aksjomatów $A =_{def} \{A_1, ..., A_n\}$ i zbioru reguł $R =_{def} \{R_1, ..., R_m\}$.

Algorytm automatycznego wnioskowania w systemie dowodzenia Hilberta *H*

Dane: formuła α.

Wynik: odpowiedź *tak*, gdy α jest twierdzeniem w systemie Hilberta, oraz *nie* w przypadku przeciwnym.

Procedura:

1. Niech Φ będzie zmienną reprezentującą zbiór formuł, a początkowa zawartość zbioru $\Phi = A$.
2. **while** $\alpha \notin \Phi$ oraz $\neg\alpha \notin \Phi$
   do
   stosuj reguły ze zbioru R, przyjmując za ich przesłanki formuły ze zbioru Φ, i rozszerzaj zbiór Φ o nowo otrzymane wnioski
   od
3. Jeżeli $\alpha \in \Phi$, to odpowiedz *tak*, jeżeli $\neg\alpha \in \Phi$ – odpowiedz *nie*.

Przedstawione dalej twierdzenie o dedukcji, udowodnione niezależnie przez Tarskiego i Herbranda, ma znaczenie praktyczne, gdyż jego dowód pokazuje, jak derywację $\{\alpha\} \vdash_H \beta$ można, w sposób konstruktywny, przekształcić w dowód twierdzenia $\alpha \Rightarrow \beta$. Ponieważ na ogół łatwiej jest znaleźć derywację niż dowód, twierdzenie pozwala na oszczędność wysiłku.

Twierdzenie 13.1 (*Twierdzenie o dedukcji*)

W dowolnym systemie H, zawierającym przynajmniej schematy aksjomatów 1, 2 oraz regułę odrywania, jako jedyną regułę wnioskowania, derywacja

$$\Phi \cup \{\alpha\} \vdash_H \beta$$

zachodzi wtedy i tylko wtedy, gdy zachodzi derywacja

$$\Phi \vdash_H (\alpha \Rightarrow \beta).$$

Dowód

Jeżeli zachodzi $\Phi \vdash_H (\alpha \Rightarrow \beta)$, to oczywiście zachodzi $\Phi \cup \{\alpha\} \vdash_H \beta$.

Wynikanie w przeciwnym kierunku jest trudniejsze do pokazania. Niech

$$\Phi \cup \{\alpha\} \vdash_H \beta$$

czyli istnieje pewien ciąg formuł

$$\gamma_1, \ldots, \gamma_n \qquad (D_1)$$

który jest derywacją β ze zbioru $\Phi \cup \{\alpha\}$. Formuła γ_i, dla $i = 1, \ldots, n$, jest elementem zbioru $\Phi \cup \{\alpha\}$ lub wynika z formuł poprzedzających w rezultacie zastosowania reguły odrywania, dodatkowo $\gamma_n \equiv \beta$. Ciąg (D_1) można przekształcić w ciąg stanowiący derywację $\alpha \Rightarrow \beta$ ze zbioru Φ. Najpierw każdą formułę z (D_1) poprzedza się prefiksem $\alpha \Rightarrow$, tworząc ciąg

$$\alpha \Rightarrow \gamma_1, \ldots, \alpha \Rightarrow \gamma_n \qquad (D_2)$$

Ciąg ten kończy się formułą $\alpha \Rightarrow \beta$, gdyż $\gamma_n \equiv \beta$. Ciąg (D_2) nie jest jeszcze prawidłową derywacją. Przekształca się go, dołączając dodatkowe formuły zgodnie z następującymi zasadami:

Jeżeli γ_i jest aksjomatem lub elementem zbioru Φ, to przed $\alpha \Rightarrow \gamma_i$ umieszcza się dwie dodatkowe formuły

$$\gamma_i, \gamma_i \Rightarrow (\alpha \Rightarrow \gamma_i)$$

Jeżeli γ_i jest formułą α, to przed $\alpha \Rightarrow \alpha$ umieszcza się ciąg formuł stanowiących dowód dla formuły $\alpha \Rightarrow \alpha$ (zob. przykład 12.1).

Jeżeli γ_i w ciągu (D_1) pojawia się jako wniosek z zastosowania reguły odrywania, to oznacza, że istnieją takie γ_j, γ_k, dla $j, k < i$, przy czym $\gamma_k \equiv \gamma_j \Rightarrow \gamma_i$. W ciągu (D_2) elementom tym odpowiadają formuły: $\alpha \Rightarrow \gamma_j$ oraz $\alpha \Rightarrow \gamma_k$ (czyli $\alpha \Rightarrow (\gamma_j \Rightarrow \gamma_i)$). Przed formułę $\alpha \Rightarrow \gamma_j$ wstawia się formułę

$$(\alpha \Rightarrow (\gamma_j \Rightarrow \gamma_i)) \Rightarrow (\alpha \Rightarrow \gamma_i)$$

która jest aksjomatem, oraz formułę

$$(\alpha \Rightarrow \gamma_j) \Rightarrow (\alpha \Rightarrow \gamma_i)$$

która wynika z zastosowania reguły odrywania do formuł ją poprzedzających. Teraz również formuła $\alpha \Rightarrow \gamma$, wynika z zastosowania reguły odrywania do formuł ją poprzedzających. Łatwo sprawdzić, że tak zmodyfikowany ciąg (D_2) stanowi derywację formuły $\alpha \Rightarrow \beta$ ze zbioru Φ. ■

Przykład 13.2

Formuła $(\alpha \Rightarrow (\beta \Rightarrow \gamma)) \Rightarrow (\beta \Rightarrow (\alpha \Rightarrow \gamma)$ jest twierdzeniem. Można to pokazać, korzystając z twierdzenia o dedukcji. Najpierw należy zauważyć, że

$$\{\alpha \Rightarrow (\beta \Rightarrow \gamma), \beta, \alpha\} \vdash_H \gamma$$

co wynika z następującej derywacji:

(1) $\alpha \Rightarrow (\beta \Rightarrow \gamma)$	– element zbioru $\{\alpha \Rightarrow (\beta \Rightarrow \gamma), \beta, \alpha\}$
(2) α	– element zbioru $\{\alpha \Rightarrow (\beta \Rightarrow \gamma), \beta, \alpha\}$
(3) $\beta \Rightarrow \gamma$	– reguła odrywania zastosowana do (1), (2)
(4) β	– element zbioru $\{\alpha \Rightarrow (\beta \Rightarrow \gamma), \beta, \alpha\}$
(5) γ	– reguła odrywania zastosowana do (3), (4)

Z twierdzenia o dedukcji wynika, że

$$\{\alpha \Rightarrow (\beta \Rightarrow \gamma), \beta\} \vdash_H \alpha \Rightarrow \gamma$$

oraz ponownie

$$\{\alpha \Rightarrow (\beta \Rightarrow \gamma)\} \vdash_H \beta \Rightarrow (\alpha \Rightarrow \gamma)$$

i ostatecznie

$$\delta_H (\alpha \Rightarrow (\beta \Rightarrow \gamma)) \Rightarrow (\beta \Rightarrow (\alpha \Rightarrow \gamma)$$

Systemów dowodzenia Hilberta dla rachunku zdań jest wiele. Przedstawiony niżej system różni się od systemu przedstawionego poprzednio tylko zbiorem aksjomatów. Wynika to z tego, że aksjomaty zawierają tylko negację i implikację. Przypomina się, że rachunek zdań wykorzystujący tylko te spójniki jest funkcjonalnie zupełny. Pozostałe spójniki mogą być definiowane za pomocą spójników podstawowych. W definicjach tych stosuje się wcześniej wprowadzone pojęcie równoważności semantycznej.

Aksjomaty

1. $(\alpha \Rightarrow (\beta \Rightarrow \alpha))$	prawo symplifikacji,
2. $(\alpha \Rightarrow (\beta \Rightarrow \gamma)) \Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \gamma))$	prawo Fregego,
3. $(\neg\alpha \Rightarrow (\alpha \Rightarrow \beta))$	prawo Dunsa Scotusa,
4. $((\neg\alpha \Rightarrow \alpha) \Rightarrow \alpha)$	prawo Claviusa.

W razie potrzeby użycia dodatkowych spójników lub stałych logicznych, wprowadza się ich definicje jako złożenie implikacji i negacji. Na przykład:

Definicje

1. $\alpha \wedge \beta =_{def} \neg(\alpha \Rightarrow \neg\beta)$
2. $\alpha \vee \beta =_{def} (\neg\alpha \Rightarrow \beta)$
3. $(\alpha \Leftrightarrow \beta) =_{def} (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
4. $\mathbf{true} =_{def} \alpha \Rightarrow \alpha$
5. $\mathbf{false} =_{def} \neg(\alpha \Rightarrow \alpha)$

Definicje pozwalają na tekstowe zastąpienie w dowolnej formule dowolnej jej podformuły, równoważnej tekstowo z jedną ze stron definicji, przez drugą ze stron tej samej definicji.

System Hilberta dla rachunku kwantyfikatorów ma wszystkie aksjomaty i reguły systemu dla rachunku zdań oraz dodatkowo jeden schemat aksjomatu i jedną regułę:

Schemat aksjomatu

10. $(\forall x \bullet \alpha) \Rightarrow \alpha[x ::= t]$

Reguła uogólniania

$$\frac{\alpha \Rightarrow \beta}{\alpha \Rightarrow \forall x \bullet \beta}$$ pod warunkiem, że $x \notin FV(\alpha)$

Szczególna postać tej reguły jest następująca:

$$\frac{\beta}{\forall x \bullet \beta}$$

Przykład 13.3

Rozpatruje się zarys dowodu dla formuły

$$(\forall x \bullet (p(x) \wedge q(x))) \Rightarrow (\forall x \bullet p(x)) \quad (1)$$

Przyjmując, że $t \equiv x$, na podstawie schematu aksjomatu 10, zachodzi implikacja

$$(\forall x \bullet (p(x) \wedge q(x))) \Rightarrow (p(x) \wedge q(x)) \quad (2)$$

Łatwo sprawdzić, że tautologią jest formuła

$$(p(x) \wedge q(x)) \Rightarrow p(x) \quad (3)$$

Z implikacji (2) i implikacji (3), na podstawie wnioskowania łańcuchowego, wynika formuła

$$(\forall x \bullet (p(x) \wedge q(x))) \Rightarrow p(x) \quad (4)$$

stąd i z reguły uogólniania wynika, że

$$(\forall x \bullet (p(x) \wedge q(x))) \Rightarrow (\forall x \bullet p(x))$$

W systemie Hilberta dla rachunku kwantyfikatorów zachodzi twierdzenie o dedukcji, tak jak dla systemu Hilberta dla rachunku zdań.
System Hilberta H dla rachunku kwantyfikatorów jest semantycznie niesprzeczny i semantycznie zupełny, tzn. dla dowolnego zbioru formuł Φ zachodzi twierdzenie:

Twierdzenie 13.2

$\Phi \vdash_H \alpha$ wtedy i tylko wtedy, gdy $\Phi \models \alpha$.

13.3. System dedukcji naturalnej Gentzena

Rozpatruje się rachunek zdań oparty na zbiorze spójników logicznych, zawierającym stałe **true**, **false**, negację $\neg$, koniunkcję $\wedge$ i implikację $\Rightarrow$.

Zestaw reguł wprowadzania (oznaczanych symbolem I) oraz eliminacji (oznaczanych symbolem E) jest następujący:

$$(\textbf{true}\ I)\ \frac{}{\textbf{true}} \qquad (\textbf{false}\ E)\ \frac{\textbf{false}}{\alpha}$$

$$(\neg I)\ \frac{\begin{array}{c}[\alpha]\\ \vdots \\ \textbf{false}\end{array}}{\neg\alpha} \qquad (\neg E)\ \frac{\begin{array}{c}[\neg\alpha]\\ \vdots \\ \textbf{false}\end{array}}{\alpha} \qquad (\neg E)\ \frac{\alpha, \neg\alpha}{\textbf{false}}$$

$$(\wedge I)\ \frac{\alpha, \beta}{\alpha \wedge \beta} \qquad (\wedge E)\ \frac{\alpha \wedge \beta}{\alpha} \qquad (\wedge E)\ \frac{\alpha \wedge \beta}{\beta}$$

$$(\Rightarrow I)\ \frac{\begin{array}{c}[\alpha]\\ \vdots \\ \beta\end{array}}{\alpha \Rightarrow \beta} \qquad (\Rightarrow E)\ \frac{\alpha, \alpha \Rightarrow \beta}{\beta}$$

Reguły związane ze stałymi logicznymi są specyficzne. Reguła wprowadzania stałej **true** (**true** I) nie ma przesłanek, nie ma też reguły eliminacji tej stałej. Reguła eliminacji **false** (**false** E) pozwala na wyprowadzenie z przesłanki **false** dowolnego wniosku, nie ma natomiast reguły wprowadzania dla **false**.

Dla koniunkcji reguły wprowadzania ($\wedge I$) oraz eliminacji ($\wedge E$) są oczywiste: jeżeli przesłankami są formuły α oraz β, to można wnioskować, że $\alpha \wedge \beta$, oraz odwrotnie – z przesłanki $\alpha \wedge \beta$ można wnioskować, że α (lub, że β).

Reguła eliminacji implikacji ($\Rightarrow E$) jest poznaną wcześniej regułą odrywania.

Pozostałe reguły wymagają dodatkowych wyjaśnień.

Pierwszą z nich jest reguła wprowadzenia negacji ($\neg I$). Pozwala ona na wprowadzenie symbolu negacji przed dowolną formułę α na podstawie przesłanki, którą jest wnioskowanie, że z założenia o prawdziwości α wynika **false**. Jest ona odzwierciedleniem dowodzenia nie wprost przez sprowadzenie do sprzeczności. Przesłanka reguły, mająca postać

$[\alpha]$
$\vdots$
false

oznacza pewne wnioskowanie (oznaczone symbolicznie pionowym zestawem trzech kropek $\vdots$), które na podstawie założenia α prowadzi do wniosku **false**, czyli do sprzeczności. Jeżeli na podstawie przyjętego założenia, że spełnione jest α, otrzymuje się sprzeczność – formułę **false**, to wnioskiem jest, że spełnione jest $\neg\alpha$. Wniosek ten jest przy tym niezależny od początkowo przyjętego założenia. Oznacza to, że od momentu przyjęcia wniosku $\neg\alpha$ założenie α staje się już nieprzydatne do dalszych wnioskowań i można je usunąć, co symbolicznie oznacza się przez zamknięcie założenia w kwadratowe nawiasy $[\alpha]$.

Podobny komentarz odnosi się do reguły ($\neg E$): jeżeli przyjęte założenie $\neg\alpha$ prowadzi do sprzeczności, to wnioskiem, jaki należy wyprowadzić, jest α.

Druga z reguł eliminacji negacji ($\neg E$) jest oczywista: jeżeli przesłankami wnioskowania są dowolna formuła i jej negacja, to wnioskiem jest stała **false** oznaczająca sprzeczność.

W przesłance reguły wprowadzania implikacji ($\Rightarrow I$) założeniem jest formuła α. Jeżeli się pokaże, że z tego założenia daje się wyprowadzić formułę β, to oznacza, że niezależnie od tego założenia zachodzi implikacja $\alpha \Rightarrow \beta$.

Poniżej przedstawia się przykłady zastosowania metody dedukcji naturalnej w dowodzeniu prostych formuł. Podobnie jak w przypadku metody sekwentów, dowód (albo ogólniej derywacja) ma strukturę drzewa: wierzchołki drzewa są etykietowane formułami, a łuki – przejścia pomiędzy wierzchołkami – odpowiadają zastosowaniu odpowiednich reguł.

Przykład 13.4

Poniższej przedstawia się drzewa dowodu dla trzech prostych formuł. Pierwsza formuła ma postać: $\alpha \wedge \beta \Rightarrow \beta \wedge \alpha$.

$$\dfrac{\dfrac{\dfrac{[\alpha \wedge \beta]_1}{\beta}(\wedge E)\quad \dfrac{[\alpha \wedge \beta]_1}{\alpha}(\wedge E)}{\beta \wedge \alpha}(\wedge I)}{\alpha \wedge \beta \Rightarrow \beta \wedge \alpha}(\Rightarrow I_1)$$

W dowodzie tym korzysta się dwukrotnie tylko z jednego, tego samego założenia, że $\alpha \wedge \beta$. Założenia są numerowane. Po prawej stronie każdego przejścia podaje się symbol wykorzystywanej reguły. Dodatkowo, w tych przypadkach, gdy wykorzystanie reguły wiąże się z wykorzystaniem i usunięciem wprowadzonego założenia, podaje się numer tego założenia.

Przykład 13.5

Kolejny dowód dotyczy formuły $\alpha \Rightarrow \neg\neg\alpha$. Tym razem wykorzystuje się dwa założenia: α oraz $\neg\alpha$.

$$\dfrac{\dfrac{\dfrac{[\alpha]_2 \quad [\neg\alpha]_1}{\textbf{false}}(\neg E)}{\neg\neg\alpha}(\neg I_1)}{\alpha \Rightarrow \neg\neg\alpha}(\Rightarrow I_2)$$

Przykład 13.6

Ostatni przykład, najbardziej złożony, dotyczy formuły: $\neg(\alpha \Leftrightarrow \neg\alpha)$. W dowodzie przyjmuje się, że równoważności przedstawia się jako koniunkcje implikacji, tzn. formuła $\alpha \Leftrightarrow \beta$ jest skróconą formą zapisu $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$, stąd biorą się wyprowadzenia:

$$\dfrac{\alpha \Leftrightarrow \beta}{\alpha \Rightarrow \beta} \qquad \dfrac{\alpha \Leftrightarrow \beta}{\beta \Rightarrow \alpha}$$

Ze względu na wymiary wywodu, pominięto komentarze dotyczące stosowanych reguł, pozostawiając tylko numerację założeń i wskazania tych miejsc, w których założenia zostały wykorzystane.

$$\dfrac{\dfrac{\dfrac{\dfrac{[\alpha \Leftrightarrow \neg\alpha]_3}{[\alpha]_1 \quad \alpha \Rightarrow \neg\alpha}(\Rightarrow E) \quad [\alpha]_1}{\neg\alpha}}{\textbf{false}}(\neg I_1) \quad \dfrac{[\alpha \Leftrightarrow \neg\alpha]_3}{\neg\alpha \Rightarrow \alpha} \quad \dfrac{\dfrac{\dfrac{[\alpha \Leftrightarrow \neg\alpha]}{[\alpha]_2 \quad \alpha \Rightarrow \neg\alpha}}{\neg\alpha} \quad [\alpha]_2}{\textbf{false}}(\neg I_2)}{\alpha} \quad 5}{\textbf{false}}(\neg I_3)}{\neg(\alpha \Leftrightarrow \neg\alpha)}$$

Przedstawione przykłady pozwalają na łatwiejsze zrozumienie kolejnych pojęć. Pierwszym z nich jest pojęcie derywacji. Derywacja jest drzewem, którego wierzchołki są etykietowane formułami. Dalej takie drzewo będzie oznaczane symbolem D.

Definicja 13.3

Zbiorem derywacji nazywa się zbiór *DER*, który jest zdefiniowany rekursywnie w sposób następujący:

(1) Drzewo złożone z jednego wierzchołka etykietowanego formułą α jest derywacją.

(2) Jeżeli $\begin{matrix} D \\ \alpha \end{matrix} \in DER$, $\begin{matrix} D' \\ \beta \end{matrix} \in DER$, to $\dfrac{\begin{matrix} D & D' \\ \alpha & \beta \end{matrix}}{\alpha \wedge \beta} \in DER$

(3) Jeżeli $\begin{matrix} D \\ \alpha \wedge \beta \end{matrix} \in DER$, to $\dfrac{\begin{matrix} D \\ \alpha \wedge \beta \end{matrix}}{\alpha} \in DER$ oraz $\dfrac{\begin{matrix} D \\ \alpha \wedge \beta \end{matrix}}{\beta} \in DER$

(4) Jeżeli $\begin{matrix} \alpha \\ D \\ \phi \end{matrix} \in DER$, to $\dfrac{\begin{matrix} [\alpha] \\ D \\ \beta \end{matrix}}{\alpha \Rightarrow \beta} \in DER$

(5) Jeżeli $\begin{matrix} D \\ \alpha \end{matrix} \in DER$, $\begin{matrix} D' \\ \alpha \Rightarrow \beta \end{matrix} \in DERIV$, to $\dfrac{\begin{matrix} D & D' \\ \alpha & \alpha \Rightarrow \beta \end{matrix}}{\beta} \in DER$

(6) Jeżeli $\begin{matrix} D \\ \mathbf{false} \end{matrix} \in DER$, to $\dfrac{\begin{matrix} D \\ \mathbf{false} \end{matrix}}{\alpha} \in DER$

(7) Jeżeli $\begin{matrix} \neg\alpha \\ D \\ \mathbf{false} \end{matrix} \in DER$, to $\dfrac{\begin{matrix} [\neg\alpha] \\ D \\ \mathbf{false} \end{matrix}}{\alpha} \in DER$

Formuła, która jest korzeniem drzewa derywacji, nazywa się wnioskiem. Liśćmi drzewa są założenia. Założenia mogą być zamknięte (skreślone) albo otwarte.

Definicja 13.4

Relacja $\Phi \vdash_G \alpha$ pomiędzy zbiorem formuł Φ oraz formułą α, zdefiniowana następująco: istnieje pewna derywacja, w której Φ stanowi zbiór nieskreślonych założeń, a α jest wnioskiem, nazywa się *relacją derywacji*.

System dedukcji naturalnej dla rachunku kwantyfikatorów jest rozszerzeniem zbioru reguł dla rachunku zdań o dodatkowe reguły wprowadzania i eliminacji kwantyfikatora ogólnego:

$$(\forall I)\ \frac{\alpha}{\forall x \bullet \alpha} \qquad (\forall E)\ \frac{\forall x \bullet \alpha}{\alpha[x ::= t]}$$

W regule $(\forall I)$ wymaga się, aby zmienna x nie występowała jako zmienna wolna w żadnym z założeń, od których zależy formuła α, w regule $(\forall E)$ wymaga się natomiast, aby term t był wolny w formule α ze względu na zmienną x. Wymagania te są istotne, gdyż – jak pokazują poniższe przykłady – ich niespełnienie prowadzi do fałszywych wniosków, czyli narusza semantyczną poprawność systemu.

Przykład 13.7

Rozpatruje się następujące drzewo dowodu

$$\dfrac{\dfrac{\dfrac{\dfrac{[x=0]_1}{\forall x \bullet (x=0)}\,(\forall I)}{(x=0) \Rightarrow \forall x \bullet (x=0)}\,(\Rightarrow I_1)}{\forall x \bullet ((x=0) \Rightarrow \forall x \bullet (x=0))}\,(\forall I)}{(0=0) \Rightarrow \forall x \bullet (x=0)}\,(\forall E)$$

Powodem absurdalnego wniosku jest niepoprawne zastosowanie reguły $(\forall I)$.

Przykład 13.8

Niepoprawne zastosowanie reguły $(\forall E)$ prowadzi do następującego wywodu

$$\dfrac{\dfrac{[\forall x \bullet \neg \forall y \bullet (x=y)]_1}{\neg \forall y \bullet (x=y)\,[x ::= y]}\,(\forall E)}{\forall x \bullet \neg \forall y \bullet (x=y) \Rightarrow \neg \forall y \bullet (y=y)}\,(\Rightarrow I)$$

Kolejny przykład jest ilustracją poprawnego stosowania reguł wprowadzania i eliminacji kwantyfikatora ogólnego.

Przykład 13.9

W dowodzie dwukrotnie wykorzystuje się regułę eliminacji i wprowadzania kwantyfikatora ogólnego. Podczas eliminacji podstawienia tożsamościowe za zmienne x, y zachowują odpowiednie wymogi. Podobnie podczas wprowadzania kwantyfikatorów są zachowane odpowiednie wymogi, gdyż zmienne x, y nie mają wolnych wystąpień w wykorzystywanym założeniu.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\forall x \bullet \forall y \bullet p(x,y)]_1}{\forall y \bullet p(x,y)[x := x]}\,(\forall E)}{p(x,y)[y := y]}\,(\forall E)}{\forall x \bullet p(x,y)}\,(\forall I)}{\forall y \bullet \forall x \bullet p(x,y)}\,(\forall I)}{\forall x \bullet \forall y \bullet p(x,y) \Rightarrow \forall y \bullet \forall x \bullet p(x,y)}\,(\Rightarrow I_1)$$

Zdefiniowane dla rachunku zdań pojęcie zbioru derywacji i relacji derywacji w oczywisty sposób uogólnia się dla rachunku kwantyfikatorów. Podobnie jak dla systemu Hilberta, dla dowolnego zbioru formuł Φ i formuły α zachodzi twierdzenie o semantycznej niesprzeczności i semantycznej zupełności:

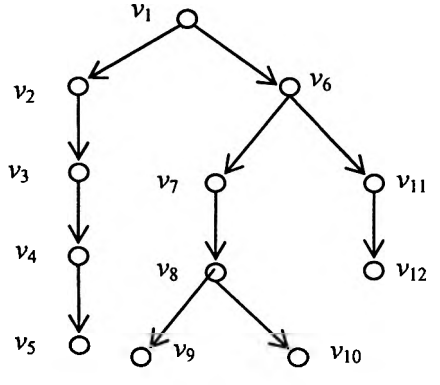
Twierdzenie 13.3

$\Phi \vdash_G \alpha$ wtedy i tylko wtedy, gdy $\Phi \models \alpha$.

13.4. Metoda tablic analitycznych

Metoda tablic analitycznych formalizuje dowodzenie nie wprost. Podobnie jak metoda rezolucji, stosuje unifikację termów, inaczej natomiast jest niż w metodzie rezolucji – podstawą badania są wprawdzie formuły zapisane w skolemowskiej postaci normalnej, ale matryca formuły musi być w dysjunkcyjnej postaci normalnej. Podobnie jak w metodzie sekwentów Gentzena dowód ma strukturę drzewa, jednak z uwagi na formę zapisu mówi się o tablicy – tablica jest formą zapisu drzewa, i również – jak w metodzie sekwentów Gentzena – tworzenie drzewa (tablicy) wiąże się z eliminacją spójników logicznych i kwantyfikatorów.

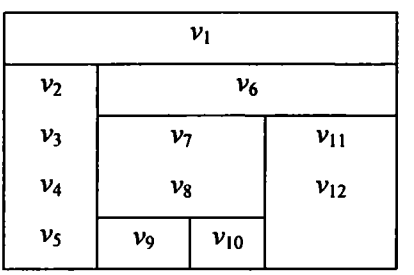
W celu wyjaśnienia struktury tablicy analitycznej rozpatrzmy drzewo dowodu na rysunku 13.1.



Rys. 13.1. Przykładowe drzewo dowodu

Reprezentacją tego drzewa jest tablica pokazana na rysunku 13.2. Związek pomiędzy drzewem a tablicą jest oczywisty.

- Korzeń drzewa zajmuje pozycję w komórce w najwyższym wierszu.
- Jeżeli wierzchołek ma rozgałęzienie, to komórka tablicy, która znajduje się poniżej kratki reprezentującej ten wierzchołek, jest podzielona na tyle części, ile wierzchołek ma rozgałęzień.
- Jeżeli wierzchołek nie ma rozgałęzienia, to jego następnik znajduje się w tej samej komórce i jego symbol jest zapisany poniżej symbolu danego wierzchołka.
- Liście drzewa znajdują się w różnych komórkach na dole tablicy. Są one końcami gałęzi drzewa, czyli ścieżek prowadzących od korzenia do liści.



Rys. 13.2. Struktura tablicy analitycznej odpowiadająca drzewu z rys. 13.1

Zawartością komórek tablicy są formuły. Komórka w górnym wierszu jest badaną formułą, a zawartość pozostałych komórek określa się kolejno, poczynając od góry tablicy, zgodnie z odpowiednim systemem dowodzenia.

Najpierw pokażemy metodę tablic analitycznych dla rachunku zdań. Badanie, czy formuła jest spełnialna, polega na dowodzeniu nie wprost, co oznacza, że zakłada się, iż badana formuła α nie jest spełnialna i próbuje się to pokazać, znajdując takie wartościowania zmiennych zdaniowych, przy których negacja formuły $\neg\alpha$ jest prawdziwa. Dla formuły $\neg\alpha$ konstruuje się drzewo dowodu – tablicę analityczną. Formuła $\neg\alpha$ ma być w dysjunkcyjnej postaci normalnej, czyli w postaci

$$\neg\alpha \equiv \delta_1 \vee \delta_2 \vee \ldots \vee \delta_n$$

gdzie δ_i, dla $i = 1, \ldots, n$, są koniunkcjami literałów.

Konstrukcja tablicy analitycznej polega na stosowaniu następujących reguł eliminacji spójników logicznych:

$$\frac{\neg\neg\alpha}{\alpha} \qquad \frac{\neg\mathbf{true}}{\mathbf{false}} \qquad \frac{\neg\mathbf{false}}{\mathbf{true}} \qquad \frac{\alpha \vee \beta}{\alpha \mid \beta} \qquad \frac{\alpha \wedge \beta}{\begin{array}{c}\alpha\\ \beta\end{array}}$$

Reguły określają, w jaki sposób na podstawie formuły-przesłanki, umieszczanej w danej komórce tablicy, określa się formuły-wnioski, umieszczane w bezpośrednio niższych komórkach.

Trzy pierwsze reguły są oczywiste.

Czwarta reguła oznacza, że jeśli w komórce jest formuła $\alpha \vee \beta$, to bezpośrednio pod nią znajdują się dwie komórki z zawartością α oraz β.

Piąta reguła oznacza, że jeśli w komórce jest formuła $\alpha \wedge \beta$, to bezpośrednio pod nią znajduje się jedna komórka, której zawartością jest, zapisana pionowo, lista formuł α oraz β.

Rozpatrzmy przykład ilustrujący zastosowanie tych reguł.

Przykład 13.10

Niech badaną formułą będzie następująca formuła rachunku zdań

$$\neg(a \vee b) \Rightarrow (\neg a \wedge \neg b)$$

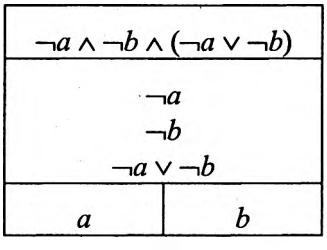
Negacja tej formuły

$$\neg(\neg(a \vee b) \Rightarrow (\neg a \wedge \neg b))$$

– po sprowadzeniu do dysjunkcyjnej postaci normalnej – ma postać

$$\neg a \wedge \neg b \wedge (\neg a \vee \neg b)$$

Tablica analityczna zbudowana dla tej formuły przedstawia się więc następująco:



Rys. 13.3. Tablica analityczna dla formuły $\neg a \wedge \neg b \wedge (\neg a \vee \neg b)$

Zawartość tablicy stanowi podstawę do stwierdzenia, czy wyjściowa formuła $\neg(a \vee b) \Rightarrow (\neg a \wedge \neg b)$ jest spełnialna. W tym celu ocenia się każdą gałąź tablicy. Gałąź tablicy reprezentuje koniunkcję formuł stanowiących zawartość komórek należących do gałęzi.

Gałąź tablicy uważa się za *zamkniętą* wtedy i tylko wtedy, gdy zawiera pewną formułę wraz z jej zaprzeczeniem. Całą tablicę uważa się za *zamkniętą* wtedy i tylko wtedy, gdy zamknięte są wszystkie jej gałęzie.

Przykładowa tablica jest zamknięta, gdyż zawiera dwie gałęzie, obie zamknięte. Oznacza to, że poszukiwanie wartościowania zmiennych, dla których formuła $\neg a \wedge \neg b \wedge (\neg a \vee \neg b)$ okazałaby się fałszywa, prowadzi do sprzeczności. Badana formuła jest zatem spełnialna, jest tautologią rachunku zdań.

Przykład 13.11

Niech badaną formułą rachunku zdań będzie

$$(a \Rightarrow b) \Rightarrow (b \Rightarrow a)$$

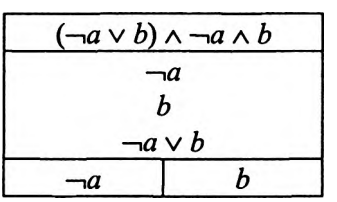
Negacja tej formuły

$$\neg((a \Rightarrow b) \Rightarrow (b \Rightarrow a))$$

– po sprowadzeniu do dysjunkcyjnej postaci normalnej – ma postać

$$(\neg a \vee b) \wedge \neg a \wedge b$$

Tablica analityczna zbudowana dla tej formuły (rys. 13.4)



Rys. 13.4. Tablica analityczna dla formuły $(\neg a \vee b) \wedge \neg a \wedge b$

nie jest zamknięta, gdyż otwarte są obie jej gałęzie, a zatem badana formuła nie jest spełnialna.

W przypadku rachunku kwantyfikatorów z kwantyfikatorem ogólnym wymaga się, aby rozważane formuły były w postaci skolemowskiej. Zbiór reguł podanych dla rachunku zdań rozszerza się o dwie nowe reguły. Pozwalają one na wydłużanie gałęzi i konkretyzację występujących w tablicy formuł.

Pierwsza reguła – reguła kwantyfikatora – pozwala na wydłużenie gałęzi, w której znajduje się formuła z kwantyfikatorem ogólnym $\forall x \bullet \alpha$, przez dołączenie formuły postaci $\alpha[x ::= y]$, gdzie zmienna y jest nie jest związana innym kwantyfikatorem w żadnej z formuł występujących w tej samej gałęzi, co formuła $\forall x \bullet \alpha$. Dołączenie nowej formuły symbolicznie przedstawia się w postaci

$$\frac{\forall x \bullet \alpha}{\alpha[x ::= y]}$$

Stosowanie tej reguły, podobnie jak w przypadku eliminacji kwantyfikatora po lewej stronie sekwentu, w istocie nie eliminuje kwantyfikatora, lecz generuje nowe formuły.

Z tego powodu wprowadza się ograniczenia na liczbę jej zastosowań przy praktycznej budowie tablicy. Wartość tego ograniczenia jest podyktowana względami pragmatycznymi. Dalej zakłada się, że ograniczenie to jest wyrażone pewną liczbą N. Sens tego ograniczenia wynika z podanego niżej algorytmu budowy tablic analitycznych. Podaną regułę można więc przedstawić w uogólnionej postaci:

$$\frac{\forall x \bullet \alpha}{\alpha[x := y_1]}$$

$$\ldots$$

$$\alpha[x := y_N]$$

Wyraża ona możliwość generacji wielu formuł z formuły $\forall x \bullet \alpha$. Zmienne $y_1, \ldots, y_N$ muszą oczywiście być zmiennymi, które nie są wiązane przez inne kwantyfikatory występujące w tej samej gałęzi, co formuła $\forall x \bullet \alpha$.

Druga reguła – reguła konkretyzacji – stwierdza, że: jeśli na danej gałęzi znajdują się dwa literały komplementarne α_1 i α_2, które dają się uzgodnić, a ich najbardziej ogólny unifikator $NOU(\alpha_1, \alpha_2) = \sigma$, to dopuszczalna jest zamiana każdej formuły β występującej w tablicy przez jej konkretyzację $\beta\,\sigma$.

Algorytm automatycznego wnioskowania metodą tablic analitycznych

Dane: formuła α.
Wynik: odpowiedź *tak*, gdy α jest spełnialna, oraz *nie* w przypadku przeciwnym.
Procedura:

1. Negację formuły α sprowadź do postaci skolemowskiej (algorytm z rozdziału 12.).
2. Matrycę otrzymanej formuły przedstaw w dysjunkcyjnej postaci normalnej (algorytm z rozdziału 9.).
3. Początkowa postać tablicy analitycznej T składa się z jednej komórki, której zawartością jest β – formuła otrzymana w wyniku czynności poprzednich punktów. Początkowa wartość zmiennej pomocniczej liczącej użycie reguły kwantyfikatora $n = 0$.
4. **while**
   tablica T nie jest zamknięta i można stosować do niej reguły eliminacji spójników logicznych, regułę kwantyfikatora i regułę konkretyzacji
   do
   a) stosuj reguły eliminacji spójników logicznych, regułę kwantyfikatora aż do uzyskania tablicy zamkniętej lub do chwili, gdy zmienna n nie przekroczy wartości N;
   b) stosuj regułę konkretyzacji, dopóki tablica T nie jest zamknięta;
   c) zwiększ o jeden wartość zmiennej n;
   od

5. Jeżeli została znaleziona tablica zamknięta – odpowiedz *tak*.
6. Jeżeli algorytm zatrzymał się ze względu na brak możliwości stosowania reguł – odpowiedz *nie*.

Poniżej podaje się bez dowodu twierdzenie gwarantujące poprawność i pełność algorytmu. Dyskusje na temat dowodu można znaleźć w książce [Szałas 1992], a pełne formalne uzasadnienie metody tablic analitycznych – w książce [Fitting 1990].

Twierdzenie 13.4

Przedstawiony algorytm implementuje metodę tablic analitycznych w tym sensie, że:

- daje odpowiedź *tak* wtedy i tylko wtedy, gdy formuła α jest tautologią,
- gdy formuła α nie jest tautologią, algorytm daje odpowiedź *nie* lub zapętla się.

Rozpatrzmy przykład zastosowania algorytmu.

Przykład 13.12

Dana jest formuła

$$(\forall x \bullet p(x) \vee q(x)) \Rightarrow (\exists x \bullet p(x) \vee \forall x \bullet q(x))$$

Jej negacją jest

$$\neg((\forall x \bullet p(x) \vee q(x)) \Rightarrow (\exists x \bullet p(x) \vee \forall x \bullet q(x)))$$

Po przekształceniach, polegających na sprowadzeniu do przedrostkowej postaci normalnej, otrzymuje się formułę

$$\forall x \bullet \forall y \bullet \exists z \bullet (p(x) \vee q(x)) \wedge \neg p(y) \wedge \neg q(z)$$

Po skolemizacji formuła przyjmuje postać

$$\forall x \bullet \forall y \bullet (p(x) \vee q(x)) \wedge \neg p(y) \wedge \neg q(f(x, y))$$

Tablica analityczna, budowana zgodnie z algorytmem, bez wykorzystania reguły konkretyzacji, ma postać przedstawioną na rysunku 13.5. Wiersz nr 2 uzyskano, stosując regułę kwantyfikatora, wynika z niego wiersz nr 3, po zastosowaniu reguły dotyczącej koniunkcji. W wierszu nr 4 następuje rozgałęzienie, którego podstawą jest formuła zawierająca dysjunkcję.

Lewą gałąź można zamknąć, stosując regułę konkretyzacji z podstawieniem $\sigma_1 = NOU(p(x_1), p(y_1))$, dla formuł $p(x_1)$ oraz $\neg p(y_1)$, które znajdują się na tej samej gałęzi.

Zamknięcie prawej gałęzi wymaga ponownego zastosowania reguły kwantyfikatora, w celu generacji formuły $(p(x_2) \vee q(x_2)) \wedge \neg p(y_2) \wedge \neg q(f(x_2, y_2))$, znajdującej się w wierszu nr 5. Stosując regułę związaną z koniunkcją, otrzymuje się dalsze formuły umieszczone w wierszu nr 6. Stosując tym razem regułę konkretyzacji dla formuł $q(x_1)$ – wiersz nr 4 – oraz $\neg q(f(x_2, y_2))$ – wiersz nr 6, z podstawieniem

$\sigma_2 = NOU(q(x_1), q(f(x_2, y_2)))$, zamyka się prawą gałąź tablicy, i – tym samym – zamyka się całą tablicę.

1	$\forall x \bullet \forall y \bullet (p(x) \vee q(x)) \wedge \neg p(y) \wedge \neg q(f(x, y))$	
2	$(p(x_1) \vee q(x_1)) \wedge \neg p(y_1) \wedge \neg q(f(x_1, y_1))$	
3	$\neg p(y_1)$ $\neg q(f(x_1, y_1))$ $p(x_1) \vee q(x_1)$	
4	$p(x_1)$	$q(x_1)$
5		$(p(x_2) \vee q(x_2)) \wedge \neg p(y_2) \wedge \neg q(f(x_2, y_2))$
6		$\neg p(y_2)$ $\neg q(f(x_2, y_2))$ $p(x_2) \vee q(x_2)$

Rys. 13.5. Tablica analityczna

13.5. Własności metalogiczne rachunku kwantyfikatorów

Logika klasyczna oparta na rachunku kwantyfikatorów jest scharakteryzowana przez język o dobrze zdefiniowanej składni i semantyce oraz przez system dowodzenia.

Systemy dowodzenia różnią się od siebie doborem aksjomatów i reguł oraz wynikającym z tego sposobem konstrukcji dowodów twierdzeń. Mają natomiast pewne wspólne własności. Przedstawione systemy dowodzenia są *semantycznie niesprzeczne* i *semantycznie zupełne*. Dodatkowo, co jest uznawane za szczególną własność logiki klasycznej, nie istnieją algorytmy dowodzenia oparte na tych systemach, które gwarantowałyby, że w skończonej liczbie kroków można rozstrzygać, czy dowolna formuła jest, czy nie jest tautologią. Rachunek logiki klasycznej, dokładniej rachunek kwantyfikatorów jest *nierozstrzygalny*, a ściślej, jest *częściowo nierozstrzygalny*. Oznacza to, że dla dowolnej formuły, jeżeli formuła jest tautologią, to istnieje algorytm, który w skończonej liczbie kroków zawsze to potwierdzi, natomiast w przypadku przeciwnym, gdy formuła nie jest tautologią, taki algorytm nie istnieje. Rozstrzygalne mogą być natomiast pewne fragmenty rachunku logicznego, na przykład rozstrzygalne są rachunek zdań, jednoargumentowy rachunek kwantyfikatorów.

Język logiki pozwala na budowanie teorii elementarnych (rozdział 10.), służących do opisu wybranego fragmentu interesującego świata. Język teorii elementarnej charakteryzuje się przyjęciem specyficznej sygnatury języka formalnego, to jest symboli funkcyjnych i symboli predykatów, oraz specyficznej interpretacji (bądź klasy interpretacji) tych symboli. Specyfika interpretacji wyraża się przez ustalenie zbioru formuł spełnialnych w tej interpretacji. Wyróżnione formuły nazywa się aksjomatami specyficznymi teorii. Formuły teorii służą do opisu specyficznych własności obiektów należących do wybranego fragmentu świata. System dowodzenia pozwala natomiast na dowodzenie tego, czy pewne formuły wyrażają, czy nie wyrażają własności zachodzących w wybranym fragmencie świata.

Okazuje się, co pokazał Gödel, że dostatecznie bogate teorie mają specyficzne własności, które wskazują na ograniczenia metody aksjomatycznej. Chodzi o teorie, które pozwalają zbudować arytmetykę liczb naturalnych, a więc o prawie wszystkie nietrywialne teorie mające praktyczne zastosowania.

Ograniczenie nazywane *niezupełnością teorii* – pierwsze twierdzenie Gödla – polega na tym, że istnieją w takiej teorii zdania spełnione, które nie są twierdzeniami teorii. Inaczej: dla teorii tej klasy nie istnieje semantycznie zupełny system dowodzenia.

Drugie ograniczenie odnosi się do *niesprzeczności* teorii. Niesprzeczność oznacza, że teoria nie zawiera takiej formuły α, że α oraz $\neg\alpha$ są twierdzeniami teorii. Z drugiego twierdzenia Gödla wynika, że dla teorii zawierających arytmetykę liczb naturalnych nie można podać takiego dowodu niesprzeczności, który korzystałby wyłącznie ze środków tej teorii. Inaczej: na gruncie danej teorii nie można podać dowodu jej niesprzeczności.

Twierdzenia Gödla mają znaczenie historyczne i filozoficzne. Znaczenie historyczne polega na tym, że został obalony, sformułowany na początku XX wieku, program Hilberta, którego myślą przewodnią było zbudowanie teorii sformalizowanej, obejmującej całą matematykę, i udowodnienie jej za pomocą prostych środków logicznych. Cała matematyka zawiera oczywiście arytmetykę liczb naturalnych, a zatem nie można udowodnić jej zupełności i niesprzeczności. Znaczenie filozoficzne bierze się stąd, że twierdzenia Gödla wskazują na ograniczoność podejścia aksjomatycznego. Pesymistyczna interpretacja tego faktu sprowadza się do stwierdzenia, że istnieją „nieprzekraczalne granice rozumu ludzkiego", podczas gdy interpretacja optymistyczna wskazuje właśnie na przewagę rozumowania umysłu ludzkiego nad wnioskowaniem prowadzonym w ramach systemów sformalizowanych. Interpretacja twierdzeń Gödla na gruncie informatyki wskazuje na ograniczoność tego, co można policzyć za pomocą komputera, gdyż wszystko to, co może wykonać komputer, da się wyrazić tylko w pewnej teorii elementarnej. Wynika też z tego pogląd, że komputery nie będą w stanie całkowicie zastąpić człowieka w podejmowanych przez niego rozumowaniach i decyzjach.

Ćwiczenia

1. Korzystając z systemu dowodzenia Hilberta, dowieść, że następujące formuły są twierdzeniami:
   a) $\alpha \vee \neg\alpha$
   b) $(\alpha \Rightarrow \neg\alpha) \Rightarrow \neg\alpha$
   c) $\forall x \bullet \forall y \bullet p(x, y) \Leftrightarrow \forall y \bullet \forall x \bullet p(x, y)$
   d) $(\exists x \bullet p(x) \vee q(x)) \Leftrightarrow (\exists x \bullet p(x)) \vee (\exists x \bullet q(x))$

2. Wykorzystując system dedukcji naturalnej Gentzena, pokazać, że tautologiami są formuły:
   a) $(\alpha \Rightarrow \beta) \Rightarrow ((\beta \Rightarrow \gamma) \Rightarrow (\neg\gamma \Rightarrow \neg\alpha))$
   b) $(\neg\alpha \Rightarrow \alpha) \Rightarrow \alpha$
   c) $\alpha \vee \neg\alpha$
   d) $(\alpha \Rightarrow \gamma) \Rightarrow ((\beta \Rightarrow \gamma) \Rightarrow (\alpha \vee \beta \Rightarrow \gamma))$
   e) $(\forall x \bullet p(x) \Leftrightarrow q(x)) \Rightarrow ((\forall x \bullet p(x)) \Leftrightarrow (\forall x \bullet q(x)))$
   f) $(\forall x \bullet p(x) \Leftrightarrow q(x)) \Rightarrow ((\exists x \bullet p(x)) \Leftrightarrow (\exists x \bullet q(x)))$
   g) $\exists x \bullet \forall y \bullet p(x, y) \Rightarrow \forall y \bullet \exists x \bullet p(x, y)$

3. Uzupełnić system dedukcji naturalnej przez wyprowadzenie reguł eliminacji i wprowadzania spójników:
   a) dysjunkcji,
   b) równoważności,
   c) *NOR*,
   d) *NAND*.

4. Wykorzystując metodę tablic analitycznych, pokazać, które formuły są tautologiami:
   a) $\exists x \bullet \forall y \bullet p(x, y) \Rightarrow \forall y \bullet \exists x \bullet p(x, y)$
   b) $\forall x \bullet \exists y \bullet p(x, y) \Rightarrow \exists y \bullet \forall x \bullet p(x, y)$
   c) $\exists x \bullet (q(x) \Rightarrow \forall x \bullet q(x))$
   d) $\forall x \bullet \exists y \bullet \forall z \bullet \exists w \bullet (p(x, y) \vee \neg p(x, y))$
   e) $\forall x \bullet (q(x) \Rightarrow \alpha) \Rightarrow \exists x \bullet (q(x) \Rightarrow \alpha)$

14. Inne logiki

14.1. Wstęp

Omawiana w poprzednich rozdziałach logika klasyczna jest jądrem wszelkich logik. Jej rozwój w XX wieku wynikał głównie z potrzeby rozwiązywania problemów z zakresu podstaw matematyki. Ogólniej można stwierdzić, że motywacje tworzenia nowych logik były podyktowane – po pierwsze – chęcią ogarnięcia możliwie szerokiej klasy wypowiedzi spotykanych nie tylko w języku matematyki, ale i w języku naturalnym, i – po drugie – identyfikacją oraz ujęciem w formalne ramy sposobów wnioskowania stosowanych przez ludzi. Wraz z poszerzaniem się zastosowań informatyki powstały nowe inspiracje do rozwoju logiki. Wynikają one na przykład z zastosowań systemów ekspertowych lub rozwoju lingwistyki matematycznej i związanej z nią konstrukcją systemów automatycznego tłumaczenia języków naturalnych.

W tym rozdziale przedstawia się podstawowe informacje tylko o niektórych logikach nieklasycznych – logikach wielowartościowych i modalnych [Bolc, Borodziewicz, Wójcik 1991], [Gabbay 1998]. Logiki modalne stanowią bardzo szeroką grupę logik. Za ich szczególne przypadki można uważać, omawiane w dalszej części rozdziału, logiki temporalne [Gabbay 1998], [Klimek 1999], a także – do pewnego stopnia – logiki intuicjonistyczne [Gabbay 1998].

Krótko wspomina się też o logikach niemonotonicznych [Gabbay 1998]. Omawia się tylko przesłanki stanowiące inspirację ich powstawania. Logiki te, obecnie intensywnie rozwijane, mają bezpośredni związek z zastosowaniami – z bazami wiedzy i systemami ekspertowymi. Charakterystycznym wyróżnikiem dla tych logik jest to, że proponują one pewne sposoby wnioskowania w sytuacji posiadania niepełnej lub niepewnej informacji.

Przeglądem nie są objęte wszystkie gałęzie logiki. Nie omawia się na przykład logik relewantnych, które próbują osłabić ograniczenie logiki klasycznej, polegające na tym, że ocena prawdziwości zdań złożonych zależy tylko od prawdziwości ich części składowych (własność ekstensjonalności – zob. rozdział 1.), pomija się natomiast zupełnie treści wyrażane przez te składowe, a właśnie uwzględnienie związków treściowych jest szczególnie ważne w systemach ekspertowych.

Obszerną, choć niewyczerpującą prezentację różnych logik zawierają pozycje encyklopedyczne [Marciszewski 1987, 1988].

Z podziałem nauk na ścisłe i empiryczne wiąże się podział metod wnioskowania na dedukcyjne i indukcyjne. Podstawą nauk empirycznych są obserwacje interesujących zjawisk i procesów. Obserwacje te często dostarczają informacji cząstkowych, zwykle obarczonych błędami pomiarów. Wnioskowania oparte na takich danych prowadzą więc do niepewnych lub niepełnych wniosków. Dodatkowo, nie zawsze z góry wiadomo, jak takie wnioskowanie prowadzić. Prowadząc na przykład po raz pierwszy pewien eksperyment, nie zawsze wiadomo, jakich można się spodziewać następstw. W metodologii nauk empirycznych rozważa się specyficzne rodzaje logik, między innymi tak zwane *logiki indukcji* [Mortimer 1982]. Ogólne zamierzenie logik indukcji wiąże się ze sposobem uzyskiwania na podstawie danych eksperymentalnych możliwie najlepszej teorii, która tłumaczyłaby związki pomiędzy obserwowanymi faktami, a także – jeszcze lepiej – pozwalałaby na przewidywanie dotychczas nieobserwowanych faktów. Taki ogólny mechanizm oczywiście nie istnieje. Możliwe jest natomiast porównywanie różnych konkretnych mechanizmów i ocenianie stopnia ich wiarogodności. Logiki indukcji nie należy utożsamiać z używanym wcześniej pojęciem indukcji matematycznej czy strukturalnej.

14.2. Logiki wielowartościowe

Logiki wielowartościowe mają początek w latach dwudziestych XX wieku, kiedy Łukasiewicz jako pierwszy przedstawił propozycję logiki trójwartościowej. Prace nad logikami wielowartościowymi podejmowali między innymi Post, Sobociński, Słupecki. Łukasiewicz opisał całą rodzinę skończenie wielowartościowych logik L_n dla $n = 3, 4, ...$, oraz jedną nieskończenie wielowartościową logikę $L_{\aleph_0}$. Zbiorem wartości logicznych logiki L_n jest zbiór

$$A_n =_{\text{def}} \{0, 1/(n-1), ..., (n-2)/(n-1), 1\} \quad \text{dla } n = 3, 4, ...$$

Poniżej przedstawia się tylko rachunek zdań w logice L_3. W tej logice interpretacja znanych spójników logiki klasycznej: $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$, $\neg$ jest wyrażona tablicą 14.1.

Tablica 14.1

b	$a \Rightarrow b$			$a \wedge b$			$a \vee b$			$a \Leftrightarrow b$			$\neg a$
a	0	½	1	0	½	1	0	½	1	0	½	1	
0	1	1	1	0	0	0	0	½	1	1	½	0	1
½	½	1	1	0	½	½	½	½	1	½	1	½	½
1	0	½	1	0	½	1	1	1	1	0	½	1	0

Opracowanie logiki L_3 wiązało się z nadawaniem wartości logicznej zdaniom odnoszącym się do przyszłości, wartość ½ oznaczała brak wiedzy.

Zdania o przyszłości mogą wyrażać fakty, które zajdą lub nie zajdą, na przykład:

W 2100 roku ludzie będą mieszkać na Marsie.

Zdanie takie wypowiadane w obecnej chwili nie jest ani prawdziwe, ani fałszywe, nadaje się mu więc wartość ½, co wyraża naszą niewiedzę o przyszłości. Wartość ½ może być także interpretowana inaczej jako: *niezdefiniowane*, *nieokreślone* albo jako *brak danych*.

Pierwsza aksjomatyka Łukasiewicza była oparta na spójnikach implikacji i negacji. Inne znane spójniki – koniunkcji, dysjunkcji i równoważności – były definiowane przez implikację i negację, tak samo jak w logice klasycznej.

Zestaw spójników logicznych, złożony z implikacji, negacji, uzupełniony stałą logiczną ½, jest systemem funkcjonalnie pełnym, tzn. za ich pomocą można wyrazić dowolne inne spójniki logiczne w L_3.

Aksjomatyka rachunku zdań trójelementowej logiki Łukasiewicza (opracowana przez Łukasiewicza, Tarskiego i Wajsberga) składa się z następujących *aksjomatów*:

a) $q \Rightarrow (p \Rightarrow q)$
b) $(p \Rightarrow q) \Rightarrow ((q \Rightarrow r) \Rightarrow (p \Rightarrow r))$
c) $((p \Rightarrow \neg p) \Rightarrow p) \Rightarrow p$
d) $((\neg q \Rightarrow \neg p) \Rightarrow (p \Rightarrow q))$
e) $((p \Rightarrow q) \Rightarrow q) \Rightarrow ((q \Rightarrow p) \Rightarrow p)$
f) $((p \Rightarrow q) \Rightarrow (q \Rightarrow p)) \Rightarrow (q \Rightarrow p)$

oraz z dwóch reguł:

Reguły odrywania: z formuł α oraz $\alpha \Rightarrow \beta$ wnioskujemy β, czyli

$$\frac{\alpha, \alpha \Rightarrow \beta}{\beta}$$

Reguły podstawiania: z formuły α, w której występuje zmienna zdaniowa a, wnioskujemy to, co otrzymamy w rezultacie podstawienia dowolnej formuły β za każde wystąpienie zmiennej a, czyli

$$\frac{\alpha}{\alpha[a ::= \beta]}$$

Przedstawiony zestaw aksjomatów nie jest minimalny, gdyż ostatni aksjomat jest zależny od aksjomatów poprzednich. System ten jest semantycznie niesprzeczny i semantycznie zupełny.

Oprócz omówionych, do logik wielowartościowych można zaliczyć również między innymi logiki prawdopodobieństwa i logiki rozmyte.

14.3. Logiki modalne

Niech będą dane trzy zdania:

Książka leży na stole. (1)
Książka leży na podłodze. (2)
Książka nieruchomo (bez podparcia) utrzymuje się w powietrzu. (3)

Jeśli założyć, że wypowiedzi te odnoszą się do sytuacji w jakimś pomieszczeniu na Ziemi, to zdania (1) oraz (2) mogą być prawdziwe lub fałszywe, natomiast zdanie (3) będzie zawsze fałszywe. Nie jest bowiem możliwe w żadnym pomieszczeniu ziemskim, aby książka zajmowała trwale nieruchome położenie. Rozróżnienie między sytuacjami związanymi ze zdaniami (1) i (2) a zdaniem (3) stanie się bardziej wyraźne, gdy rozpatruje się pewne ich modyfikacje:

Możliwe, że książka leży na stole. (4)
Możliwe, że książka leży na podłodze. (5)
Możliwe, że książka nieruchomo (bez podparcia) utrzymuje się w powietrzu. (6)

Zdania (4) i (5) są oczywiście prawdziwe, a zdanie (6) jest fałszywe. Przytoczone oceny prawdziwości zdań odnoszą się do zjawisk w bezpośrednio otaczającym nas świecie. Te same zdania odniesione do zjawisk zachodzących na przykład w świecie obserwowanym przez kosmonautów w pojeździe kosmicznym będą miały inne oceny, zwłaszcza zdanie (6) stanie się prawdziwe. Warto też zwrócić uwagę na to, że nawet osoba przebywająca na powierzchni Ziemi byłaby gotowa uznać prawdziwość zdania (6), gdyby tylko wiedziała, że loty kosmiczne są osiągalne.

Zdania są przykładami tak zwanych wypowiedzi *modalnych* – występujący w nich zwrot – *możliwe jest, że* α – jest przykładem operatora modalnego. Symbolicznie jest on zapisywany $\Diamond\alpha$. Wypowiedź dualna: *konieczne jest, że* α, jest symbolicznie zapisywana $\Box\alpha$. Takie zwroty spotyka się często w wypowiedziach formułowanych w języku naturalnym. Pomiędzy oboma zwrotami zachodzi związek semantyczny

$$\Box\alpha = \neg\Diamond\neg\alpha$$

Symbole $\Box$ oraz $\Diamond$ są traktowane jako jednoargumentowe operatory logiczne.

Uwaga

Logiki modalne korzeniami sięgają czasów Arystotelesa. Nowożytne badania podjął na początku XX wieku C.I. Lewis, który pojęcie możliwości wykorzystał w celu rozróżnienia między implikacją materialną a implikacją ścisłą. *Implikacja materialna*, zdefiniowana w klasycznym rachunku zdań, ma ułomność (zob. rozdział 1.), która na podstawie fałszywej przesłanki pozwala na wyprowadzenie dowolnego wniosku. Wady tej nie ma *implikacja ścisła* (symbol $\Rightarrow$) zdefiniowana przez Lewisa jako

$$p \Rightarrow q =_{\text{def}} \neg\Diamond(p \wedge \neg q)$$

czyli, że q wynika ściśle z p wtedy i tylko wtedy, gdy nie jest możliwe, by jednocześnie prawdziwe było p i fałszywe q. Definicja ta odróżnia implikację ścisłą od materialnej, której definicją jest

$$p \Rightarrow q =_{\text{def}} \neg(p \wedge \neg q).$$

Implikacja ścisła usuwa niektóre paradoksy implikacji materialnej, ale nadal pozostawia prawdziwe formuły, które do takich paradoksów się zalicza, na przykład:

$$p \Rightarrow (p \Rightarrow q) \quad (p \Rightarrow p) \Rightarrow (q \Rightarrow q) \quad (p \wedge \neg p) \Rightarrow q \quad (p \vee \neg p) \Rightarrow q$$

Z punktu widzenia składni, logiki modalne są rozszerzeniem języka formalnego logiki klasycznej. W definiowaniu semantyki logik modalnych przyjmuje się powszechnie podejście S. Kripkego, oparte na pojęciu zbioru możliwych stanów (lub światów). Podejście to przedstawia się poniżej, na przykładzie zdaniowej logiki modalnej.

Alfabet modalnego języka rachunku zdań składa się z następujących jednostek *leksykalnych*:

- symboli *stałych logicznych* reprezentowanych przez napisy **true** oraz **false**;
- przeliczalnej liczby symboli *zmiennych zdaniowych*,
- symboli *spójników logicznych* klasycznego rachunku zdań: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$,
- symboli *spójników logicznych modalnych*: $\Box$,$\Diamond$,
- symboli *nawiasów*: (,).

Zbiór *formuł* modalnego rachunku zdań *FORM* jest definiowany rekursywnie:

- symbole zmiennych zdaniowych oraz symbole stałych logicznych są formułami *elementarnymi*; zbiór zmiennych zdaniowych będzie oznaczony symbolem V;
- jeżeli α oraz β są formułami, to formułami *złożonymi* są napisy:

$$\neg\alpha, (\alpha \Rightarrow \beta), (\alpha \wedge \beta), (\alpha \vee \beta), (\alpha \Leftrightarrow \beta), \quad \Box\alpha, \Diamond\alpha.$$

Semantyka języka jest określana w strukturze Kripkego.

Definicja 14.1

Modelem Kripkego nazywa się trójkę $K = <S, \rho, v>$, gdzie: S jest dowolnym zbiorem nazywanym zbiorem *stanów* (lub *światów*), $\rho \subseteq S^2$ jest relacją binarną nazywaną relacją *osiągalności* stanów (światów), $v : V \times S \rightarrow$ *Logiczne* jest funkcją *wartościującą* zmienne zdaniowe w każdym ze stanów.

Jeżeli $<s, s'> \in \rho$, to stan s' nazywa się stanem osiągalnym ze stanu s.

Dziedziną interpretacji formuł jest, tak samo jak w klasycznym rachunku zdań, zbiór wartości *Logiczne*. Semantyka modalnego rachunku zdań zachowuje interpretację klasycznych spójników logicznych. Funkcja wartościowania v jest uogólnieniem odpowiedniej funkcji wartościującej v, która była wprowadzona przy definiowaniu semantyki

klasycznego rachunku zdań. Różnica polega na tym, że w modalnym rachunku zdań wartościowanie zmiennej zdaniowej zależy dodatkowo od stanu. W różnych stanach wartościowania tej samej zmiennej mogą być różne, podczas gdy w klasycznym rachunku zdań wartościowanie zmiennej jest tylko jedno – inaczej: w klasycznym rachunku zdań ma się do czynienia tylko z jednym stanem.

Niech $\alpha \in FORM$ będzie dowolną formułą oraz $s \in S$ będzie dowolnym stanem. Interpretacja formuły α przy wartościowaniu v w stanie s, zapisywana $INT_{v,s}(\alpha)$, jest definiowana rekursywnie względem struktury składniowej:

a) $INT_{v,s}(p) =_{def} v(p, s)$, dla zmiennej zdaniowej p
b) $INT_{v,s}(\mathbf{true}) =_{def} \mathbf{P}$
c) $INT_{v,s}(\mathbf{false}) =_{def} \mathbf{F}$
d) $INT_{v,s}(\neg\alpha) =_{def} \neg INT_{v,s}(\alpha)$
e) $INT_{v,s}(\alpha \circ \beta) =_{def} INT_{v,s}(\alpha) \circ INT_{v,s}(\beta)$, dla spójnika binarnego $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$
f) $INT_{v,s}(\Box\alpha) =_{def} \mathbf{P}$ wtedy i tylko wtedy, gdy dla dowolnego stanu s' osiągalnego ze stanu s zachodzi $INT_{v,s'}(\alpha) = \mathbf{P}$
g) $INT_{v,s}(\Diamond\alpha) =_{def} \mathbf{P}$ wtedy i tylko wtedy, gdy istnieje stan s' osiągalny ze stanu s, dla którego zachodzi $INT_{v,s'}(\alpha) = \mathbf{P}$

Przedstawiona wyżej semantyka, w zależności od konkretnych zastosowań, może być jeszcze zawężana przez narzucenie dodatkowych postulatów. Mogą one mieć postać formuł-aksjomatów, które powinny być spełniane w języku. W zależności od zestawu takich aksjomatów wyróżnia się różne rodzaje logik modalnych. Przykładami takich formuł są:

$\Box(\alpha \Rightarrow \beta) \Rightarrow (\Box\alpha \Rightarrow \Box\beta)$

$\Box\alpha \Rightarrow \Diamond\alpha$

$\Box\alpha \Rightarrow \alpha$

$\Box\alpha \Rightarrow \Box\Box\alpha$

$\Diamond\alpha \Rightarrow \Box\Diamond\alpha$

$\Diamond\Box\alpha \Rightarrow \Box\Diamond\alpha$

$\Box\Diamond\alpha \Rightarrow \Diamond\Box\alpha$

Omawiane wyżej modalności określa się mianem modalności *aletycznych*. Modalności mogą mieć także inne interpretacje. W zależności od przyjętej interpretacji, modalności czyta się w różny sposób i ma się do czynienia z różnymi rodzajami logik modalnych.

Na przykład pojęciem centralnym logiki *deontycznej* jest pojęcie obowiązku, formuły $\Box\alpha$ oraz $\Diamond\alpha$ odczytuje się jako: *jest obowiązkowe to, że* α oraz *jest dozwolone to, że* α.

Logika *epistemiczna* odnosi się do aktów lub stanów poznawczych, operuje pojęciami takimi, jak widzieć, wierzyć, uznawać, dlatego formuły $\Box\alpha$ oraz $\Diamond\alpha$ odczytuje się jako: *jest wiarygodne to, że* α oraz *jest niewiarygodne to, że* α.

W logice *temporalnej* przedmiotem zainteresowania są wypowiedzi, które uwzględniają związki czasowe – formuły $\Box\alpha$ oraz $\Diamond\alpha$ czyta się jako: *zawsze zachodzi* α oraz *czasem zachodzi* α.

Obszerniejsze omówienie logik modalnych i ich związków z logiką klasyczną zawiera książka [Szałas 1992].

14.4. Logiki temporalne

Przedmiotem logik temporalnych są wypowiedzi, które uwzględniają czas. Tłem, na którym rozpatruje się wypowiedzi, jest struktura czasowa. Zbiór stanów S w modelu Kripkego $K = <S, \rho, v>$ jest tu interpretowany jako zbiór chwil czasowych – oznaczany T, a relacja osiągalności ρ jest interpretowana jako uporządkowanie chwil w sensie chwila wcześniejsza–późniejsza – oznaczana $\preccurlyeq$.

Strukturą czasową nazywa się parę $SC = <T, \preccurlyeq>$, gdzie $\preccurlyeq \subseteq T^2$ jest relacją porządku.

W zależności od ustaleń dotyczących struktury czasowej otrzymuje się różne rodzaje logik temporalnych.

Jeżeli $\preccurlyeq$ jest relacją porządku częściowego (to znaczy jest zwrotna, antysymetryczna i przechodnia), to mamy do czynienia ze strukturą czasu *rozgałęzionego*, a jeśli jest relacją porządku liniowego (to znaczy jest zwrotna, antysymetryczna, przechodnia i spójna), to mamy do czynienia ze strukturą czasu *liniowego*.

Strukturę czasową nazywa się *ciągłą*, gdy

$$\forall t_1 \in T\bullet\ \forall t_2 \in T\bullet\ \exists t_3 \in T\bullet\ (t_1 \preccurlyeq t_2 \Rightarrow t_1 \preccurlyeq t_3 \wedge t_3 \preccurlyeq t_2)$$

dyskretną prawostronnie, gdy

$$\forall t_1 \in T\bullet \forall t_2 \in T\bullet((t_1 \preccurlyeq t_2 \wedge t_1 \neq t_2) \Rightarrow$$
$$(\exists t_3 \in T\bullet(t_1 \preccurlyeq t_3 \wedge t_1 \neq t_3) \wedge \neg\exists t_4 \in T\bullet(t_1 \preccurlyeq t_4 \wedge t_4 \preccurlyeq t_3 \wedge t_3 \neq t_4))$$

dyskretną lewostronnie, gdy

$$\forall t_1 \in T\bullet \forall t_2 \in T\bullet(t_1 \preccurlyeq t_2 \wedge t_1 \neq t_2) \Rightarrow$$
$$(\exists t_3 \in T\bullet(t_3 \preccurlyeq t_2 \wedge t_3 \neq t_2) \wedge \neg\exists t_4 \in T\bullet(t_4 \preccurlyeq t_2 \wedge t_3 \preccurlyeq t_4 \wedge t_3 \neq t_4))$$

Przykładem zbioru, na którym można zbudować strukturę ciągłą, jest zbiór liczb wymiernych, a dyskretną – zbiór liczb naturalnych.

W dalszych rozważaniach zakłada się dyskretną (lewo- i prawostronnie) strukturę czasu liniowego. Dla ustalenia uwagi przyjmuje się, że zbiór chwil jest zbiorem liczb naturalnych, a relacja osiągalności jest relacją $\leq$ w zbiorze liczb naturalnych, $n \leq m$ oznacza: chwila n nie jest późniejsza od chwili m. Struktura czasowa jest zatem parą $<Nat, \leq>$.

Zostanie przedstawiony rachunek zdań liniowej logiki temporalnej PLTL (*Propositional Linear Temporal Logic*). Logika jest logiką czasu przyszłego, co oznacza, że formuły wyrażają pewne własności, które odnoszą się do przyszłości, poczynając od ustalonej chwili odniesienia. Została ona opracowana przez Mannę i Pnueliego na początku lat osiemdziesiątych [Manna, Pnueli 1992, 1995], z przeznaczeniem do specyfikacji i weryfikacji własności programów.

Składnia logiki PLTL różni się od składni modalnego rachunku zdań przedstawionego w poprzednim podrozdziale tylko tym, że wprowadza dwa dodatkowe spójniki modalne: jednoargumentowy operator *next* i dwuargumentowy *until*. Spójniki te służą do wyrażania pewnych własności, które można także wyrazić za pomocą pozostałych spójników.

Zbiór formuł *FORM* logiki PLTL jest definiowany rekursywnie:

- symbole zmiennych zdaniowych oraz stałych logicznych są formułami,
- jeżeli α oraz β są formułami, to formułami są również:

  $\neg\alpha$, $(\alpha \Rightarrow \beta)$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \Leftrightarrow \beta)$, $\Box\alpha$, $\Diamond\alpha$, *next* α, $(\alpha$ *until* $\beta)$.

Interpretacja formuły temporalnej jest definiowana – tak samo jak w przypadku modalnego rachunku zdań – względem struktury czasowej $<Nat, \leq>$ i wartościowania v. Modelem dla formuł temporalnych jest więc trójka $<Nat, \leq, v>$.

Niech $\alpha \in FORM$ będzie dowolną formułą oraz $n \in Nat$ będzie dowolną chwilą. Interpretacja formuły α przy wartościowaniu v w chwili n, zapisywana $INT_{v,n}(\alpha)$, jest definiowana rekursywnie względem struktury składniowej:

a) $INT_{v,n}(p) =_{def} v(p, n)$, dla zmiennej zdaniowej p
b) $INT_{v,n}(\mathbf{true}) =_{def} \mathbf{P}$
c) $INT_{v,n}(\mathbf{false}) =_{def} \mathbf{F}$
d) $INT_{v,n}(\neg\alpha) =_{def} \neg INT_{v,n}(\alpha)$
e) $INT_{v,n}(\alpha \circ \beta) =_{def} INT_{v,n}(\alpha) \circ INT_{v,n}(\beta)$, gdzie $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$
f) $INT_{v,n}(\Box\alpha) =_{def} \mathbf{P}$ wtedy i tylko wtedy, gdy dla dowolnej chwili m takiej, że $n \leq m$ zachodzi $INT_{v,m}(\alpha) = \mathbf{P}$
g) $INT_{v,n}(\Diamond\alpha) =_{def} \mathbf{P}$ wtedy i tylko wtedy, gdy istnieje chwila m taka, że $n \leq m$, dla której zachodzi $INT_{v,m}(\alpha) = \mathbf{P}$
h) $INT_{v,n}(next\alpha) =_{def} \mathbf{P}$ wtedy i tylko wtedy, gdy dla chwili $n+1$ zachodzi $INT_{v,n+1}(\alpha) = \mathbf{P}$

i) $INT_{v,n}(\alpha \textit{ until } \beta) =_{def}$ **P** wtedy i tylko wtedy, gdy istnieje chwila $j \geq 0$ taka, że zachodzi $INT_{v,j}(\beta) =$ **P** oraz dla każdej chwili $i < j$ zachodzi $INT_{v,j}(\alpha) =$ **P**.

Formuła α, której interpretacja $INT_{v,n}(\alpha) =$ **P**, dla dowolnego wartościowania v i dowolnej chwili n, jest tautologią logiki PLTL. Formułę taką nazywa się też prawem logiki.

Warto zwrócić uwagę na spójnik *until*. Za jego pomocą można byłoby wyrazić spójniki $\Box$ oraz $\Diamond$, mianowicie:

$\Box\alpha =_{def}$ **true** *until* α

$\Diamond\alpha =_{def} \alpha$ *until* **false**

Formuły logiki PLTL pozwalają na zwarte wyrażenie złożonych własności, na przykład:

$\Box\Diamond\alpha$ formuła czytana: *zawsze możliwe* α, wyraża własność, że kiedykolwiek w przyszłości formuła α stanie się fałszywa, to jest pewne, że kiedyś w dalszej przyszłości stanie się znowu prawdziwa.

$\Diamond\Box\alpha$ formuła czytana: *kiedyś koniecznie* α, wyraża własność, że w przyszłości istnieje taka chwila, od której formuła α będzie prawdziwa.

Oto przykłady niektórych kategorii praw:

Prawa dualizmu

$\Box\neg\alpha \Leftrightarrow \neg\Diamond\alpha$

$\Diamond\neg\alpha \Leftrightarrow \neg\Box\alpha$

$next\ \neg\alpha \Leftrightarrow \neg\ next\alpha$

Prawa introspektywności

$\Box\alpha \Rightarrow \alpha$

$\alpha \Rightarrow \Diamond\alpha$

$(\alpha\ until\ \beta) \Rightarrow (\alpha \vee \beta)$

$\beta \Rightarrow (\alpha\ until\ \beta)$

Prawa idempotencji

$\Box\Box\alpha \Leftrightarrow \Box\alpha$

$\Diamond\Diamond\alpha \Leftrightarrow \Diamond\alpha$

Prawa rozdzielności

$\Box(\alpha \wedge \beta) \Leftrightarrow (\Box\alpha) \wedge (\Box\beta)$

$\Diamond(\alpha \vee \beta) \Leftrightarrow (\Diamond\alpha) \vee (\Diamond\beta)$

Prawa przemienności

$(\Box \; next\; \alpha) \Leftrightarrow (next\; \Box\alpha)$

$(\Diamond \; next\; \alpha) \Leftrightarrow (next\; \Diamond\alpha)$

$next\; (\alpha\; until\; \beta) \Leftrightarrow (next\; \alpha)\; until\; (next\; \beta)$

Prawa dołączania

$(\Box\alpha \wedge \Diamond\beta) \Rightarrow \Diamond(\alpha \wedge \beta))$

$(\Box\alpha \wedge next\; \beta) \Rightarrow next\; (\alpha \wedge \beta)$

$(\Box\alpha \wedge (\beta\; until\; \gamma)) \Rightarrow (\alpha \wedge \beta)\; until\; (\alpha \wedge \gamma)$

Dla logik temporalnych były opracowane różne systemy aksjomatyzacji. Jeden z pierwszych przykładów aksjomatyzacji logiki PLTL pochodzi z książki [Gabbay 1998] i składa się z następujących *aksjomatów*:

a) $\Box(\alpha \Rightarrow \beta) \Rightarrow (\Box\alpha \Rightarrow \Box\beta)$

b) $next \neg\alpha \Leftrightarrow \neg\; next\alpha$

c) $next\; (\alpha \Rightarrow \beta) \Rightarrow (next\; \alpha \Rightarrow next\; \beta)$

d) $\Box\alpha \Rightarrow (next\; \alpha \wedge next\; \Box\alpha)$

e) $\Box(\alpha \Rightarrow next\; \alpha) \Rightarrow (next\; \alpha \Rightarrow \Box\alpha)$

f) $(\alpha\; until\; \beta) \Rightarrow \Diamond\beta$

g) $(\alpha\; until\; \beta) \Leftrightarrow (next\; \beta \vee (next\alpha \wedge next\; (\alpha\; until\; \beta)))$

oraz *reguł* odrywania, podstawiania i generalizacji:

Reguła generalizacji: z formuły α wnioskuje się, że $\Box\alpha$, czyli

$$\frac{\alpha}{\Box\alpha}$$

System ten jest semantycznie niesprzeczny i semantycznie zupełny.

14.5. Logiki intuicjonistyczne

Składnia logiki intuicjonistycznej jest taka sama jak logiki klasycznej. Różnica wynika ze sposobu podejścia do oceny prawdziwości zdań. W logice intuicjonistycznej podejście to opiera się na specyficznej interpretacji spójników logicznych i kwantyfikatorów podanej przez Heytinga[25] – jednego z twórców tej logiki, który przedstawił pierwszy system aksjomatyczny dla intuicjonistycznego rachunku zdań. Jej pod-

[25] A. Heyting (1898–1980).

stawą jest intuicja, że stwierdzić prawdziwość zdania to tyle, co mieć dowód dla tego zdania.

Logika intuicjonistyczna jest jedną z logik konstruktywnych. Ilustracją różnic w stosunku do logiki klasycznej jest dowód twierdzenia:

Istnieją dwie liczby niewymierne a i b takie, że a^b jest wymierne.

Dowód twierdzenia jest niekonstruktywny: albo $(\sqrt{2})^{\sqrt{2}}$ jest wymierne i wtedy $a = b = \sqrt{2}$, albo $(\sqrt{2})^{\sqrt{2}}$ jest niewymierne i wtedy $a = (\sqrt{2})^{\sqrt{2}}$, $b = \sqrt{2}$. Z dowodu wynika, że liczby istnieją, ale oczywiście nie wiadomo, jakie są to liczby.

W interpretacji Heytinga prawdziwość formuł w logice intuicjonistycznej jest rozumiana w sposób następujący:

- Prawdziwość formuły $a \wedge b$ oznacza fakt posiadania dowodu d_a dla formuły a oraz dowodu d_b dla formuły b. Dowód formuły $a \wedge b$ jest zatem parą $<d_a, d_b>$.
- Dowód formuły $a \vee b$ to konstrukcja, która wybiera jedną z dwóch formuł i daje dowód wybranej formuły.
- Dowód formuły $a \Rightarrow b$ to konstrukcja, która każdemu dowodowi d_a formuły a przyporządkowuje dowód $d_b(d_a)$ formuły b.
- Dowód formuły $\neg a$ to dowód dla formuły $a \Rightarrow$ **false**, czyli konstrukcja tworząca dowód sprzeczności z każdego dowodu mającego być dowodem formuły a.
- Dowód formuły $\exists x \bullet p(x)$ to konstrukcja, która polega na wskazaniu pewnego obiektu n (z danej dziedziny rozważań) i podaniu dowodu dla formuły $p(n)$.
- Dowód formuły $\forall x \bullet p(x)$ to konstrukcja, która dla każdego obiektu n (z danej dziedziny rozważań) podaje dowód dla formuły $p(n)$.

Aksjomatyzacja Heytinga dla intuicjonistycznego rachunku zdań składa się z następujących *aksjomatów*:

(1) $a \Rightarrow (a \wedge a)$
(2) $(a \wedge b) \Rightarrow (b \wedge a)$
(3) $(a \wedge b) \Rightarrow ((a \wedge c) \Rightarrow (b \wedge c))$
(4) $((a \Rightarrow b) \wedge (b \Rightarrow c)) \Rightarrow (a \Rightarrow c)$
(5) $b \Rightarrow (a \Rightarrow b)$
(6) $(a \wedge (a \Rightarrow b)) \Rightarrow b$
(7) $a \Rightarrow (a \vee b)$
(8) $(a \vee b) \Rightarrow (b \vee a)$
(9) $((a \Rightarrow c) \wedge (b \Rightarrow c)) \Rightarrow ((a \vee b) \Rightarrow c))$
(10) $\neg a \Rightarrow (a \Rightarrow b)$
(11) $((a \Rightarrow b) \wedge (a \Rightarrow \neg b)) \Rightarrow \neg a$

Jedyną *regułą* jest reguła odrywania.

Wszystkie prawa intuicjonistycznego rachunku zdań są również prawami logiki klasycznej, ale nie odwrotnie: są tautologie logiki klasycznej, które nie są prawami logiki intuicjonistycznej. Przykładami takich formuł są:

$\neg a \vee a$
$\neg\neg a \Rightarrow a$

Dołączenie jednej z nich do zestawu wcześniej podanych aksjomatów dałoby rachunek równoważny logice klasycznej. W interpretacji intuicjonistycznej przyjęcie na przykład formuły $\neg a \vee a$ jako aksjomatu oznaczałoby, że dla dowolnej formuły ma się dowód jej prawdziwości lub dowód jej fałszywości.

Pokrewne podejście przedstawił A. Kołmogorow[26], który zaproponował, aby zdania w logice intuicjonistycznej traktować jako *problemy* lub *zadania*. Z zadaniem kojarzy się sposób jego rozwiązania. W logice klasycznej wypowiedzi, która jest zdaniem, przypisuje się wartość prawdy albo fałszu, natomiast w logice intuicjonistycznej zadaniu przypisuje się rozwiązanie albo bezsensowność, czyli brak możliwości rozwiązania. Inaczej: oceną logiczną zadania jest jego konstruktywne rozwiązanie albo bezsensowność zadania. Niech będą dane następujące przykłady zadań [Turski 1988]:

1. Znaleźć cztery liczby całkowite x, y, z, n takie, że: $x^n + y^n = z^n$ dla $n > 2$.
2. Udowodnić fałszywość wielkiego twierdzenia Fermata.
3. Przeprowadzić okrąg przez trzy zadane punkty p, q, r, nie posługując się innymi narzędziami niż cyrklem i linijką.
4. Zakładając, że znany jest jeden pierwiastek równania: $ax^2 + bx + c = 0$, znaleźć drugi pierwiastek tego równania.
5. Zakładając, że liczba π jest wymierna, $\pi = m/n$, znaleźć podobne wyrażenia dla liczby e.

Rozwiązanie zadania 1. oznacza rozwiązanie zadania 2., natomiast odwrotnie tak być nie musi, gdyż możliwe byłoby rozwiązanie zadania 2. przez sprowadzenie do sprzeczności, bez podawania kontrprzykładu. Zadania 3. i 4. są oczywiście rozwiązywalne, natomiast zadanie 5. jest bezsensowne, gdyż założenie o wymierności liczby π jest niemożliwe do spełnienia.

Jednym ze sposobów wyrażania semantyki formuł logiki intuicjonistycznej jest model Kripkego, wprowadzony już wcześniej przy omawianiu logik modalnych. Model ten dogodnie jest opisać w terminach procesu nabywania wiedzy w kolejnych chwilach (etapach).

Jak poprzednio, model Kripkego jest trójką $K = <S, \preccurlyeq, v>$, gdzie: S jest dowolnym zbiorem chwil (etapów), $\preccurlyeq$ jest relacją porządku częściowego nad S, $v : V \times S \rightarrow Lo-$

[26] Andriej Nikołajewicz Kołmogorow (1903–1987).

giczne jest funkcją wartościującą zmienne zdaniowe w każdej z chwil. Dodatkowo wymaga się, aby funkcja wartościująca spełniała następujący warunek:

jeżeli $s \preccurlyeq t$, to $v(a, s) \Rightarrow v(a, t)$, dla dowolnego $a \in V$.

Warunek ten oznacza, że jeżeli w pewnej chwili (etapie) $s \in S$ wartościowanie zmiennej $a \in V$ stanie się prawdziwe, to pozostanie ono prawdziwe we wszystkich następnych chwilach $t \in S$ (etapach) procesu nabywania wiedzy. Ponieważ porządek $\preccurlyeq$ jest częściowy, nie musi więc być porządkiem liniowym, istnieją różne drogi nabywania wiedzy.

Niech $\alpha \in FORM$ będzie dowolną formułą intuicjonistycznego rachunku zdań oraz $s \in S$ będzie dowolną chwilą. Interpretacja formuły α przy wartościowaniu v w chwili s, zapisywana $INT_{v,s}(\alpha)$, jest definiowana rekursywnie względem struktury składniowej:

a) $INT_{v,s}(a) =_{def} \mathbf{P}$ wtt $v(a, s) = \mathbf{P}$, dla $a \in V$,

b) $INT_{v,s}(\alpha \vee \beta) =_{def} \mathbf{P}$ wtt $INT_{v,s}(\alpha) = \mathbf{P}$ lub $INT_{v,s}(\beta) = \mathbf{P}$,

c) $INT_{v,s}(\alpha \wedge \beta) =_{def} \mathbf{P}$ wtt $INT_{v,s}(\alpha) = \mathbf{P}$ oraz $INT_{v,s}(\beta) = \mathbf{P}$,

d) $INT_{v,s}(\alpha \Rightarrow \beta) =_{def} \mathbf{P}$ wtt dla dowolnej chwili t takiej, że $s \preccurlyeq t$, zachodzi $INT_{v,t}(\alpha) = \mathbf{P}$ oraz $INT_{v,t}(\beta) = \mathbf{P}$,

e) $INT_{v,s}(\neg\alpha) =_{def} \mathbf{P}$ wtt dla dowolnej chwili t takiej, że $s \preccurlyeq t$, nie zachodzi $INT_{v,t}(\alpha) = \mathbf{P}$,

Użyty tu symbol wtt jest skrótem zwrotu *wtedy i tylko wtedy, gdy*.

Formuła α jest tautologią wtedy i tylko wtedy, gdy $INT_{v,s}(\alpha) = \mathbf{P}$ dla dowolnego wartościowania v i dowolnej chwili s.

Łatwo sprawdzić, że jeśli $S = \{0, 1\}$, $v(a, 0) = \mathbf{F}$ oraz $v(a, 1) = \mathbf{P}$, to formuła $\neg a \vee a$ nie jest tautologią, gdyż nie zachodzi $INT_{v,0}(\neg a \vee a) = \mathbf{P}$, co z kolei wynika z tego, że nie zachodzi $INT_{v,0}(a) = \mathbf{P}$ ani $INT_{v,0}(\neg a) = \mathbf{P}$.

14.6. O logikach niemonotonicznych

Rozpatrywane dotychczas logiki mają wspólną własność, określaną mianem *monotoniczności*. Oznacza to, że jeżeli α jest konsekwencją składniową pewnego zbioru formuł Φ, symbolicznie $\Phi \vdash \alpha$, to α jest również konsekwencją składniową dowolnego rozszerzenia zbioru Φ, symbolicznie $\Phi \cup \Gamma \vdash \alpha$, gdzie Γ jest dowolnym zbiorem formuł, czyli:

jeżeli $\Phi \vdash \alpha$, to $\Phi \cup \Gamma \vdash \alpha$.

Własność monotoniczności jest zachowana w tych wszystkich praktycznych sytuacjach, gdy wnioskowanie na podstawie pewnego zbioru przesłanek opiera się na założeniu, że dysponuje się pełną wiedzą o fragmencie opisywanego świata – założenie o zamkniętości świata (rozdział 12.). Założenie takie nie zawsze jest prawdziwe, gdyż mamy często do czynienia z informacją niepełną lub niepewną.

Rozpatrzmy na przykład dwie bazy danych: rozkład odjazdów pociągów z danej stacji oraz książkę telefoniczną. Jeżeli w rozkładzie pociągów odjeżdżających nie znajdziemy miejscowości, do której chcemy jechać, to znaczy, że nie ma do niej bezpośredniego pociągu. Jeżeli w książce telefonicznej nie znajdziemy nazwiska znajomego, to nie znaczy, że nie ma on telefonu, gdyż książka może być nieaktualna lub telefon może być zastrzeżony. W przypadku rozkładu jazdy pociągów założenie o zamkniętości świata jest uzasadnione, nie jest tak natomiast w przypadku książki telefonicznej.

We wnioskowaniach stosowanych na co dzień używa się reguł wnioskowania opartych na posiadanej wiedzy oraz niewiedzy. Przykładami reguł, które na takich podstawach wyprowadzają różne przeciwstawne rodzaje wniosków, są:

Jeżeli nie ma dowodu winy podejrzanego, to należy uznać, że jest on niewinny.

Jeżeli nadlatuje samolot i nie można wykluczyć, że jest to samolot wroga (nie ma dowodu, że jest to „swój" samolot), należy uznać, że jest to samolot wroga (i zestrzelić).

Wnioski wyprowadzane na podstawie tych reguł mogą się okazać sprzeczne z dodatkowo ujawnionymi faktami – nowymi informacjami o podejrzanym, wynikami oględzin strąconego samolotu.

Przedstawione reguły wnioskowania określa się jako *reguły domniemań*. Mają one często postać

$$\frac{\alpha, UNLESS(\beta)}{\gamma}$$

gdzie $UNLESS(\beta)$ oznacza: nie jest możliwe wyprowadzenie β. Logika stosująca reguły o takiej postaci narusza własność monotoniczności. Na przykład, na podstawie reguły

$$\frac{UNLESS(\alpha)}{\beta}$$

można stwierdzić, że $\varnothing \vdash \beta$, ale $\{\alpha\} \nvdash \beta$.

Przegląd różnych podejść do wnioskowania w sytuacji niepełnej informacji i związanych z nimi problemów można znaleźć na przykład w książce [Bolc, Borodziewicz, Wójcik 1991].

W sytuacji posiadania wiedzy niepewnej powstaje problem niejednoznaczności wnioskowania. Na przykład, jaki wniosek wyprowadzić przy założeniu posiadania następującej wiedzy:

- *Kwakierzy są na ogół pacyfistami.*
- *Republikanie na ogół nie są pacyfistami.*
- *Nixon jest kwakierem i republikaninem.*

Równie uzasadniony jest każdy z dwóch nasuwających się przeciwstawnych wniosków, ale nie jest możliwe jednocześnie, że:

Nixon jest pacyfistą.
Nixon raczej nie jest pacyfistą.

Z podobną niejednoznacznością ma do czynienia lekarz, gdy na podstawie badań pacjenta okazuje się, że może on być chory na jedną z kilku chorób.

Wnioskowanie w takich przypadkach opiera się na analizie scenariuszy postępowania, któremu towarzyszy dokonanie wyborów – podejmowanie decyzji. Praktycznie chodzi o ocenę skutków (koszt) podejmowanych decyzji. Stosuje się różne podejścia do takich ocen, oparte na przykład na miarach probabilistycznych lub miarach rozmytych. W logikach probabilistycznych miarą logicznej wartości zdania jest prawdopodobieństwo jego prawdziwości, a w logikach rozmytych miarami są rozmyte wartości prawdy.

Przegląd różnych podejść do wnioskowania w sytuacji niepewnej informacji i związanych z nimi problemów można znaleźć na przykład w książce [Bolc, Borodziewicz, Wójcik 1991].

Ćwiczenia

1. Która z podanych definicji jest poprawną definicją implikacji w logice $\boldsymbol{L}_3$:
   a) $p \Rightarrow q =_{\text{def}} min(1, 1 + p - q)$
   b) $p \Rightarrow q =_{\text{def}} max(1 - p, 1 - q)$
   c) $p \Rightarrow q =_{\text{def}} min(1, 1 - p + q)$
2. Czy formuły $p \Rightarrow q$ oraz $\neg p \vee q$ są równoważne w logice $\boldsymbol{L}_3$?
3. Które z podanych formuł są tautologiami temporalnej logiki zdań:
   a) $q \Rightarrow (p \Rightarrow q)$
   b) $p \wedge q \Rightarrow q$
   c) $\Box\ (p \wedge q \Rightarrow q)$
   d) $\Box\ (q \Rightarrow (\Box\ (p \Rightarrow q))$
4. Pokazać, że dla formuły $\Box q$ temporalnej logiki zdań nie istnieje równoważna jej formuła, składająca się wyłącznie ze spójników $\wedge$,$\vee$ oraz $\neg$.
5. Które z podanych formuł są tautologiami intuicjonistycznego rachunku zdań:
   a) $\neg p \vee p$
   b) $\neg\neg p \Rightarrow p$
   c) $p \Rightarrow \neg\neg p$
   d) $(p \Rightarrow q) \vee (q \Rightarrow p)$
   e) $\neg p \wedge \neg q \Rightarrow \neg(p \vee q)$
   f) $\neg\neg(\neg p \vee p)$.

15. Definiowanie języka programowania

15.1. Uwagi wstępne

W tym rozdziale zilustrowano zastosowanie metod logiki klasycznej do definiowania języków programowania. Języki programowania są na ogół bardziej złożone niż omawiane wcześniej języki logiki. Struktura definiowania każdego języka, zarówno sztucznego, jak i naturalnego, jest podobna – wspólnymi elementami są definicja słownika, ogólniej zbioru jednostek leksykalnych, definicja składni i semantyki.

Języki programowania stanowią bardzo obszerną grupę języków. Dzieli się je na trzy kategorie: języki *imperatywne* (proceduralne), *funkcyjne* i *logiczne*. Krótkie informacje na temat logicznego języka programowania były przedstawione w rozdziale 12. W bieżącym rozdziale jest przedstawiona definicja bardzo prostego języka programowania imperatywnego. Języki imperatywne są najszerszą stosowaną kategorią, należą do niej tak popularne teraz języki, jak Ada, C, C++, C#, Java, Pascal itp. Obecnie wśród języków imperatywnych dominują języki *programowania obiektowego*, podczas gdy dziesięć lat wcześniej przeważały języki *programowania strukturalnego*.

Przegląd metod definiowania języków programowania sekwencyjnego zawiera między innymi książka [Dembiński, Małuszyński 1981], definiowanie semantyki języków programowania sekwencyjnego i równoległego – książka [Apt, Olderog 1991], języków programowania czasu rzeczywistego – monografia [Huzar 1989].

W tym rozdziale omawia się trzy coraz bardziej rozbudowywane wersje prostego języka programowania strukturalnego. Wersje te będą oznaczane symbolami *BPJP* – bardzo prosty język programowania, *PJP* – prosty język programowania oraz *JP* – język programowania. Bardzo prosty język programowania *BPJP* jest skrajnie uproszczoną wersją języka programowania strukturalnego. *BPJP* nie uwzględnia hierarchicznej struktury blokowej programów, nie zawiera procedur, nie ma typów złożonych, nie ma typów referencyjnych itp. Język *PJP* jest rozszerzeniem *BPJP* o procedury nierekursywne, natomiast *JP* jest kolejnym rozszerzeniem o procedury rekursywne. Wszystkie opisywane konstrukcje mają swoje odpowiedniki we współczesnych językach programowania strukturalnego i obiektowego.

15.2. Jednostki leksykalne *BPJP*

Alfabet każdego nietrywialnego języka programowania jest zwykle bardzo liczny, czasem teoretycznie nieograniczony. Z tego względu alfabet języka wymaga oddzielnej definicji. Alfabet jest zbiorem napisów nad pewnym skończonym repertuarem symboli, w przypadku języka *BPJP* jest to podzbiór symboli graficznych alfabetu polskiego, rozszerzonego o powszechnie spotykane symbole matematyczne. Precyzyjne określenie tego podzbioru nie jest konieczne, gdyż – po pierwsze – wynika to w oczywisty sposób z przytoczonych niżej określeń jednostek leksykalnych i – po drugie – wybór innego repertuaru symboli nie ma znaczenia dla definicji semantyki języka.

Alfabet języka *BPJP* zawiera następujące kategorie jednostek leksykalnych:

- zbiór zmiennych indywiduowych, dalej krótko – zmiennych, reprezentowanych przez identyfikatory – elementy zbioru *Identyfikator*,
- zbiór stałych indywiduowych typu logicznego – wartości logicznych ze zbioru
  $Boolean =_{def} \{true, false\}$
- zbiór symboli operacji logicznych
  $SpLog =_{def} \{not, and, or\}$
- zbiór stałych indywiduowych typu całkowitoliczbowego – liczb w postaci dziesiętnej ze zbioru
  $Integer =_{def} \{-N, ..., N\}$ dla wybranego $N \in Nat$
- zbiór symboli operacji arytmetycznych – symboli funkcyjnych
  $FunInt =_{def} \{-_1\} \cup \{+, -_2, *, div, mod\}$

  o sygnaturach:
  $-_1 : Integer \rightarrow Integer$,
  $\_+\_, -_2\_, \_*\_, div\_, \_mod\_ : Integer^2 \rightarrow Integer$
- zbiór symboli relacji arytmetycznych – symboli predykatów
  $PredInt =_{def} \{ =, \neq, \leq\}$

  o sygnaturach:
  $\_=\_, \_\neq\_, \_\leq\_ : Int^2 \rightarrow Boolean$,
- zbiór stałych indywiduowych typu znakowego – symboli ze zbioru
  $Character =_{def} \{$'a', 'b', ..., 'z'$\} \cup \{$'0', '1', ..., '9'$\}$
- zbiór symboli operacji znakowych – symboli funkcyjnych
  $FunChar =_{def} \{ord, chr\}$

o sygnaturach:

ord : *Character* → *Integer*,
chr : *Integer* → *Character*,

- zbiór symboli relacji znakowych – symboli predykatów

  $PredChar =_{def} \{=, \neq, \leq\}$

  o sygnaturach:

  $\_=\_, \_\neq\_, \_\leq\_ : Character^2 \rightarrow Boolean$,

- zbiór stałych indywiduowych typu napisowego – symboli ze zbioru

  $String =_{def} Fin\text{-}seq_0(Character) \cup ... \cup Fin\text{-}seq_{256}(Character)$

  Jest to zbiór ciągów nad zbiorem *Character* o długości ograniczonej do 256. Dla odróżnienia napisu od identyfikatora, napisy będą ujmowane w cudzysłowy, na przykład: "abc", "10cd"; ciąg pusty będzie zapisywany w postaci " ".

- zbiór symboli operacji napisowych – symboli funkcyjnych

  $FunString =_{def}$ {head, tail, length} $\cup$ {^}

  o sygnaturach:

  head : *String* → *Character*,
  tail : *String* → *String*,
  length : *String* → *Integer*,
  $\_\hat{}\_ : String^2 \rightarrow String$.

- zbiór symboli relacji napisowych – symboli predykatów

  $PredString =_{def} \{=, \neq, \leq\}$

  o sygnaturach:

  $\_=\_, \_\neq\_, \_\leq\_ : String^2 \rightarrow Boolean$,

- zbiór symboli pomocniczych

  $SymbPom =_{def} \{(,), ;, :=, \text{“}, \text{”}, \text{‘}, \text{’}\} \cup \{,\}$

  Jednym z symboli pomocniczych jest przecinek. W celu odróżnienia przecinka jako separatora od przecinka jako symbolu pomocniczego został on dołączony jako odrębny jednoelementowy zbiór.

- zbiór nazw typów

  SymbType = {bool, int, char, string}

- zbiór słów kluczowych

  $SymbKlucz =_{def}$ {**program**, **begin**, **end**, **if**, **then**, **else**, **fi**, **while**, **do**, **od**}

- zbiór elementarnych instrukcji

  $InsElem =_{def}$ {read, write, skip}

Symbole predykatów =, ≠, ≤ są symbolami przeciążonymi, gdyż jako argumenty mogą mieć wartości różnych typów. Podobnie jest przeciążony symbol funkcyjny –, gdyż może występować w roli operatora jedno- bądź dwuargumentowego, w dalszym ciągu dolny indeks wskazujący liczbę argumentów będzie pomijany.

Specyfiką przedstawionego zestawu jednostek leksykalnych jest to, że w pośredni sposób wyznaczają one zbiory dziedzin semantycznych. Zbiory stałych indywiduowych wyróżnionych tu typów (logiczne, całkowitoliczbowe, znakowe i napisowe) są właśnie takimi dziedzinami.

Uwaga

Symbole jednostek leksykalnych rozpatrywanych tu języków programowania będą pisane antykwą, natomiast wszystkie pozostałe symbole pomocnicze będą pisane kursywą.

15.3. Składnia *BPJP*

Wyróżnia się następujące kategorie napisów występujących w BPJP:

- deklaracje zmiennych,
- wyrażenia – odpowiedniki termów w języku logiki – typów całkowitoliczbowych, znakowych i napisowych,
- wyrażenia – odpowiedniki formuł w języku logiki,
- instrukcje,
- programy.

Deklaracje zmiennych

Zmienne mają typy. Wyróżnia się zmienne typów: logicznego, całkowitoliczbowego, znakowego i napisowego. Jeżeli $x \in Ident$, to pojedyncza deklaracja, że identyfikator x jest zmienną odpowiedniego typu przyjmuje jedną z dwóch postaci:

bool x
int x
char x
string x

albo

bool x := t_{bool}
int x := t_{int}
char x := t_{char}
string x := t_{string}

gdzie: t_{bool}, t_{int}, t_{char}, t_{string} są wyrażeniami stałymi (niezawierającymi zmiennych – patrz niżej) odpowiednio typów logicznego, całkowitoliczbowego, znakowego i napisowego.

Każdy z wymienionych wyżej napisów jest deklaracją pojedynczej zmiennej.

Zbiór deklaracji zmiennych *var* jest zapisywany w postaci ciągu pojedynczych deklaracji, w których kolejne elementy są oddzielane średnikami, na przykład:

```
int abc;
int b10 := 10;
bool w := true;
char x := 'x';
string st := "aaaabb";
```

Rodzina zbiorów deklaracji zmiennych V będzie oznaczona przez $VAR(V)$.

Dalej będą używane następujące oznaczenia na zbiory zmiennych poszczególnych typów:

$V_{bool} = \{x \mid (\text{bool } x \in VAR) \vee (\text{bool } x := t_{int} \in VAR(V))\}$

$V_{int} = \{x \mid (\text{int } x \in VAR) \vee (\text{int } x := t_{int} \in VAR(V))\}$

$V_{char} = \{x \mid \text{char } x \in VAR \vee \text{char } x := t_{int} \in VAR(V)\}$

$V_{string} = \{x \mid \text{string } x \in VAR \vee \text{string } x := t_{int} \in VAR(V)\}$

Zakłada się, że zbiory zmiennych różnych typów V_{bool}, V_{int}, V_{char} oraz V_{string} są rozłączne. Suma wszystkich zmiennych będzie oznaczana przez V, czyli

$$V = V_{int} \cup V_{char} \cup V_{string} \cup V_{bool}$$

Wyrażenia

Zbiór wyrażeń języka *BPJP* jest mnogościową sumą

$$TERM(V) = TERM_{int}(V) \cup TERM_{char}(V) \cup TERM_{string}(V)$$

gdzie:

$TERM_{int}(V)$ jest zbiorem wyrażeń całkowitoliczbowych nad zbiorem zmiennych V,
$TERM_{char}(V)$ jest zbiorem wyrażeń znakowych nad zbiorem zmiennych V,
$TERM_{string}(V)$ jest zbiorem wyrażeń napisowych nad zbiorem zmiennych V.

Poszczególne zbiory wyrażeń są definiowane rekursywnie.

Zbiór $TERM_{int}(V)$ jest zdefiniowany następująco:

- $V_{int} \cup Integer \subseteq TERM_{int}(V)$,
- jeżeli $t_1, t_2 \in TERM_{int}(V)$, to

  $-t_1, (t_1 + t_2), (t_1 - t_2), (t_1 * t_2), (t_1 \text{ div } t_2), (t_1 \text{ mod } t_2) \in TERM_{int}(V)$

  W dalszym ciągu, tam, gdzie nie będzie to wprowadzać niejednoznaczności, nawiasy będą pomijane, a kolejność wykonywanych operacji będzie wynikać ze znanej powszechnie konwencji arytmetycznej.

- jeżeli $t \in TERM_{char}(V)$, to
  $\text{ord}(t) \in TERM_{int}(V)$
- jeżeli $t \in TERM_{string}(V)$, to
  $\text{length}(t) \in TERM_{int}(V)$

Zbiór $TERM_{char}(V)$ jest zdefiniowany następująco:

- $V_{char} \cup Character \subseteq TERM_{char}(V)$
- jeżeli $t \in TERM_{int}(V)$, to
  $\text{chr}(t) \in TERM_{char}(V)$

Zbiór $TERM_{string}(V)$ jest zdefiniowany następująco:

- $V_{string} \cup String \subseteq TERM_{string}(V)$
- jeżeli $t_1, t_2 \in TERM_{string}(V)$, to
  $\text{head}(t_1)$, $\text{tail}(t_1)$, $\text{length}(t_1)$, $(t_1 \wedge t_2) \in TERM_{string}(V)$

Formuły

Zbiór wyrażeń logicznych $FORM(V)$ jest definiowany rekursywnie w sposób następujący:

- $V_{bool} \cup Boolean \subseteq FORM(V)$,
- jeżeli $t_1, t_2 \in TERM_{typ}(V)$, gdzie $typ \in \{Int, Char, String\}$, to
  $t_1 = t_2$, $t_1 \neq t_2$, $t_1 \leq t_2 \in FORM(V)$
- jeżeli $\alpha, \beta \in FORM(V)$, to
  $\text{not}\,\alpha$, $(\alpha \text{ and } \beta)$, $(\alpha \text{ or } \beta) \in FORM(V)$

Zbiór formuł $FORM(V)$ języka *BPJP* jest podzbiorem formuł rachunku kwantyfikatorów, gdyż nie zawiera kwantyfikatorów – jest zbiorem formuł otwartych.

W tradycyjnych definicjach języków programowania składnię termów i formuł definiuje się jako wspólny zbiór wyrażeń, a formuły są traktowane jako wyrażenia typu logicznego. W celu zachowania zgodności opisu przedstawianego języka z wcześniejszymi opisami języków logiki, formuły wyróżniono jako oddzielną grupę napisów.

Instrukcje

Zbiór instrukcji *INSTR* języka *BPJP* jest zdefiniowany rekursywnie:

- read(x), write(x), skip $\in$ *INSTR*, gdzie $x \in V$,
- jeżeli $t \in TERM_{typ}(V)$ oraz $x \in V_{typ}$, gdzie $typ \in$ {int, char, string}, to
  $x := t \in INSTR$,

- jeżeli $\alpha \in FORM(V)$ oraz $x \in V_{bool}$, to
  $x := \alpha \in INSTR$,
- jeżeli $\alpha \in FORM(V)$ oraz $ins_1, ins_2 \in INSTR$, to
  if α **then** ins_1 **else** ins_2 **fi** $\in INSTR$,
  (α nazywa się dozorem instrukcji)
- jeżeli $\alpha \in FORM(V)$ oraz $ins \in INSTR$, to
  while α **do** *ins* **od** $\in INSTR$,
  (α nazywa się dozorem, a *ins* – treścią instrukcji)
- jeżeli $ins_1, ins_2 \in INSTR$, to $ins_1; ins_2 \in INSTR$.

Instrukcje read(x), write(x), skip oraz instrukcja przypisania $x := t$ są instrukcjami elementarnymi, pozostałe są instrukcjami złożonymi.

Programy

Programem jest napis postaci:

program *P*
  var
begin
  ins
end

gdzie: $P \in Identyfikator$ jest nazwą programu, $var \in VAR(V)$ jest zbiorem deklaracji globalnych zmiennych programu oraz $ins \in INSTR$ jest instrukcją zwaną treścią programu.

Zbiór wszystkich programów języka *BPJP* będzie oznaczany przez $PROG_{BPJP}$.

Przedstawione wyżej reguły składniowe są regułami bezkontekstowymi i dlatego nie wyrażają wszystkich ograniczeń składniowych. Należy je uzupełnić o dodatkowe reguły, które określają poprawność składniową poszczególnych elementów programu biorąc pod uwagę kontekst, w którym elementy te występują.

Do ograniczeń tych należą:

- nazwa programu jest różna od nazw zmiennych deklarowanych w programie,
- zmienna może być zadeklarowana tylko jeden raz, co gwarantuje, że zbiory zmiennych różnych typów są rozłączne,
- dowolna zmienna występująca w treści programu musi mieć deklarację.

Uwaga

Składnię kontekstową, albo ograniczenia kontekstowe, nazywa się czasem semantyką statyczną.

15.4. Semantyka *BPJP*

Dziedzinami semantycznymi dla wyrażeń *BPJP* są:

- dla wyrażeń całkowitoliczbowych dziedziną tą będzie zbiór
  $Integer \cup \{nadmiar, \perp\}$,
  gdzie, przypomnijmy, symbol $\perp$ oznacza wartość niezdefiniowaną,
- dla wyrażeń znakowych – zbiór
  $Character \cup \{\perp\}$,
- dla wyrażeń napisowych – zbiór
  $String \cup \{\perp\}$,
- dla wyrażeń logicznych (formuł) – zbiór
  $Boolean \cup \{\perp\}$.

Interpretacja bazowa symboli operacji arytmetycznych oraz symboli relacji arytmetycznych jest rozumiana podobnie jak w przykładzie 8.8. Modyfikacja tej interpretacji wynika z wprowadzenia do dziedziny interpretacji nowej wartości $\perp$ (niezdefiniowane).

W celu wyjaśnienia tej modyfikacji podaje się interpretację jednej operacji dzielenia całkowitoliczbowego div:

$$a \text{ div } b = \begin{cases} a & \text{gdy } a \in \{nadmiar, niezdefiniowane\} \\ b & \text{gdy } a \in Integer \text{ oraz } b \in \{nadmiar, niezdefiniowane\} \\ a/b & \text{gdy } a \in Integer, b \in Integer \setminus \{0\} \text{ oraz } a/b \in Integer \\ nadmiar & \text{gdy } a \in Integer, b \in Integer \setminus \{0\} \text{ oraz } a/b \notin Integer \end{cases}$$

Podana definicja reprezentuje pewien sposób postępowania – algorytm – przy obliczaniu wartości wyrażenia: najpierw jest badany pierwszy argument i jeśli nie jest liczbą, to decyduje o wartości całego wyrażenia, w przypadku przeciwnym jest badany drugi argument i jeśli on również nie jest liczbą, to decyduje o wartości całego wyrażenia. Dopiero, gdy oba argumenty są liczbami, jest wykonywane dzielenie, a jego wynik, jeżeli mieści się w zbiorze *Integer*, stanowi liczbową wartość całego wyrażenia. Operacja dzielenia całkowitoliczbowego *a/b* jest zdefiniowana następująco:

$a/b = c$, gdzie $b*c + r = a$, oraz $0 \le r < b$.

Niezbędne modyfikacje pozostałych operacji i relacji arytmetycznych pozostawia się Czytelnikowi do samodzielnego opracowania.

Podobnie samodzielnie, kierując się znajomością dowolnie wybranego języka programowania, Czytelnik, zgodnie z dalej przedstawioną nieformalnie określoną interpreta-

cją, ustanowi formalną bazową interpretację symboli funkcyjnych i relacyjnych dla pozostałych typów wyrażeń: znakowych i napisowych.

Funkcja 'ord' ma znakowi przypisywać liczbę całkowitą, odpowiadającą pozycji tego znaku w ciągu: 'a', 'b', ..., 'z', '0', '1', ..., '9'. Do tego samego ciągu odnosi się relacja liniowego porządku '≤'. Funkcja 'chr' ma być funkcją odwrotną do funkcji 'ord'.

Funkcje '^', 'head', 'tail' oraz 'lenght' mają znaczenie określone w rozdziale 7. Relacja '≤' jest relacją leksykograficznego porządku określonego w rozdziale 3.

Wartościowanie zmiennych jest funkcją:

$$v = v_{int} \cup v_{char} \cup v_{string} \cup v_{bool}$$

gdzie:

$v_{int} : V_{int} \rightarrow Integer \cup \{nadmiar, \perp\}$
$v_{char} : V_{char} \rightarrow Character \cup \{\perp\}$
$v_{string} : V_{int} \rightarrow String \cup \{\perp\}$
$v_{bool} : V_{bool} \rightarrow Boolean \cup \{\perp\}$

Ponieważ v_{int}, v_{char}, v_{string}, v_{bool} są funkcjami o rozłącznych dziedzinach, zatem ich mnogościowa suma v jest również funkcją. Przez $WAR(V)$ oznacza się zbiór wszystkich wartościowań zbioru zmiennych $V = V_{int} \cup V_{char} \cup V_{string} \cup V_{bool}$.

Mając ustaloną interpretację bazową I wszystkich symboli funkcyjnych i relacyjnych oraz zdefiniowane pojęcie wartościowania zmiennych, można określić – podobnie jak w rozdziale 10. – interpretację $INT_v(t)$ dla dowolnego wyrażenia $t \in TERM(V)$ oraz $INT_v(\alpha)$ dla dowolnej formuły $\alpha \in FORM(V)$. Szczegóły tej definicji pozostawia się Czytelnikowi do uzupełnienia.

Deklaracja zmiennych $var \in VAR(V)$ wyznacza początkowe wartościowanie zmiennych:

- jeżeli zmienna x jest zadeklarowana w *var* i jej deklaracja ma postać *typ* x, gdzie *typ*∈ {bool, int, char, string}, to $v(x) = \perp$,
- jeśli jej deklaracja ma postać *typ* x := *t*, to $v(x) = INT_v(t)$; należy zwrócić uwagę na to, że *t* musi być termem stałym albo formułą stałą, co oznacza, że interpretacja $INT_v(t)$ nie zależy od wartościowania *v*.

Interpretacja instrukcji wymaga wprowadzenia nowego pojęcia – *konfiguracji programu*. Konfiguracja jest zdefiniowana jako para

$\langle v, ins \rangle$,

gdzie $v \in WAR(V)$ jest wartościowaniem zmiennych, a $ins \in INSTR$ jest instrukcją.

Konfiguracja odnosi się do stanu programu w danym momencie jego obliczeń. Obliczenie programu realizuje się przez wykonanie pewnego ciągu akcji – elementarnych

czynności obliczeniowych. Konfiguracje opisują stan programu przed i po wykonaniu akcji. Przedstawione podejście do opisu znaczenia programu w postaci pewnego ciągu konfiguracji towarzyszących wykonaniu obliczeń programu nazywa się *podejściem operacyjnym.*

Jeżeli program jest w konfiguracji $\langle v, ins \rangle$, oznacza to, że v reprezentuje aktualne wartościowanie zmiennych programu, a *ins* jest instrukcją, która pozostaje jeszcze do wykonania przez program.

Jeżeli nie pozostaje już nic do wykonania, czyli instrukcja *ins* jest pusta, zapisuje się to wyróżnionym symbolem END – konfiguracją $\langle v, \text{END} \rangle$.

Jeżeli podczas wykonywania akcji obliczeniowej zajdą warunki uniemożliwiające jej wykonanie, to oznacza zerwanie obliczeń programu, co będzie reprezentowane specjalnie wyróżnioną konfiguracją ABORT.

Zbiorem wszystkich konfiguracji jest

$$KONF = (WAR \times (INSTR \cup \{\text{END}\}) \cup \{\text{ABORT}\}$$

Interpretacja instrukcji polega na zdefiniowaniu relacji zmian konfiguracji. *Relacją zmiany konfiguracji* jest relacja o sygnaturze

$$\longrightarrow \subseteq KONF \times KONF$$

Jeśli $konf_1, konf_2 \in KONF$, to fakt $\langle konf_1, konf_2 \rangle \in \longrightarrow$ będzie zapisywany w postaci

$$konf_1 \longrightarrow konf_2$$

Nieformalna interpretacja tego faktu jest następująca: jeżeli program jest w konfiguracji $konf_1$, to po wykonaniu pewnej akcji obliczeniowej program znajdzie się w konfiguracji $konf_2$.

Jeżeli $konf_0$ jest początkową konfiguracja programu, to obliczeniem programu jest skończony lub nieskończony ciąg zmian konfiguracji. Skończony ciąg zmian konfiguracji jest postaci

$$konf_0 \longrightarrow konf_1 \longrightarrow \ldots \longrightarrow konf_n$$

gdzie $konf_n$ jest konfiguracją końcową.

Końcową może być jedna z konfiguracji: $\langle v_n, \text{END} \rangle$ albo ABORT. Pierwsza z nich oznacza, że program zakończył się pomyślnie, dostarczając wartościowanie v_n jako końcowy wynik obliczeń, druga z nich oznacza natomiast, że program zakończył się niepomyślnie – nastąpiło zerwanie jego obliczeń, i nie dostarcza żadnego wyniku końcowego.

Obliczenie programu może być ciągiem nieskończonym

$$konf_0 \longrightarrow konf_1 \longrightarrow \ldots \longrightarrow konf_n \longrightarrow \ldots$$

gdy nie istnieje konfiguracja końcowa.

Jeżeli dla każdej konfiguracji relacja $\longrightarrow$ wyznacza co najwyżej jedną nową konfigurację, czyli relacja jest funkcją, to mamy do czynienia z programem deterministycznym, w przeciwnym przypadku – z programem niedetrministycznym.

Relacja $\longrightarrow$ jest definiowana rekursywnie przez zbiór aksjomatów i reguł, dobrany tak, aby dla dowolnej konfiguracji było możliwe wyznaczenie następnej konfiguracji. Ponieważ zasadniczym elementem różnicującym konfiguracji jest instrukcja, aksjomaty i reguły definiujące zmiany konfiguracji są powiązane z regułami składniowymi, definiującymi zbiór instrukcji. Z tego względu mówimy, że definicja relacji zmiany konfiguracji jest definicją strukturalną względem definicji składni instrukcji.

Łącznie formalny opis relacji zmiany konfiguracji jest określony przez zbiór aksjomatów i reguł wnioskowania dotyczących wyprowadzania obliczenia programu.

Znaczenie instrukcji czytania read(x) jest oczywiste: instrukcja oznacza, że z otoczenia programu, za pomocą pewnego urządzenia, wprowadza się pewną wartość i przypisuje się ją zmiennej x, i tym samym zmienia się aktualne wartościowanie v. Formalnie wyraża to aksjomat

$$< v, \text{read(x)} > \longrightarrow < v[\text{x} := e], \text{END} >$$

gdzie e jest dowolną wartością ze zbioru tego samego typu, co typ zmiennej x.

Znaczenie instrukcji pisania write(x) jest również oczywiste: program przekazuje do swego otoczenia, za pomocą pewnego urządzenia, aktualną wartość przypisaną zmiennej x. Dla instrukcji pisania write(x) mamy dwa aksjomaty:

$$< v, \text{write(x)} > \longrightarrow < v, \text{END} > \qquad \text{gdy } v(\text{x}) \neq \perp$$

$$< v, \text{write(x)} > \longrightarrow \text{ABORT} \qquad \text{gdy } v(\text{x}) = \perp$$

Pierwszy aksjomat odnosi się do przypadku, gdy wartość zmiennej x jest zdefiniowana, a drugi – gdy taka wartość jest niezdefiniowana, czego rezultatem jest zerwanie obliczeń instrukcji – instrukcja kończy się w trybie awaryjnym.

Dla instrukcji skip jest tylko jeden aksjomat

$$< v, \text{skip} > \longrightarrow < v, \text{END} >$$

Aksjomat ten wyraża to, że instrukcja „nic nie robi". Instrukcja jest przydatna do wyrażenia szczególnej postaci instrukcji warunkowej.

Instrukcja przypisania x := *t* oznacza, że wartość wyrażenia *t* przypisuje się zmiennej x. Oczywiście, co było zaznaczone wcześniej, typ zmiennej x i wyrażenia *t* muszą być identyczne. Formalnie instrukcję charakteryzują dwa aksjomaty:

$$< v, \mathrm{x} := t > \longrightarrow < v[\mathrm{x} := INT_v(t)], \mathrm{END} > \quad \text{gdy } INT_v(t) \notin \{nadmiar, \perp\},$$

$$< v, \mathrm{x} := t > \longrightarrow \mathrm{ABORT} \quad \text{gdy } INT_v(t) \in \{nadmiar, \perp\}.$$

Aksjomaty odnoszą się do dwóch możliwych przypadków, gdy wartość wyrażenia *t* należy do zbioru wartości danego typu lub nie należy.

Instrukcja warunkowa **if** α **then** ins_1 **else** ins_2 **fi** oznacza, że w zależności od wartości wyrażenia logicznego – formuły α, należy wykonać instrukcję ins_1, gdy α ma wartość *true*, oraz instrukcję ins_2 w przypadku przeciwnym. Takie znaczenie instrukcji warunkowej wyrażają dwie reguły:

$$\frac{< v, ins_1 > \longrightarrow < v', ins_1' >}{< v, \textbf{if } \alpha \textbf{ then } ins_1 \textbf{ else } ins_2 \textbf{ fi} > \longrightarrow < v', ins_1' >} \quad \text{gdy } INT_v(\alpha) = \text{true}$$

$$\frac{< v, ins_2 > \longrightarrow < v', ins_2' >}{< v, \textbf{if } \alpha \textbf{ then } ins_1 \textbf{ else } ins_2 \textbf{ fi} > \longrightarrow < v', ins_2' >} \quad \text{gdy } INT_v(\alpha) = \text{false}$$

Instrukcja iteracji **while** α **do** *ins* **od** ma się wykonywać w taki sposób, że jeśli wartość wyrażenia logicznego α jest fałszem, to instrukcja się kończy, w przypadku przeciwnym powinna być wykonana instrukcja *ins*, po czym ponownie wylicza się wyrażenie α i – w zależności od wyliczonej wartości – powtarza się opisane postępowanie. Instrukcję charakteryzują aksjomat i reguła:

$$< v, \textbf{while } \alpha \textbf{ do } ins \textbf{ od} > \longrightarrow < v, \mathrm{END} > \quad \text{gdy } INT_v(\alpha) = \text{false}$$

$$\frac{< v, ins > \longrightarrow < v', ins' >}{< v, \textbf{while } \alpha \textbf{ do } ins \textbf{ od} > \longrightarrow < v', ins'; \textbf{while } \alpha \textbf{ do } ins \textbf{ od} >} \quad \text{gdy } INT_v(\alpha) = \text{true}$$

Warto zwrócić uwagę na to, że nowa konfiguracja

<*v'*, *ins'*; **while** α **do** *ins* **od**>,

do której następuje przejście, jest bardziej złożona niż konfiguracja startowa

<*v*, **while** α **do** *ins* **od**>.

Wynika to z faktu, że instrukcja

ins'; **while** α **do** *ins* **od**

reprezentuje sekwencyjne złożenie instrukcji *ins'*, która reprezentuje część instrukcji *ins* pozostającej do wykonania oraz instrukcji **while** α **do** *ins* **od** reprezentującej ewentualne kolejne powtórzenia pętli. W szczególnym przypadku, gdy *ins'* ≡ END, reguła przyjmuje postać

$$\frac{< v, ins > \longrightarrow < v', \text{END} >}{< v, \textbf{while } \alpha \textbf{ do } ins \textbf{ od} > \longrightarrow < v', \textbf{while } \alpha \textbf{ do } ins \textbf{ od} >} \quad \text{gdy } INT_v(\alpha) = \text{true}$$

Ostatnie reguły dotyczą sekwencyjnego złożenia instrukcji: ins_1; ins_2.

$$\frac{< v, ins_1 > \longrightarrow < v', ins_1' >}{< v, ins_1 ; ins_2 > \longrightarrow < v', ins_1' ; ins_2 >} \quad \text{gdy } ins_1' \neq \text{ABORT}$$

$$\frac{< v, ins_1 > \longrightarrow < v', \text{END} >}{< v, ins_1 ; ins_2 > \longrightarrow < v', ins_2 >} \quad \text{gdy } ins_1' \neq \text{ABORT}$$

$$\frac{< v, ins_1 > \longrightarrow \text{ABORT}}{< v, ins_1 ; ins_2 > \longrightarrow \text{ABORT}}$$

Sekwencyjne złożenie oznacza, że najpierw ma być wykonana instrukcja ins_1, następnie – po jej zakończeniu – instrukcja ins_2. Jeżeli instrukcja ins_1 nie wykona się w całości, ale podczas wykonywania nie nastąpi zerwanie obliczeń, to jej część ins_1', która pozostaje jeszcze do wykonania, jest sekwencyjnie złożona z instrukcją ins_2. Druga reguła odnosi się do szczególnego przypadku, gdy $ins_1' \equiv$ END. Jeżeli w trakcie wykonywania tej instrukcji nastąpi zerwanie obliczenia, to oznacza zerwanie całych obliczeń.

Semantyka operacyjna programu *P*:

program *P*
  var
begin
  ins
end

jest określona przez jego obliczenie (skończone bądź nieskończone)

$$Comp(P) = < v_0, ins_0 > \longrightarrow < v_1, ins_1 > \longrightarrow \ \ldots \ \longrightarrow < v_n, ins_n > \longrightarrow \ldots$$

rozpoczęte w konfiguracji początkowej $< v_0, ins_0 >$, gdzie: v_0 jest wartościowaniem początkowym wyznaczonym przez deklarację zmiennych *var*, a $ins_0 \equiv ins$ jest instrukcją początkową, którą stanowi treść programu.

Należy zauważyć, że program może mieć wiele obliczeń. Powodem tego jest instrukcja 'read', która powoduje, że podczas wykonywania instrukcji, w trakcie realizacji całego programu, zmienne programu mogą otrzymywać wartościowania zależne od wczytywanych wartości. To, jakie są to wartości, zależy od otoczenia programu.

Jeżeli natomiast w programie nie ma tej instrukcji, to program ma jedno obliczenie. Obliczenie nieskończone albo skończone osiągnięciem końcowej konfiguracji ABORT oznacza niepoprawne zachowanie programu w tym sensie, że jego obliczenie nie dostarcza wartościowych wyników. Wartościowe są tylko te obliczenia, które są skończone osiągnięciem końcowej konfiguracji postaci $\langle \nu, \text{END} \rangle$. Wartościowanie ν reprezentuje wyniki końcowe obliczenia programu. Należy zwrócić uwagę, że jeżeli obliczenie ma taką postać, nie stanowi to gwarancji, że dostarczone wyniki są zgodne z wynikami oczekiwanymi. Inaczej: fakt, że program liczy, nie oznacza, że jest programem poprawnym.

Poniżej rozpatruje się przykłady obliczeń prostych programów.

Przykład 15.1

Program mnożenie czyta dwie liczby i pisze wynik ich mnożenia:

```
program mnożenie
    int x;
    int y
begin
    read(x);
    read(y);
    x := x * y;
    write(x)
end
```

Obliczenie programu przebiega następująco:

Comp(mnożenie) =

$\langle \nu_0$, read(x); read(y); x := x * y; write(x)$\rangle \longrightarrow$

$\langle \nu_1$, read(y); x := x * y; write(x)$\rangle \longrightarrow$

$\langle \nu_2$, x := x * y; write(x)$\rangle \longrightarrow$

$\langle \nu_3$, write(x)$\rangle \longrightarrow$

$\langle \nu_4$, END$\rangle$

gdzie:

Wartościowanie początkowe ν_0 jest wyznaczone przez deklarację zmiennych:

$\nu_0(x) = \perp$,

$\nu_0(y) = \perp$.

Zakładając, że 10 jest pierwszą wczytaną liczbą, $v_1 = v_0[x := 10]$, czyli:
$v_1(x) = 10$,
$v_1(y) = \perp$.

Zakładając, że 2 jest drugą wczytaną liczbą, $v_2 = v_1[y := 2]$, czyli:
$v_2(x) = 10$,
$v_2(y) = 2$.

Dalsze wartościowania nie zmieniają się, czyli $v_2 = v_3 = v_4$.

W przypadku, gdyby wczytane liczby były takie, że ich iloczyn wykraczałby poza zakres dla liczb całkowitych, wówczas obliczenie miałoby postać:

Comp(mnożenie) =
$\langle v_0$, read(x); read(y); x := x * y; write(x)$\rangle \longrightarrow$
$\langle v_1$, read(y); x := x * y; write(x)$\rangle \longrightarrow$
$\langle v_2$, x := x * y; write(x)$\rangle \longrightarrow$
ABORT

Semantyka operacyjna przedstawia znaczenie programu w postaci ciągu zmian konfiguracji. Czasem jest wygodniejsze wyrażanie znaczenia programu P w postaci relacji o sygnaturze $Sem(P) \subseteq WAR^2$. Relację tę można zdefiniować na podstawie semantyki operacyjnej w sposób następujący:

$$Sem(P) = \{\langle v_0, v\rangle \mid \langle v_0, ins_0\rangle \xrightarrow{*} \langle v, \text{END}\rangle\},$$

gdzie: $\xrightarrow{*}$ jest zwrotnym, tranzytywnym domknięciem relacji zmian konfiguracji $\longrightarrow$. Wartościowanie początkowe v_0 reprezentuje dane początkowe programu, natomiast v – dane końcowe programu odpowiadające danym początkowym.

15.5. Język *PJP* – procedury nierekursywne

Składnia

Rozszerzenie języka od strony leksykalnej polega na dołączeniu do zbioru słów kluczowych języka *BPJP* zbioru nowych słów {**procedure**, **in**, **out**}.

Zasadniczymi elementami rozszerzenia składni są: nowa kategoria napisów – definicje procedur, oraz nowy rodzaj instrukcji – wywołanie procedury. Wprowadzenie tych elementów modyfikuje oczywiście zbiór instrukcji i postać programów.

Definicja procedury o nazwie p, gdzie $p \in$ *Identyfikator*, ma postać:

procedure p (*lista wejściowych i wyjściowych parametrów formalnych*)
  var_p
begin
  ins_p
end

Pierwsza linia tekstu określa sygnaturę procedury, druga – jest deklaracją jej zmiennych lokalnych, a trzecia określa jej treść. Struktura definicji procedury jest podobna do struktury definicji programu w języku *BPJP*.

Lista parametrów formalnych jest ciągiem, być może pustym, postaci

$typ_1\ kier_1\ par_1$; ... ; $typ_n\ kier_n par_n$

gdzie:

$typ_i \in$ {bool, int, char, string} jest typem parametru $par_i \in Identyfikator$, $i = 1, ..., n$,

$kier_i \in$ {**in**, **out**} oznacza kierunek komunikacji parametru par_i: **in** oznacza, że dany parametr jest parametrem wejściowym procedury, a **out** – że parametrem wyjściowym,

var_p jest deklaracją zmiennych lokalnych procedury, $var_p \in VAR(V)$,

ins_p jest instrukcją – treścią procedury; zakłada się przy tym, że w treści procedury nie ma instrukcji wywołania tej samej czy innej procedury, czyli $ins_p \in INSTR_{BPJP}$, gdzie $INSTR_{BPJP}$ jest zbiorem instrukcji języka *BPJP*.

Zbiór definicji procedur, oznaczany przez *proc*, będzie zapisywany w postaci listy pojedynczych definicji oddzielanych średnikami, na przykład:

```
procedure kwadrat(int in x; int out y)
begin
   y := x*x
end ;

procedure suma(int in x; int in y; int out z)
begin
   z := x + y
end
```

Rodzina zbiorów definicji procedur będzie oznaczana przez *PROC*.

Zbiór wyrażeń języka *PJP* jest taki sam jak języka *BPJP*. Zbiór instrukcji języka *PJP* będzie oznaczany przez $INSTR_{PJP}$. Stanowi on rozszerzenie zbioru instrukcji języka *BPJP*, oznaczanego $INSTR_{BPJP}$, które wynika z dołączenia nowej instrukcji.

Niech procedura ma sygnaturę

p(typ_1 **in** par_1; ... ; t_n **in** par_n; typ_{n+1} **out** par_{n+1}; ... ; typ_{n+m} **out** par_{n+m}),

w której – dla uproszczenia notacji – założono, że pierwsza grupa parametrów stanowi parametry wejściowe, a druga – parametry wyjściowe procedury. Instrukcja wywołania procedury ma wówczas postać

p(e_1, ..., e_n, y_1, ..., y_m)

gdzie:

e_1, ..., e_n są wyrażeniami takiego samego typu jak parametry par_1, ..., par_n,

y_1, ..., y_m są zmiennymi takiego samego typu jak parametry par_{n+1}, ..., par_{n+m}.

Listę $e_1, ..., e_n, y_1, ..., y_m$ nazywa się listą parametrów aktualnych.

Przyjęty mechanizm komunikacji pomiędzy procedurą a programem jest jednym z kilku spotykanych we współczesnych językach programowania.

Parametry wejściowe służą do komunikacji nazywanej komunikacją przez wartość. Oznacza to, że wartość reprezentowana przez parametr aktualny e_i – wartość wyrażenia $INT_v(e_i)$ w bieżącym wartościowaniu v zmiennych programu – jest przypisywana odpowiadającemu mu parametrowi formalnemu par_i, dla $i = 1, ..., n$.

Parametry wyjściowe służą do komunikacji przez wartości, ale w kierunku przeciwnym – od procedury do programu. Oznacza to, że wartość reprezentowana przez parametr formalny par_{n+j}, czyli jego wartościowanie, w momencie zakończenia procedury jest przypisywana odpowiadającemu mu parametrowi aktualnemu y_j, dla $j = 1, ..., m$.

Należy dodać, że procedura może korzystać z dostępu do zmiennych globalnych programu, co oznacza dodatkowy sposób wymiany danych pomiędzy procedurą a programem.

Składnia programów języka *PJP* jest określona następująco: jeżeli $P \in Identyfikator$ jest nazwą programu, $proc \in PROC$ jest zbiorem definicji procedur, $var \in VAR(V)$ jest zbiorem deklaracji zmiennych, $ins \in INSTR_{PJP}$, to napis postaci

program *P*
  var
  proc
begin
  ins
end

jest programem języka *PJP*.

Zbiór wszystkich programów języka *PJP* będzie oznaczany przez $PROG_{PJP}$. Oczywiście $PROG_{BPJP} \subseteq PROG_{PJP}$.

Podobnie jak poprzednio, przy definicji języka *BPJP*, przedstawione wyżej reguły składniowe wymagają uzupełnienia o reguły ograniczeń kontekstowych. Dodatkową regułą będzie tu wymóg, aby parametry procedury oraz jej zmienne lokalne były unikatowe i różne od zmiennych i parametrów innych procedur oraz od zmiennych globalnych – zmiennych deklarowanych w całym programie.

Semantyka

Konsekwencją wprowadzenia procedur jest wprowadzenie nowych konfiguracji pomocniczych związanych z opisem wywołania i zakończenia procedury. Wywołaniu i zakończeniu procedury będą odpowiadać dwa nowe aksjomaty.

Dla uproszczenia zakłada się, że procedura ma co najwyżej jeden wejściowy i co najwyżej jeden wyjściowy parametr formalny, dlatego będą rozpatrywane instrukcje wy-

wołania procedury postaci p(e, y), p(e), p(y), gdzie wyrażenie e jest pierwszym parametrem aktualnym, a zmienna y jest drugim parametrem aktualnym.

Wywołanie procedury p(e, y), która ma definicję

p(typ_1 **in** par_1; typ_2 **out** par_2)
  var_p
begin
  ins_p
end

będzie opisane następującym aksjomatem wywołania procedury nierekursywnej

$$< v, p(e, y) > \longrightarrow < v \cup v^p, \mathbf{begin}^p\ ins_p\ \mathbf{end}^p >$$

gdzie v^p jest wartościowaniem parametrów i zmiennych lokalnych procedury p.

Na mocy przyjętego założenia o unikalności parametrów i zmiennych lokalnych, funkcje v oraz v^p mają rozłączne dziedziny, a więc ich suma mnogościowa pozostaje funkcją.

Wartościowanie początkowe v^p jest określone następująco:

- $v^p(par_1) = INT_v(e)$,
- $v^p(par_2) = \perp$,
- jeżeli zmienna x jest zadeklarowana w var_p i jej deklaracja ma postać $typ\ x$, gdzie $typ \in$ {bool, int, char, string}, to $v^p(x) = \perp$,
- jeżeli zmienna x jest zadeklarowana w var_p i jej deklaracja ma postać $typ\ x := t$, to $v^p(x) = INT_v(t)$.

Aksjomat wyraża fakt, że wywołanie procedury, na okres jej wykonywania, rozszerza zbiór zmiennych programu. Wprowadzenie bloku pomocniczego, wyznaczonego słowami **begin**^{p} oraz **end**^{p}, służy do zaznaczenia tego fragmentu instrukcji programu, który należy do procedury p. We fragmencie tym obowiązuje rozszerzony zbiór wartościowanych parametrów i zmiennych.

Z zakończeniem wykonywania procedury nierekursywnej wiąże się aksjomat postaci

$$< v \cup v^p, \mathbf{begin}^p\ \text{END}\ \mathbf{end}^p;\ ins > \longrightarrow < v_1, ins >$$

gdzie $v_1(y) = v\ [y := v^p(par_2)]$.

Reguła opisuje przekazanie wartości obliczonej przez procedurę do tej części programu, z której nastąpiło jej wywołanie. Wynik obliczeń jest reprezentowany przez parametr formalny par_2. Wartościowanie tego parametru zostaje przypisane zmiennej y, która jest odpowiadającym mu parametrem aktualnym.

Przedstawiane aksjomaty rozszerzają definicję relacji zmian konfiguracji.

Obliczenie programu *PJP* jest definiowane tak samo jak programu *BPJP*, z tym że jest oparte na rozszerzonej relacji zmian konfiguracji.

Przykład 15.2

Niech dana będzie modyfikacja programu z poprzedniego przykładu:

```
program mnożenie
    int a;
    int b;
    int c;
    procedure razy(int in x; int in y, int out z)
    begin
        z := x * y
    end
begin
    read(a);
    read(b);
    razy(a, b, c);
    write(c)
end
```

Obliczenie programu przebiega następująco:

$Comp$(mnożenie) =

$\langle \nu_0$, read(a); read(b); razy(a, b, c); write(c)$\rangle \longrightarrow$

$\langle \nu_1$, read(b); razy(a, b, c); write(c)$\rangle \longrightarrow$

$\langle \nu_2$, razy(a, b, c); write(c)$\rangle \longrightarrow$

$\langle \nu_3$, **begin**^{razy} z := x * y **end**^{razy}; write(c)$\rangle \longrightarrow$

$\langle \nu_4$, **begin**^{razy} END **end**^{razy}; write(c)$\rangle \longrightarrow$

$\langle \nu_5$, write(c)$\rangle \longrightarrow$

$\langle \nu_6$, END$\rangle$

Wartościowanie początkowe ν_0 jest wyznaczone przez deklarację zmiennych:

$\nu_0(a) = \perp$,
$\nu_0(b) = \perp$,
$\nu_0(c) = \perp$.

Zakładając, że 10 jest pierwszą wczytaną liczbą, $\nu_1 = \nu_0[a := 10]$, czyli:

$\nu_1(a) = 10$,
$\nu_1(b) = \perp$,
$\nu_1(c) = \perp$.

Zakładając, że 2 jest drugą wczytaną liczbą, $\nu_2 = \nu_1[b := 2]$, czyli:

$\nu_2(a) = 10$,
$\nu_2(b) = 2$,
$\nu_2(c) = \perp$.

Wywołanie procedury razy prowadzi do wartościowania $v_3 = v_2 \cup v_1^{razy}$, gdzie:
$v_1^{razy}(x) = v_2(a)$,
$v_1^{razy}(y) = v_2(b)$,
$v_1^{razy}(z) = \perp$.

Wykonanie treści procedury prowadzi do wartościowania $v_4 = v_2 \cup v_2^{razy}$, gdzie:
$v_2^{razy}(x) = v_2(a)$,
$v_2^{razy}(y) = v_2(b)$,
$v_2^{razy}(z) = INT_{v_3}(x * y) = 20$.

Wyjście z procedury i powrót do głównego programu prowadzi do wartościowania $v_5 = v_4[c := v_2^{razy}(z)]$, czyli:
$v_5(a) = 10$,
$v_5(b) = 2$,
$v_5(c) = 20$.

Oczywiście $v_5 = v_6$.

15.6. Język *JP* – procedury rekursywne

Kolejne, ostatnie rozszerzenie języka nie wprowadza nowych jednostek leksykalnych. Zmiana dotyczy składni i polega tylko na eliminacji założenia, że w treści procedur nie dopuszcza się wywoływania innych procedur. W szczególności w treści danej procedury może być wywołanie tej samej procedury.

Zmiana składniowa pociąga za sobą modyfikację relacji zmian konfiguracji, a dokładniej – modyfikację samej konfiguracji. Potrzeba modyfikacji wiąże się z tym, że wywołanie procedury powoduje rozszerzenie zbioru wartościowanych parametrów i zmiennych. W przypadku, gdy w trakcie wykonywania procedury następuje ponowne wywołanie tej samej procedury, powstaje konieczność zapewnienia, by parametry i zmienne lokalne kolejnego wywołania były odróżniane od parametrów i poprzedniego wywołania tej procedury. W tym celu, przy kolejnym wywołaniu procedury, należy dokonać odpowiedniego przemianowania parametrów i zmiennych procedury. Mechanizm stosowanego dalej przemianowania polega na tym, że nazwa każdego parametru i zmiennej lokalnej procedury otrzymuje dodatkowy indeks – liczbę, która zwiększa się o jeden przy każdym kolejnym wywołaniu procedury.

Wywołanie procedury $p(e, y)$ o definicji

$p(typ_1$ **in** par_1; typ_2 **out** $par_2)$
  var_p
begin
  ins_p
end

gdzie tym razem var_p może zawierać instrukcje wywołania procedur, będzie opisane dwoma aksjomatami wywołania procedury rekursywnej.

Pierwszy aksjomat, dotyczący pierwszego wywołania procedury p, stanowi uogólnienie poprzedniego aksjomatu wywołania procedury nierekursywnej

$$< v, p(e,y) > \longrightarrow < v \cup v^{p,0}, \mathbf{begin}^{p,0}\ ins\ \mathbf{end}^{p,0} >$$

gdzie $v^{p,0}$ jest wartościowaniem parametrów i zmiennych lokalnych procedury p.

Wprowadzony blok, określony symbolami $\mathbf{begin}^{p,0}$ ... $\mathbf{end}^{p,0}$ oznacza, że parametry formalne i zmienne lokalne procedury p są indeksowane liczbą 0, będzie to zaznaczane w postaci górnego indeksu dodanego do każdego parametru i zmiennej lokalnej.

Drugi aksjomat dotyczy wywołania procedury p, które zachodzi w trakcie wykonywania uprzedniego, jeszcze niezakończonego wywołania tej procedury.

$$< v \cup v^{p,0} \cup \ldots \cup v^{p,k},$$
$$\mathbf{begin}^{p,0} \ldots \mathbf{begin}^{p,k}\ p(e,y); ins_k\ \mathbf{end}^{p,k} \ldots ins_0\ \mathbf{end}^{p,0} >$$
$$\longrightarrow$$
$$< v \cup v^{p,0} \cup \ldots \cup v^{p,k} \cup v^{p,k+1},$$
$$\mathbf{begin}^{p,0} \ldots \mathbf{begin}^{p,k} \mathbf{begin}^{p,k+1} ins_p\ \mathbf{end}^{p,k+1} ins_k \mathbf{end}^{p,k} \ldots ins_0\ \mathbf{end}^{p,0} >$$

Wartościowanie $v^{p,k+1}$ jest określone podobnie jak było określone v^p, mianowicie:

- $v^{p,k+1}(par_1) = INT_{v \cup v^{p,0} \cup \ldots \cup v^{p,k}}(e)$,
- $v^{p,k+1}(par_2) = \perp$,
- jeżeli zmienna x jest zadeklarowana w var_p i jej deklaracja ma postać *typ* x, gdzie $typ \in$ {bool, int, char, string}, to $v^{p,k+1}(x) = \perp$,
- jeżeli zmienna x jest zadeklarowana w var_p i jej deklaracja ma postać *typ* $x := t$, to $v^{p,k+1}(x) = INT_{v \cup v^{p,0} \cup \ldots \cup v^{p,k}}(t)$.

Z zakończeniem wykonywania procedury rekurencyjnej wiążą się również dwa aksjomaty. Pierwszy dotyczy ostatecznego wyjścia z procedury i powrotu do głównej części programu:

$$< v \cup v^{p,0}, \mathbf{begin}^{p,0}\ \mathrm{END}\ \mathbf{end}^{p,0};\ ins > \longrightarrow < v_1, ins >$$

gdzie $v_1(y) = v\ [y := v^{p,0}(par_2)]$.

Drugi aksjomat dotyczy wyjścia z $(k+1)$-zagnieżdżonego wykonania procedury i powrotu do k-zagnieżdżonego wykonania procedury:

$$< v \cup v^{p,0} \cup \ldots \cup v^{p,k} \cup v^{p,k+1},$$
$$\mathbf{begin}^{p,0} \ldots \mathbf{begin}^{p,k} \ldots \mathbf{begin}^{p,k+1} \mathrm{END}\ \mathbf{end}^{p,k+1} ins_k \mathbf{end}^{p,k} \ldots ins_0\ \mathbf{end}^{p,0} >$$
$$\longrightarrow$$
$$< v_1 \cup v^{p,0} \cup \ldots \cup v^{p,k},$$
$$\mathbf{begin}^{p,0} \ldots \mathbf{begin}^{p,k} \ldots ins_k\ \mathbf{end}^{p,k} \ldots\ ins_0\ \mathbf{end}^{p,0} >$$

gdzie $(v_1 \cup v^{p,0} \cup \ldots \cup v^{p,k})(y) = v\,[y := v^{p,k+1}(par_2)]$.

Obliczenie programu *JP* jest definiowane podobnie jak programu *PJP*, z tym że uwzględnia kolejne rozszerzenie relacji zmian konfiguracji.

Podsumowując kolejne rozszerzenia języków programowania, łatwo stwierdzić, że

$PROG_{RPJP} \subseteq PROG_{PJP} \subseteq PROG_{JP}$.

Przykład 15.3

Niech będzie dany program pr wykorzystujący procedurę rekursywnego obliczania silni. Program ma dwie zmienne globalne, a procedura – dwa parametry formalne i jedną zmienną lokalną.

```
program pr
    int a;
    int b
    procedure silnia(int in n; int out s)
        int s1
    begin
        if n = 0 then s := 1 else silnia(n – 1, s1); s = n * s1 fi
    end
begin
    read(a);
    silnia(a, b);
    write(b)
end
```

Wartościowanie v_0 w początkowej konfiguracji programu jest określone następująco:

$v_0(a) = \perp$,

$v_0(b) = \perp$.

Pomijając pierwsze zmiany konfiguracji, rozpatrzmy tę, która następuje po wykonaniu instrukcji czytania. Załóżmy dodatkowo, że wczytaną wartością jest liczba 2. Program znajdzie się w konfiguracji:

$<v_2$, silnia(a, b); write(b)>

gdzie:

$v_2(a) = 2$,

$v_2(b) = \perp$.

Zgodnie z aksjomatem wywołania procedury rekursywnej nastąpi przejście do konfiguracji:

$<v \cup v^{silnia,0}$,

begin^{silnia,0}
if $n^0 = 0$ **then** $s^0 := 1$ **else** silnia($n^0 - 1$, $s1^0$); $s^0 = n^0 * s1^0$ **fi**
end^{silnia,0};
write(b) >

gdzie:
$\nu^{silnia,0}(n^0) = 2$,
$\nu^{silnia,0}(s^0) = \perp$,
$\nu^{silnia,0}(s1^0) = \perp$.

Kolejne przejście, bez zmiany wartościowania, nastąpi do konfiguracji:

$<\nu \cup \nu^{silnia,0}$,
begin^{silnia,0}
silnia($n^0 - 1$, $s1^0$); $s^0 = n^0 * s1^0$
end^{silnia,0};
write(b) >

Ponowne zastosowanie aksjomatu wywołania procedury9 rekurencyjnej prowadzi do konfiguracji:

$<\nu \cup \nu^{silnia,0} \cup \nu^{silnia,1}$,
begin^{silnia,0}
begin^{silnia,1}
if $n^1 = 0$ **then** $s^1 := 1$ **else** silnia($n^1 - 1$, $s1^1$); $s^1 = n^1 * s1^1$ **fi**
end^{silnia,1};
$s^0 = n^0 * s1^0$
end^{silnia,0};
write(b) >

gdzie:
$\nu^{silnia,1}(n^1) = 1$,
$\nu^{silnia,1}(s^1) = \perp$,
$\nu^{silnia,1}(s1^1) = \perp$.

Bez zmiany wartościowania następuje przejście do konfiguracji

$<\nu \cup \nu^{silnia,0} \cup \nu^{silnia,1}$,
begin^{silnia,0}
begin^{silnia,1}
silnia($n^1 - 1$, $s1^1$); $s^1 = n^1 * s1^1$
end^{silnia,1};
$s^0 = n^0 * s1^0$
end^{silnia,0};
write(b) >

Kolejne, ostatnie zastosowanie aksjomatu wywołania procedury rekursywnej prowadzi do konfiguracji:

$<\nu \cup \nu^{silnia,0} \cup \nu^{silnia,1} \cup \nu^{silnia,2},$
begin^{silnia,0}
begin^{silnia,1}
begin^{silnia,2}
if $n^2 = 0$ **then** $s^2 := 1$ **else** silnia($n^2 - 1$, sl^2); $s^2 = n^2 * sl^2$ **fi** ;
end^{silnia,2} ;
$s^1 = n^1 * sl^1$
end^{silnia,1} ;
$s^0 = n^0 * sl^0$
end^{silnia,0} ;
write(b) >

gdzie:
$\nu^{silnia,2}(n^2) = 0,$
$\nu^{silnia,2}(s^2) = \perp,$
$\nu^{silnia,2}(sl^2) = \perp.$

Kolejne przejście, bez zmiany wartościowania, następuje do konfiguracji:

$<\nu \cup \nu^{silnia,0} \cup \nu^{silnia,1} \cup \nu^{silnia,2},$
begin^{silnia,0}
begin^{silnia,1}
begin^{silnia,2}
$s^2 := 1$
end^{silnia,2} ;
$s^1 = n^1 * sl^1$
end^{silnia,1} ;
$s^0 = n^0 * sl^0$
end^{silnia,0} ;
write(b) >

Po osiągnięciu tej konfiguracji będzie następować seria powrotów z zagnieżdżonych wywołań, kolejno do:

$<\nu \cup \nu^{silnia,0} \cup \nu^{silnia,1} \cup \nu_1{}^{silnia,2},$
begin^{silnia,0}
begin^{silnia,1}
begin^{silnia,2}
END
end^{silnia,2} ;
$s^1 = n^1 * sl^1$
end^{silnia,1} ;

$s^0 = n^0 * sl^0$
end^{silnia,0};
write(b) >

gdzie:
$\nu_1^{silnia,2}(n^2) = 0$,
$\nu_1^{silnia,2}(s^2) = 1$,
$\nu_1^{silnia,2}(sl^2) = \perp$.

Następnie do konfiguracji:

$<\nu \cup \nu^{silnia,0} \cup \nu_1^{silnia,1}$,
begin^{silnia,0}
begin^{silnia,1}
$s^1 = n^1 * sl^1$
end^{silnia,1};
$s^0 = n^0 * sl^0$
end^{silnia,0};
write(b) >

gdzie:
$\nu_1^{silnia,1}(n^1) = 1$,
$\nu_1^{silnia,1}(s^1) = \perp$,
$\nu_1^{silnia,1}(sl^1) = 1$.

I, podobnie, dalej do:

$<\nu \cup \nu^{silnia,0} \cup \nu_2^{silnia,1}$,
begin^{silnia,0}
begin^{silnia,1}
END
end^{silnia,1};
$s^0 = n^0 * sl^0$
end^{silnia,0};
write(b) >

gdzie:
$\nu_2^{silnia,1}(n^1) = 1$,
$\nu_2^{silnia,1}(s^1) = 1$,
$\nu_2^{silnia,1}(sl^1) = 1$.

Następna zmiana konfiguracji daje:

$<\nu \cup \nu_1^{silnia,0}$,
begin^{silnia,0}
$s^0 = n^0 * sl^0$
end^{silnia,0};
write(b) >

gdzie:

$v_1^{silnia,0}(n^0) = 2,$
$v_1^{silnia,0}(s^0) = \perp,$
$v_1^{silnia,0}(s1^0) = 1.$

Stąd otrzymuje się konfigurację:

$<v \cup v^{silnia,0},$
begin^{silnia,0}
END
end^{silnia,0};
write(b) >

gdzie:

$v_1^{silnia,0}(n^0) = 2,$
$v_1^{silnia,0}(s^0) = 1,$
$v_1^{silnia,0}(s1^0) = 1.$

Po ostatecznym wyjściu z procedury przechodzi się do konfiguracji <v , write(b) >, gdzie:

$v_1(a) = 2,$
$v_1(b) = 2.$

po czym osiąga się konfigurację końcową <v , END>.

Ćwiczenia

1. Przedstawić gramatyki bezkontekstowe dla omawianych języków *BPJP*, *PJP* oraz *PJP*.
2. Rozszerzyć język *BPJP* o nowe typy danych:
   a) typ wyliczeniowy,
   b) tablicowy,
   c) rekordowy.
3. Przedstawić semantykę operacyjną nowych instrukcji pętli:
   a) **for** i:=1 **to** n **do** *ins* **od**
   b) **repeat** *ins* **until** α **end**
4. Przedstawić obliczenie programu:

```
program sk
    int n;
    int s := 0;
    int k := 1;
```

```
begin
    read(n);
    while k < n do s := s + k * k ; k := k + 1 od;
    write(s)
end
```

5. Określić relację, która określa związek pomiędzy danymi czytanymi przez program a danymi pisanymi przez program:

```
program qq
    int n;
    int p := 1;
begin
    read(n);
    p := 0;
    while n ≠ 0 do p := p * n ; n := n – 1 od;
    write(p)
end
```

6. Dana jest definicja procedury:

```
procedure sum(int in n; int out s)
begin
    if n = 0 then s := 0 else sum(n – 1, s1); s := s + s1 fi
end
```

Przedstawić obliczenia dla wywołań procedury sum:

sum(1, x), sum(2, y), sum(3, z).

7. Dana jest definicja procedury:

```
procedure pr(int in m; int in n; int out k)
    int p := 0
begin
    k := 0;
    while n < 2 * m do k := k + m; n := n + 1 od
end
```

Określić relację pomiędzy formalnymi parametrami wejściowymi i parametrami wyjściowymi. Określić tę relację, gdy warunek w instrukcji iteracji ma postać: $n \neq 2 * m$.

16. Logika programów Hoare'a

16.1. Programy ze specyfikacją

Program jest zapisem algorytmu w postaci jednoznacznie interpretowanej przez komputer. Interpretacja ta wyraża się przez obliczenie, które dla ustalonego zestawu danych wejściowych dostarcza pewnego zestawu danych wyjściowych. Od programu oczekuje się, że jest on poprawny, to znaczy że dostarczane wyniki jego obliczeń są zgodne z oczekiwanymi. Jak określić, jakie są oczekiwania od programu, czyli „co program ma liczyć", i jak stwierdzać, czy dany program spełnia te oczekiwania, czy „liczy to, co należy" – oto dwa zasadnicze pytania, na które musi odpowiadać programista.

Określenie tego – co program ma liczyć – nazywa się specyfikacją programu. Specyfikacja może być wyrażana w różny sposób. Najprostszą formą jest specyfikacja słowna, wyrażona w języku naturalnym, na przykład:

Program ma obliczać pierwiastek z danej liczby.
Program ma obliczać największy wspólny podzielnik dwóch liczb.
Program ma mnożyć dwie macierze.

Z każdym przykładem wiążą się pewne niejasności, na przykład:

Dla jakich liczb i z jaką dokładnością ma być obliczony pierwiastek?
Dla jakich liczb ma być obliczony największy wspólny podzielnik?
Jakie mają być rozmiary i jakiego typu mają być elementy mnożonych macierzy?

Jednym ze sposobów precyzyjnego sformułowania specyfikacji programów jest użycie pary formuł, nazywanych *warunkiem wstępnym* i *warunkiem końcowym*. Warunek wstępny i warunek końcowy będą oznaczane przez *pre* i *post*. Są one formułami ustalonego języka rachunku kwantyfikatorów. Język ten będzie oznaczany przez *SPEC*. Para warunków <*pre*, *post*> stanowi specyfikację programu, czyli sformułowanie własności oczekiwanych od programu. Specyfikacja taka powinna być sformułowana przed powstaniem programu, po to, aby programista wiedział, co ma napisać, natomiast – gdy program już powstanie – pojawia się pytanie: czy program jest zgodny z tą specyfikacją. Inaczej: czy program jest poprawny względem specyfikacji.

Na dalsze potrzeby zakłada się, że jako język programowania będzie przyjęty język *JP*, zdefiniowany w rozdziale poprzednim.

Trójka postaci

$\{pre\}\ prog\ \{post\}$

gdzie $pre, post \in SPEC$, $prog \in PROG_{JP}$, będzie nazywana programem ze specyfikacją. Ponieważ program składa się z instrukcji, każda instrukcja składowa *ins* może mieć własną specyfikację, która w pewien sposób wynika ze specyfikacji całego programu. Można zatem rozpatrywać trójki:

$\{pre_{ins}\}\ ins\ \{post_{ins}\}$

wyrażające częściową poprawność instrukcji $ins \in INSTR_{JP}$ względem warunków pre_{ins} i $post_{ins}$.

Programowi ze specyfikacją przypisuje się jedno z dwóch znaczeń, nazywane – odpowiednio – częściową i całkowitą poprawnością programu. Mówi się też inaczej, że program jest częściowo lub całkowicie zgodny ze swoją specyfikacją.

Definicja 16.1

Częściowa poprawność programu *prog* względem pary warunków *pre* i *post* oznacza zachodzenie następującej własności:

Dla każdego wartościowania początkowego v_0 zmiennych programu spełniającego warunek wstępny *pre*, jeśli program *prog* zakończy się pomyślnie (nie ulegnie zerwaniu i nie zapętli się), to wartościowanie końcowe v_k zmiennych programu spełnia warunek końcowy *post*.

Częściowa poprawność programu *prog* oznacza, że jeśli $INT_{v_0}(pre) = \mathbf{P}$, to jeśli tylko jego obliczenie ma postać

$$Comp(prog) = \langle v_0, ins_0\rangle \longrightarrow \langle v_1, ins_1\rangle \longrightarrow \ldots \longrightarrow \langle v_k, \text{END}\rangle$$

to $INT_{v_k}(post) = \mathbf{P}$.

Należy zwrócić uwagę, że z podanej definicji częściowej poprawności wynika, że program, którego obliczenie ulegnie zerwaniu lub się nie zakończy, jest częściowo poprawny względem dowolnej specyfikacji! Dowolny program *prog* jest również częściowo poprawny względem każdej specyfikacji, w której $pre \equiv \mathbf{false}$ lub $post \equiv \mathbf{true}$.

Definicja 16.2

Całkowita poprawność programu *prog* względem pary warunków *pre* i *post* oznacza zachodzenie następującej własności:

Dla każdego wartościowania początkowego v_0 zmiennych programu spełniającego warunek wstępny *pre*, program *prog* kończy się pomyślnie (nie ulega zerwaniu i nie pętli się), a wartościowanie końcowe v_k zmiennych programu spełnia warunek końcowy *post*.

Oznacza to, że jeśli $INT_{v_0}(pre) = \mathbf{P}$, to obliczenie programu się kończy, czyli

$$Comp(prog) = \langle v_0, ins_0\rangle \longrightarrow \langle v_1, ins_1\rangle \longrightarrow \ldots \longrightarrow \langle v_k, \text{END}\rangle$$

oraz $INT_{v_k}(post) = \mathbf{P}$.

Całkowita poprawność programu oznacza gwarancję, że obliczenie programu zawsze się kończy, jeśli tylko wartościowanie początkowe zmiennych programu spełnia warunek wstępny *pre*.

Całkowita poprawność programu pociąga, oczywiście, poprawność częściową względem danej specyfikacji.

Nasuwa się pytanie o związek pomiędzy językiem specyfikacji *SPEC* a językiem programowania, bowiem częścią języka *JP* jest zbiór formuł $FORM(V)$.

Zbiór formuł $FORM(V)$ języka programowania *JP* nie zawiera kwantyfikatorów. Zbiór ten jest podzbiorem formuł języka rachunku kwantyfikatorów, oznaczanego tutaj przez $JRK(F_{JP}, P_{JP}, V)$, nad sygnaturą $Sig_{JP} = \langle F_{JP}, P_{JP}\rangle$, gdzie F_{JP} i P_{JP} są zbiorami symboli funkcyjnych i predykatywnych języka programowania *JP* oraz zbiorem zmiennych V, zatem $FORM(V) \subseteq JRK(F_{JP}, P_{JP}, V)$.

Język specyfikacji *SPEC* jest również zbiorem formuł języka rachunku kwantyfikatorów $JRK(F_{SPEC}, P_{SPEC}, V)$ nad pewną sygnaturą $Sig_{SPEC} = \langle F_{SPEC}, P_{SPEC}\rangle$ i zbiorem zmiennych V. W szczególnym przypadku jako język specyfikacji *SPEC* może być wybrany język $JRK(F_{JP}, P_{JP}, V)$. Dla ułatwienia dalszych rozważań przyjmuje się założenie, że język specyfikacji *SPEC* jest co najmniej tak bogaty jak język $JRK(F_{JP}, P_{JP}, V)$.

16.2. System dowodzenia poprawności częściowej

Budowa poprawnych programów opiera się na dwóch podejściach: podejściu konstruktywnym i weryfikacyjnym. Podejście konstruktywne polega na takim ciągu przekształceń specyfikacji programu w konstrukcje programowe, w którym każde przekształcenie – na mocy konstrukcji – jest poprawne, zapewniając zachowanie zgodności ze specyfikacją. Takie podejście jest przedstawiane na przykład w książkach: [Dijkstra 1978] i [Cook 2005]. W tym rozdziale omawia się tylko drugie podejście, które polega na weryfikacji (badaniu poprawności) gotowego programu względem specyfikacji.

Jeden ze sposobów badania poprawności programu opiera się na systemie wnioskowania Hoare'a. Jak każdy system wnioskowania składa się on z zestawu aksjomatów i reguł, których elementami są instrukcje ze specyfikacjami. Przedstawiony niżej system odnosi się do wnioskowania o częściowej poprawności programów w języku *BPJP*.

Aksjomat instrukcji skip

$\{pre\}$ skip $\{pre\}$

Aksjomat wyraża to, że instrukcja nie zmienia wartościowana zmiennych programu: jeżeli warunek *pre* jest spełniony przed wykonaniem instrukcji, to jest, oczywiście, spełniony również po wykonaniu tej instrukcji.

Aksjomat instrukcji pisania

$\{pre\}$ write(x) $\{pre\}$

Instrukcja pisania również nie zmienia wartościowania zmiennych programu, zatem jest w tym sensie równoważna instrukcji skip. Instrukcja daje, oczywiście, dodatkowy efekt, którym jest przekazanie wartości zmiennej x do otoczenia programu, za pośrednictwem pewnego urządzenia piszącego.

Aksjomat instrukcji przypisania

$\{post[\mathrm{x} ::= t]\}\ \mathrm{x} := t\ \{post\}$ gdzie $t \in TERM(\mathrm{V})$

Rezultatem instrukcji przypisania jest zmiana wartościowania zmiennej x. Jeśli v jest wartościowaniem przed wykonaniem instrukcji, to po jej wykonaniu nowym wartościowaniem jest $v[\mathrm{x} := INT_v(t)]$. Jeśli po zakończeniu instrukcji jest spełniony warunek *post*, czyli $INT_{v[x:=INT_v(t)]}(post) = \mathbf{P}$, to $INT_v(post[\mathrm{x} ::= t] = \mathbf{P}$ – zob. lemat 11.2.

Aksjomat instrukcji czytania

$\{post[\mathrm{x} ::= t]\}$ read(x) $\{post\}$ gdzie $t \in TERM(\varnothing)$

Instrukcja czytania daje wynik podobny do instrukcji przypisania, ale istotne jest to, że wartość przypisana zmiennej x nie jest z góry określona. Może to być dowolna wartość ze zbioru wartości odpowiadających typowi zmiennej x. Instrukcja, z punktu widzenia wykonawcy programu, jest niedeterministyczna, gdyż wartość przypisana zmiennej zależy od otoczenia programu. Po wykonaniu tej instrukcji program może mieć różne obliczenia, które zależą od wartości wczytanej zmiennej.

Reguła dla instrukcji warunkowej

$$\frac{\{pre \wedge \alpha\}\ ins_1\ \{post\},\ \{pre \wedge \neg\alpha\}\ ins_2\ \{post\}}{\{pre\}\ \textbf{if}\ \alpha\ \textbf{then}\ ins_1\ \textbf{else}\ ins_2\ \textbf{fi}\ \{post\}}$$

Reguła ma dwie przesłanki odnoszące się do poprawności częściowej instrukcji składowych ins_2 oraz ins_2. Sumą warunków wstępnych dla obu instrukcji składowych jest

formuła *pre*, która jest warunkiem wstępnym całej instrukcji warunkowej. (W przypadku instrukcji warunkowej o konkretnie podanej składni i semantyce, warunki wstępne dla instrukcji składowych są rozłączne.) Ponieważ warunkami końcowymi obu instrukcji jest *post*, zatem instrukcja warunkowa jest poprawna względem warunków *pre* i *post*.

Reguła dla instrukcji iteracji

$$\frac{\{pre \wedge \alpha\}\ ins\ \{pre\}}{\{pre\}\ \textbf{while}\ \alpha\ \textbf{do}\ ins\ \textbf{od}\ \{pre \wedge \neg\alpha\}}$$

Warunek *pre* nazywa się niezmiennikiem instrukcji iteracji, co oznacza, że jest on spełniony w wyróżnionym miejscu instrukcji iteracji, po każdym wykonaniu instrukcji *ins*. Postać warunku wstępnego w przesłance reguły zakłada, że instrukcja *ins* wykonywana, gdy spełniony jest dozór α, gwarantuje, że warunek α jest spełniony po jej wykonaniu. Postać warunku końcowego dla całej instrukcji wynika z kolei z faktu, że spełnienie $\neg\alpha$ oznacza koniec wykonywania całej instrukcji.

Pojęcie *niezmiennika* jest ogólniejsze, gdyż można odnosić je do dowolnej instrukcji: warunek *inv* jest niezmiennikiem instrukcji *ins* w wyróżnionym jej miejscu przy warunku początkowym *init*, jeżeli dla każdego obliczenia instrukcji z wartościowaniem początkowym spełniającym *init*, za każdym razem, gdy obliczenie dochodzi do wyróżnionego miejsca instrukcji *ins*, aktualne wartościowanie zmiennych spełnia przypisany temu miejscu warunek *inv*.

Reguła dla sekwencyjnego złożenia instrukcji

$$\frac{\{pre\}\ ins_1\ \{midl\},\ \{midl\}\ ins_2\ \{post\}}{\{pre\}\ ins_1; ins_2\ \{post\}}$$

Reguła konsekwencji

$$\frac{pre \Rightarrow pre_1,\ \{pre_1\}\ ins\ \{post_1\},\ post_1 \Rightarrow post}{\{pre\}\ ins\ \{post\}}$$

Reguła konsekwencji pozwala na zawężanie warunków wstępnych i osłabianie warunków końcowych. Specyfiką reguły jest to, że zawiera dwie przesłanki w postaci instrukcji ze specyfikacją, ale w postaci implikacji, których spełnienie bądź niespełnienie nie może być wnioskowane w ramach danego systemu wnioskowania o poprawności częściowej programów. Można o tym wnioskować na przykład przez wykorzystanie jednego z przedstawianych systemów dowodzenia dla rachunku predykatów.

Reguła podstawienia dla dowolnej instrukcji

$$\frac{\{pre\}\ ins\ \{post\}}{\{pre[y ::= t]\}\ ins\ \{post[y := t]\}}$$

dla zmiennej *y*, która nie występuje w instrukcji *ins*, i dla termu *t*, który nie zawiera zmiennych występujących w tej instrukcji. Reguła ma charakter pomocniczy, przydaje się do orzekania o poprawności częściowej względem pary warunków różniących się od *pre* i *post* zastąpieniem zmiennej *y* dowolnym termem.

Reguła dla programu

$$\frac{\{pre\}\ ins\ \{post\}}{\{pre\}\ \textbf{program}\ p\ var\ \textbf{begin}\ ins\ \textbf{end}\ \{post\}}$$

Reguła stwierdza, że częściowa poprawność programu oznacza częściową poprawność jego treści.

Przedstawione wyżej aksjomaty i reguły odnoszą się do języka *BPJP*.

Przykład 16.1

Łatwo sprawdzić, że poprawne częściowo są niżej podane instrukcje przypisania względem odpowiadających im specyfikacji:

$\{x > 9\}\ \ x := x + 1\ \{x > 10\}$
$\{y > 9\}\ x := y + 1\ \{x > 10\}$
$\{x + y > 0\} x := x + y\ \{x > 0\}$

Czytelnikowi pozostawia się sprawdzenie, że poprawne częściowo są niżej podane instrukcje warunkowe względem odpowiadających im specyfikacji:

$\{x > 0\}$ **if** $x > 0$ **then** $x := x + 10$ **else** $x := x + 5$ **fi** $\{x > 10\}$
$\{x > 0\}$ **if** $x > 5$ **then** $x := x + 5$ **else** $x := x + 10$ **fi** $\{x > 10\}$

Bardziej złożone jest sprawdzenie poprawności częściowej dla instrukcji iteracji względem specyfikacji:

$\{s = 0 \wedge i = m\}$
while $i \leq n$ **do** $s := s + i\ ;\ i := i + 1$ **od**
$\{s = \sum_{k=m}^{n} k \wedge i = n + 1\}$

Zauważmy, że niezmiennikiem dla instrukcji iteracji jest warunek

$$pre \equiv (s = \sum_{k=m}^{i-1} k \wedge i \leq n + 1)$$

Można to sprawdzić, posługując się aksjomatami dla instrukcji przypisania

$s := s + i\ ;\ i := i + 1$

stanowiących treść instrukcji iteracji.

Druga z tych instrukcji jest częściowo poprawna względem specyfikacji

$$\{s = \sum_{k=m}^{i} k \wedge i \leq n\}$$
$$i := i + 1$$
$$\{s = \sum_{k=m}^{i-1} k \wedge i \leq n+1\}$$

Dla pierwszej z nich zachodzi:

$$\{s = \sum_{k=m}^{i} k \wedge i \leq n\}$$
$$s := s + i$$
$$\{s = \sum_{k=m}^{i-1} k \wedge i \leq n\}$$

Zauważmy, że

$$pre \wedge (i \leq n) \equiv s = \sum_{k=m}^{i-1} k \wedge i \leq n$$

oraz

$$pre \wedge \neg(i \leq n) \equiv s = \sum_{k=m}^{n} k \wedge i = n+1$$

co dowodzi częściowej poprawności instrukcji iteracji względem podanej specyfikacji.

Następne reguły są związane z instrukcjami wywołania procedur nierekursywnych – język *PJP* – oraz rekursywnych – język *JP*.

Reguła wywołania procedury nierekursywnej

$$\frac{\{pre\}\ a_1 := e_1; \ldots; a_n := e_n; ins_p; x_1 := b_1; \ldots; x_m := b_m\ \{post\}}{\{pre\}\ p(e_1, \ldots, e_n, x_1, \ldots, x_m)\ \{post\}}$$

gdzie procedura p ma definicję:

p(*typ*$_1$ **in** a_1; ... ; *typ*$_n$ **in** a_n; *typ*$_{n+1}$ **out** b_1; ... ; *typ*$_{n+m}$ **out** b_m)
  var$_p$
begin
  ins$_p$
end

Ciąg instrukcji

$$a_1 := e_1; \ldots; a_n := e_n; ins_p; x_1 := b_1; \ldots; x_m := b_m$$

stanowi zmodyfikowaną treść definicji procedury.

Modyfikacja polega na dołączeniu przed ins_p ciągu instrukcji przypisania, które odzwierciedlają komunikację parametrów wejściowych, oraz na dołączeniu po ins_p ciągu instrukcji przypisania, które odzwierciedlają komunikacją parametrów wyjściowych. Efektem pierwszej grupy dołączonych instrukcji przypisania jest modyfikacja wartościowania wejściowych parametrów formalnych procedury, a efektem drugiej grupy – modyfikacja wartościowania wyjściowych parametrów aktualnych.

Reguła wywołania procedury rekursywnej, dla procedury o takiej samej definicji ma postać

$$\frac{\{pre\}\ \mathrm{p}(e_1, \ldots, e_n, \mathrm{x}_1, \ldots, \mathrm{x}_m)\ \{post\} \quad \vdash\ \{pre\}\ \mathrm{a}_1 := e_1; \ldots; \mathrm{a}_n := e_n; ins_p; \mathrm{x}_1 := \mathrm{b}_1; \ldots; \mathrm{x}_m := \mathrm{b}_m\ \{post\}}{\{pre\}\ \mathrm{p}(e_1, \ldots, e_n, \mathrm{x}_1, \ldots, \mathrm{x}_m)\ \{post\}}$$

Ostatnia reguła różni się od pozostałych postacią przesłanki. Sens przesłanki jest następujący: istnieje dowód na to, że:

jeżeli zachodzi

$$\{pre\}\ \mathrm{p}(e_1, \ldots, e_n, \mathrm{x}_1, \ldots, \mathrm{x}_m)\ \{post\}$$

to zachodzi

$$\{pre\}\ \mathrm{a}_1 := e_1; \ldots; \mathrm{a}_n := e_n; ins_p; \mathrm{x}_1 := \mathrm{b}_1; \ldots; \mathrm{x}_m := \mathrm{b}_m\ \{post\}$$

Symbol $\vdash$ oznacza tu fakt istnienia dowodu.

Sens całej reguły jest następujący: instrukcja wywołania procedury rekursywnej jest częściowo poprawna względem danej specyfikacji, czyli:

$$\{pre\}\ \mathrm{p}(e_1, \ldots, e_n, \mathrm{x}_1, \ldots, \mathrm{x}_m)\ \{post\}$$

jeśli na postawie założenia takiej poprawności można dowieść częściowej poprawności instrukcji stanowiącej zmodyfikowaną treść definicji procedury.

Przedstawiony system dowodzenia częściowej poprawności programów jest semantycznie poprawny, nie jest natomiast semantycznie zupełny. Dowód poprawności, a także powody braku zupełności są przedstawione w książkach: [Dembiński, Małuszyński 1981], [Apt, Olderog 1991].

Przykład 16.2

Dana jest rekursywna procedura obliczająca największy wspólny dzielnik dwóch liczb, oparta na algorytmie Euclidesa:

```
procedure NWD(int in x; int in y; int out z)
    int r ;
begin
```

```
    r := x mod y;
    if r = 0 then z := y else NWD(y, r, z) fi
end
```

Zakłada się, że dla wywołania procedury zachodzi:

$\{a > 0 \wedge b > 0\}$ NWD(a, b, c) $\{c = nwd(a, b)\}$

gdzie nwd(a, b) jest funkcją, która określa największy wspólny dzielnik liczb a oraz b. Należy zbadać, czy na podstawie przyjętego założenia zachodzi:

```
{a > 0 ∧ b > 0}
    x := a; y := b;
    r := x mod y;
    if r = 0 then z := y else NWD(y, r, z) fi ;
    c := z
{c = nwd(a, b)}
```

Łatwo stwierdzić, że poprawne częściowo są fragmenty programu, wyróżnione boczną linią, względem przedstawionych specyfikacji:

```
{a > 0 ∧ b > 0}
  x := a; y := b;
{x > 0 ∧ y > 0}
    r := x mod y;
    if r = 0 then z := y else NWD(y, r, z) fi ;
{z = nwd(x, y)}
  c := z
{c = nwd(a, b)}
```

Pozostaje zbadanie, czy poprawny jest fragment:

```
{x > 0 ∧ y > 0}
    r := x mod y;
    if r = 0 then z := y else NWD(y, r, z) fi ;
{z = nwd(x, y)}
```

Wykazanie tego sprowadza się do dwóch przypadków:

Jeżeli r = 0, to oczywiście wartość z jest największym wspólnym dzielnikiem liczb x oraz y, czyli z = nwd(x, y).

Jeżeli $r \neq 0$, to korzysta się z twierdzenia arytmetyki:

jeżeli r jest resztą z dzielenia całkowitoliczbowego x przez y, to

nwd(x, y) = nwd(y, r),

zatem i w tym przypadku z = nwd(x, y) = nwd(y, r).

16.3. Dowodzenie poprawności całkowitej

Poprawność całkowita programu oznacza, że program spełnia następujące warunki:

- jest częściowo poprawny względem danej specyfikacji,
- jego obliczenie nie ulegnie zerwaniu,
- jego obliczenie nie będzie nieskończone.

W dalszej części rozdziału będzie omówione tylko jak można badać, czy obliczenie programu jest nieskończone, inaczej: czy program ma własność stopu. Przyczyną nieskończonych obliczeń może być pętlenie się instrukcji iteracyjnej. Badanie stopu instrukcji iteracji można prowadzić między innymi metodą liczników iteracji i metodą malejących wielkości [Banachowski, Kreczmar 1982] oraz funkcji zmniejszającej [Dijkstra 1978]. Poniżej jest przedstawiona tylko pierwsza z tych metod – metoda liczników iteracji.

Niech dana będzie instrukcja iteracji standardowej postaci

while α **do** ins **od**

Zakłada się, że się jej obliczenie rozpoczyna się przy wartościowaniu spełniającym ustalony warunek początkowy *init*. Załóżmy ponadto, że *inv* jest niezmiennikiem instrukcji przy warunku początkowym *init*.

Metoda liczników polega na wstawieniu do treści instrukcji iteracji dodatkowej zmiennej, innej od wszystkich innych zmiennych występujących w treści instrukcji, która zwiększa swoją wartość o jeden po każdorazowym wykonaniu instrukcji *ins*:

n = 0; **while** α **do** *ins*; n := n + 1 **od**

a następnie na określeniu wyrażenia całkowitoliczbowego *up*, którego wartość w trakcie obliczenia programu ograniczałaby wartość zmiennej n.

Twierdzenie 16.1

Jeżeli $inv \wedge (n \leq up)$ jest niezmiennikiem rozszerzonej instrukcji iteracji, to instrukcja iteracji w standardowej postaci ma własność stopu.

Dowód jest przedstawiony w książce [Banachowski, Kreczmar 1982].

Często instrukcja iteracji zawiera instrukcje, które mogą pełnić rolę licznika pętli. W takich przypadkach pozostaje zadanie ustalenia wyrażenia *up*, wyznaczającego górne ograniczenie licznika. Rozpatrzmy prosty przykład.

Przykład 16.3

Dana jest procedura obliczająca wynik i resztę dzielenia całkowitoliczbowego:

procedure dc(int **in** x; int **in** y; int **out** q; int **out** r)

$\{x \geq 0 \wedge y > 0\}$
begin
  q := 1;
  r := x;
  while $y \leq r$ **do** q := q + 1; r := r – y **od**
end
$\{x = q * y + r \wedge 0 \leq r < y \wedge x \geq 0 \wedge y > 0\}$

Łatwo stwierdzić, że program jest częściowo poprawny względem przedstawionej specyfikacji. Podobnie można sprawdzić, że warunek

$$\varphi \equiv x = q * y + r \wedge 0 \leq r \wedge x \geq 0 \wedge y > 0$$

jest niezmiennikiem instrukcji iteracji. Kandydatem na licznik jest zmienna q. Wartość przypisywana zmiennej q jest ograniczona przez x/y. Aby stwierdzić, że instrukcja iteracji ma własność stopu należy pokazać, że warunek

$$\varphi \wedge (q \leq x/y)$$

jest niezmiennikiem instrukcji iteracji. Ponieważ już wiadomo, że φ jest niezmiennikiem po to, aby pokazać, że $\varphi \wedge (q \leq x/y)$ jest niezmiennikiem, wystarczy pokazać (por. zad. 4.), że $(q \leq x/y)$ jest niezmiennikiem, ale $(q \leq x/y)$ wynika z φ, mianowicie:

$$(x = q * y + r \wedge y > 0) \Rightarrow (q = x/y - r/y) \Rightarrow (q \leq x/y)$$

Według twierdzenia 16.1 można stwierdzić, że procedura ma własność stopu, z czego – na podstawie częściowej poprawności procedury – wynika, że procedura jest całkowicie poprawna względem podanej specyfikacji.

Ćwiczenia

1. Sformułować regułę wnioskowania poprawności częściowej dla instrukcji iteracji postaci: **repeat** *ins* **until** α.
2. Udowodnić częściową poprawność ciągu instrukcji względem podanej specyfikacji:
   a) $\{x = y*q + r\}$ r := r – y; q := q + 1 $\{x = y*q + r\}$
   b) $\{x \geq 0 \wedge y > 0\}$
      q := 0; r := x;
      while $r \geq y$ **do** r = r – y; q = q + 1 **else** skip **od**
   $\{x = y * q + r \wedge r \geq 0 \wedge y > 0 \wedge r < y\}$

3. Uzasadnić aksjomat Floyda

$$\{\alpha\}\ x := e\ \{\exists y \bullet (\alpha[x ::= y] \wedge x = e[x ::= y])\}$$

4. Niech warunki inv_1 oraz inv_2 będą niezmiennikami danej instrukcji iteracji względem tego samego warunku początkowego *init*. Czy warunki $inv_1 \wedge inv_2$ oraz $inv_1 \vee inv_2$ będą niezmiennikami tej instrukcji względem warunku początkowego *init*?

5. Sformułować specyfikację procedury, która bada, czy dana liczba typu int jest liczbą parzystą. W języku *PJP* napisać tę procedurę i udowodnić jej częściową oraz całkowitą poprawność względem utworzonej specyfikacji.

6. Określić, co jest celem obliczeń podanego niżej programu? Sprawdzić, czy program ten jest zgodny z przedstawionymi specyfikacjami wewnętrznymi.

```
program bp
   int x; int y; int z; int n; int m;
begin
   read(x); read(y);
{n ≥ 0}
   z := x; y := 1; m := n;
   repeat
   {x^n = y * z^m ∧ m ≥ 0}
      if nieparzysta(m) then y := y * z else skip fi;
      m := m div 2;
      z := z * z;
   {x^n = y * z^m ∧ m ≥ 0}
   until m =0;
   write(y);
{y = x^n }
end
```

7. Przedstawić specyfikację procedury obliczającej silnię, następnie pokazać jej częściową poprawność względem tej specyfikacji. Czy procedura jest poprawna całkowicie? Jak określić warunek gwarantujący brak zerwania obliczeń i warunek stopu:

```
procedure silnia(int in n; int out s)
   int s1
begin
   if n = 0 then s := 1 else silnia(n – 1, s1); s = n * s1 fi
end
```

Literatura

ADAMOWICZ Z., ZBIERSKI P., 1991, *Logika matematyczna*, PWN.

APT K., OLDEROG E.R., 1991, *Verification of Sequential and Concurrent Programs*, Springer.

ARBIB M.A., 1968, *Mózg, maszyna, matematyka*, PWN.

BANACHOWSKI L., KRECZMAR A., 1982, *Elementy analizy algorytmów*, WNT.

BANERJI P.B. (ed.), 1990, *Formal Techniques in Artificial Intelligence. A Sourcebook*, North-Holland.

BEN-ARI M., 2001, *Mathematical Logic for Computer Science*, Springer (tłum. na polski: *Logika matematyczna w informatyce*, WNT, 2005).

BICARREGUI J.C., FITZGERALD J.S., LINDSAY P.A., MOORE R., RITCHIE B., 1994, *Proof in VDM: A Practitioners Guide*, Springer.

BOCHEŃSKI J.M., 1992, *Współczesne metody myślenia*, Wydawnictwo „W drodze".

BOCHEŃSKI J.M., 1993, *Logika i filozofia. Wybór pism*, PWN.

BOLC L., BORODZIEWICZ W., WÓJCIK M., 1991, *Podstawy przetwarzania informacji niepewnej i niepełnej*, PWN.

BORZYSZKOWSKI A.M., SOKOŁOWSKI S., 1995, *Matematyczne podstawy informatyki*, EFP, Poznań.

BUBNICKI Z., 1990, *Wstęp do systemów ekspertowych*, PWN.

CARNAP R., 1990, *Logiczna składnia języka*, PWN.

COOK J., 2005, *Constructing Correct Software*, Springer.

DAVIS P.J., HERSH R., 1994, *Świat matematyki*, PWN.

DEMBIŃSKI P., MAŁUSZYŃSKI J., 1981, *Matematyczne metody definiowania języków programowania*, WNT.

DIJKSTRA E.W., 1978, *Umiejętność programowania*, WNT.

EHRIG H., MAHR B., 1985, *Fundamentals of Algebraic Specifications*, Springer.

FITTING M., 1990, *First-order logic and automated theorem proving*, Springer.

GABBAY D., 1998, *Elementary Logics: A procedural perspective*, Prentice Hall.

GUZICKI W., ZAKRZEWSKI P., 2005, *Wykłady ze wstępu do matematyki. Wprowadzenie do teorii mnogości*, PWN.

GUZICKI W., ZAKRZEWSKI P., 2005, *Wykłady ze wstępu do matematyki. Zbiór zadań*, PWN.

HOPCROFT J.E., ULLMAN J.D., 2003, *Wprowadzenie do teorii automatów, języków i obliczeń*, PWN.

HUNTER G., 1982, *Metalogika*, PWN.

HARRISON M.A., 1973, *Wstęp do teorii sieci przełączających i automatów*, PWN.

HUZAR Z., 1989, *Programowanie procesów komunikujących się w czasie rzeczywistym*, Prace Naukowe Centrum Obliczeniowego Politechniki Wrocławskiej 8, Seria: Monografie 1, Wydawnictwo Politechniki Wrocławskiej.

HUZAR Z., KURZYŃSKI M., SAS J., 1994, *Rule-Based Pattern Recognition with Learning*, Oficyna Wydawnicza Politechniki Wrocławskiej.

GERSTING J.L., 1993, *Mathematical Structures for Computer Science*, Computer Science Press.

GRZEGORCZYK A., 1975, *Zarys logiki matematycznej*, PWN.

GRZEGORCZYK A., 1983, *Zarys arytmetyki teoretycznej*, PWN.
KACPRZYK J., 1986, *Zbiory rozmyte w analizie systemowej*, PWN.
KELLY J., 1997, *The Essence of Logic*, Prentice Hall.
KLIMEK R., 1999, *Wprowadzenie do logiki temporalnej*, Wydawnictwa AGH.
KOWALSKI R., 1989, *Logika w rozwiązywaniu zadań*, WNT.
KOTARBIŃSKI T., 1985, *Wykłady z dziejów logiki*, PWN.
KRASZEWSKI J., 2007, *Wstęp do matematyki*, WNT.
LYNDON R.C., 1978, *O logice matematycznej*, PWN.
ŁAWROW I.A., MAKSIMOWA Ł.L., 2004, *Zadania z teorii mnogości, logiki matematycznej i teorii algorytmów*, PWN.
MANNA Z., PNUELI A., 1992, *Temporal Verification of Reactive Systems: Specification*, Springer.
MANNA Z., PNUELI A., 1995, *Temporal Verification of Reactive Systems: Verification*, Springer.
MARCISZEWSKI W. (red.), 1987, *Logika formalna. Zarys encyklopedyczny z zastosowaniem do informatyki i lingwistyki*, PWN.
MARCISZEWSKI W. (red.), 1988, *Mała encyklopedia logiki*, Ossolineum.
MAREK W., ONYSZKIEWICZ J., 1975, *Elementy logiki i teorii mnogości w zadaniach*, PWN.
MIRKOWSKA G., SALWICKI A, 1987, *Algorithmic logic*, PWN.
MORTIMER H., 1982, *Logika indukcji*, PWN.
MOSTOWSKI A.W., PAWLAK Z., 1970, *Logika dla inżynierów*, PWN.
MURAWSKI R., 1995, *Filozofia matematyki. Zarys dziejów*, PWN.
NISSANKE N., 1999, *Introductory Logic and Sets Theory for Computer Scientists*, Addison-Wesley Longam.
PACHOLSKI L., 2004, *Logika dla informatyków – materiały do zajęć*, Uniwersytet Wrocławski, Instytut Informatyki.
PAWLAK Z., 1991, *Rough Sets, Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers.
PENROSE R., 1996, *Nowy umysł cesarza – o komputerach, umyśle i prawach fizyki*, PWN.
POGORZELSKI W., 1981, *Klasyczny rachunek kwantyfikatorów. Zarys teorii*, PWN.
RASIOWA H., 1998, *Wstęp do matematyki współczesnej*, PWN.
RUTKOWSKA D., PILIŃSKI M., RUTKOWSKI L., 1997, *Sieci neuronowe, algorytmy genetyczne i systemy rozmyte*, PWN.
SHEPARD D., *An Introduction to Formal Specification with Z and VDM*, McGraw-Hill, 1995.
SŁUPECKI J., HAŁKOWSKA K., PIRÓG-RZEPECKA K., 1978, *Logika i teoria mnogości*, PWN.
SOCHER-AMBROSIUS R., JOHANN P., 1996, *Deduction Systems*, Springer.
STANOSZ B., 2002, *Ćwiczenia z logiki*, PWN.
SZAŁAS A., 1992, *Zarys dedukcyjnych metod automatycznego wnioskowania*, Akademicka Oficyna Wydawnicza RM.
TIURYN J., 2003, *Wstęp do teorii mnogości i logiki*, Uniwersytet Warszawski, Wydział Matematyki, Informatyki i Mechaniki.
TURSKI W.M., 1988, *Logiki nieklasyczne (dla informatyka pracującego)*, materiały V Jesiennej Szkoły PTI, Polskie Towarzystwo Informatyczne, 103–132.
WÓJCICKI R., 1982, *Wykłady z metodologii nauk*, PWN.
TYUGU E.C., 1989, *Programowanie z bazą wiedzy*, WNT.
WÓJCIK M., 1991, *Zasada rezolucji. Metoda automatycznego wnioskowania*, PWN.

Indeks

Podręcznik jest przeznaczony dla studen[tów] cych informatykę na uczelniach technicznych.

Wprowadzenie wyjaśniające czym jest logika zawarto w pierwszym rozdziale, natomiast pozostały materiał podzielono na cztery części.

Część pierwsza, obejmująca rozdziały od 2. do 6., jest prezentacją elementów teorii mnogości, algebr abstrakcyjnych i języków formalnych.

W części drugiej, obejmującej rozdziały od 7. do 10., omówiono rachunek zdań i kwantyfikatorów – ich składnię, semantykę oraz związane z nimi systemy dowodzenia oparte na sekwentach Gentzena i regule rezolucji.

Część trzecia ma charakter informacyjny. W rozdziałach 11. i 12. omówiono krótko inne systemy dowodzenia oraz dokonano przeglądu innych, nieklasycznych logik.

W części czwartej, obejmującej rozdziały 13. i 14., przedstawiono zastosowanie metod logiki do definiowania składni i semantyki języków programowania oraz klasyczną logikę programów Hoare'a służącą dowodzeniu poprawności programów.

Wydawnictwa Politechniki Wrocławskiej
są do nabycia w księgarni
„Tech"
plac Grunwaldzki 13, 50-377 Wrocław
budynek D-1 PWr., tel. 071 320 29 35
Prowadzimy sprzedaż wysyłkową

ISBN 978-83-7493-349-0