Biblioteka
Inżynierii Oprogramowania

Redaktor serii BARBARA OSUCHOWSKA

Komitet Redakcyjny

ANDRZEJ J. BLIKLE
STANISŁAW GANCARCZYK
LEON ŁUKASZEWICZ
JAN MADEY
ANTONI MAZURKIEWICZ
ANDRZEJ SALWICKI
*przewodniczący* WŁADYSŁAW M. TURSKI
STANISŁAW WALIGÓRSKI
JAN WĘGLARZ
JAN ZABRODZKI

Wykaz dotychczas wydawanych tomów zamieszczono na końcu książki

Grażyna Mirkowska Andrzej Salwicki

# Logika algorytmiczna dla programistów

Ludwikow
z przyjaźnią
G.A. & A.S.

Wydawnictwa Naukowo-Techniczne
Warszawa

Grażyna Mirkowska
Instytut Informatyki UW

Andrzej Salwicki
Instytut Informatyki UW

© Copyright by Wydawnictwa
Naukowo-Techniczne
Warszawa 1992

All rights reserved
*Printed in Poland*

ISBN 83-204-1296-X

# Logika algorytmiczna dla programistów

# Algorithmic logic for programmers

English summary, see p. 295

# Логика для программистов

Русское резюме, см. смр. 296

Tematem tej książki jest logika algorytmiczna i jej zastosowania w analizie semantycznych własności programów, specyfikowaniu modułów programów oraz aksjomatycznym definiowaniu semantyki języków programowania.
Zastosowania logiki algorytmicznej zilustrowano licznymi przykładami. Omówiono też oryginalne matematyczne modele obliczeń współbieżnych.
Wykazano, że klasyczna logika pierwszego rzędu nie może być adekwatnym narzędziem badania własności programów. Przedstawiona w tej książce logika algorytmiczna zawiera w sobie klasyczne rachunki logiczne i stanowi ich istotne rozszerzenie. Dokonano porównania z innymi, znanymi logikami programów.
Książka jest przeznaczona dla programistów, projektantów systemów przetwarzania informacji, pracowników nauki zajmujących się informatyką oraz dla studentów kierunków informatycznych i matematycznych.

Redaktor
Henryka Walas

Okładkę i układ typograficzny serii projektował
Tadeusz Pietrzyk

Redaktor techniczny
Maria Żurawska

Tytuł dotowany przez Ministra
Edukacji Narodowej

# Spis treści

Przedmowa

1 Wstęp

1.1 Dlaczego logika? . . . . . . . . . . . . . . . . . . . . . . 13
1.2 Jaka logika? . . . . . . . . . . . . . . . . . . . . . . 16

2 Struktury danych

2.1 Wprowadzenie . . . . . . . . . . . . . . . . . . . . . . 22
2.2 O systemach relacyjnych . . . . . . . . . . . . . . . . . . . . . . 22
2.3 Systemy relacyjne do reprezentacji zbiorów i ciągów skończonych . . . . . . . . . . . . . . . . . . . . . . 34
2.4 O języku formalnym . . . . . . . . . . . . . . . . . . . . . . 38
2.5 Problemy wyrażalności w języku pierwszego rzędu . . . . . 47

3 Deterministyczne programy iteracyjne

3.1 Wprowadzenie . . . . . . . . . . . . . . . . . . . . . . 53
3.2 Język programów . . . . . . . . . . . . . . . . . . . . . . 53
3.3 Semantyka . . . . . . . . . . . . . . . . . . . . . . 56
3.4 Semantyczne własności programów . . . . . . . . . . . . . 65
3.5 Język algorytmiczny . . . . . . . . . . . . . . . . . . . . . . 74
3.6 Wyrażalność w języku algorytmicznym . . . . . . . . . . . . 82

4 Logika algorytmiczna

4.1 Wprowadzenie . . . . . . . . . . . . . . . . . . . . . . 92
4.2 Aksjomatyzacja . . . . . . . . . . . . . . . . . . . . . . 92
4.3 Twierdzenie o pełności logiki algorytmicznej . . . . . . . . 103
4.4 Teorie algorytmiczne . . . . . . . . . . . . . . . . . . . . . . 114

Przykład dowodu . . . . . . . . . . . . . . . . . . . 122
Aksjomatyczna definicja semantyki . . . . . . . . . . . . . . 125
Czy można zautomatyzować dowodzenie twierdzeń? . . . . 132

Algorytmiczne teorie struktur danych 142

1 Od specyfikacji do implementacji . . . . . . . . . . . . . . . . 142
2 Specyfikacja struktur. Typy pierwotne . . . . . . . . . . . . 146
3 Kolejki . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 152
4 Kolejki priorytetowe . . . . . . . . . . . . . . . . . . . . . . 159
5 Drzewa binarne . . . . . . . . . . . . . . . . . . . . . . . . . 168
6 Drzewa binarnych poszukiwań . . . . . . . . . . . . . . . . . 171
7 Interpretacja . . . . . . . . . . . . . . . . . . . . . . . . . . . 178
8 Implementacja . . . . . . . . . . . . . . . . . . . . . . . . . . 183
9 Moduł symulacyjny . . . . . . . . . . . . . . . . . . . . . . . 187
10 Synteza . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 189

O procedurach 193

1 Wprowadzenie . . . . . . . . . . . . . . . . . . . . . . . . . . 193
2 Bloki . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 193
3 Aksjomat bloku . . . . . . . . . . . . . . . . . . . . . . . . . . 199
4 Podprogramy i procedury . . . . . . . . . . . . . . . . . . . . 203
5 Aksjomaty procedur . . . . . . . . . . . . . . . . . . . . . . . 209
6 Procedury funkcyjne. Obliczenia formalne . . . . . . . . . . . 211

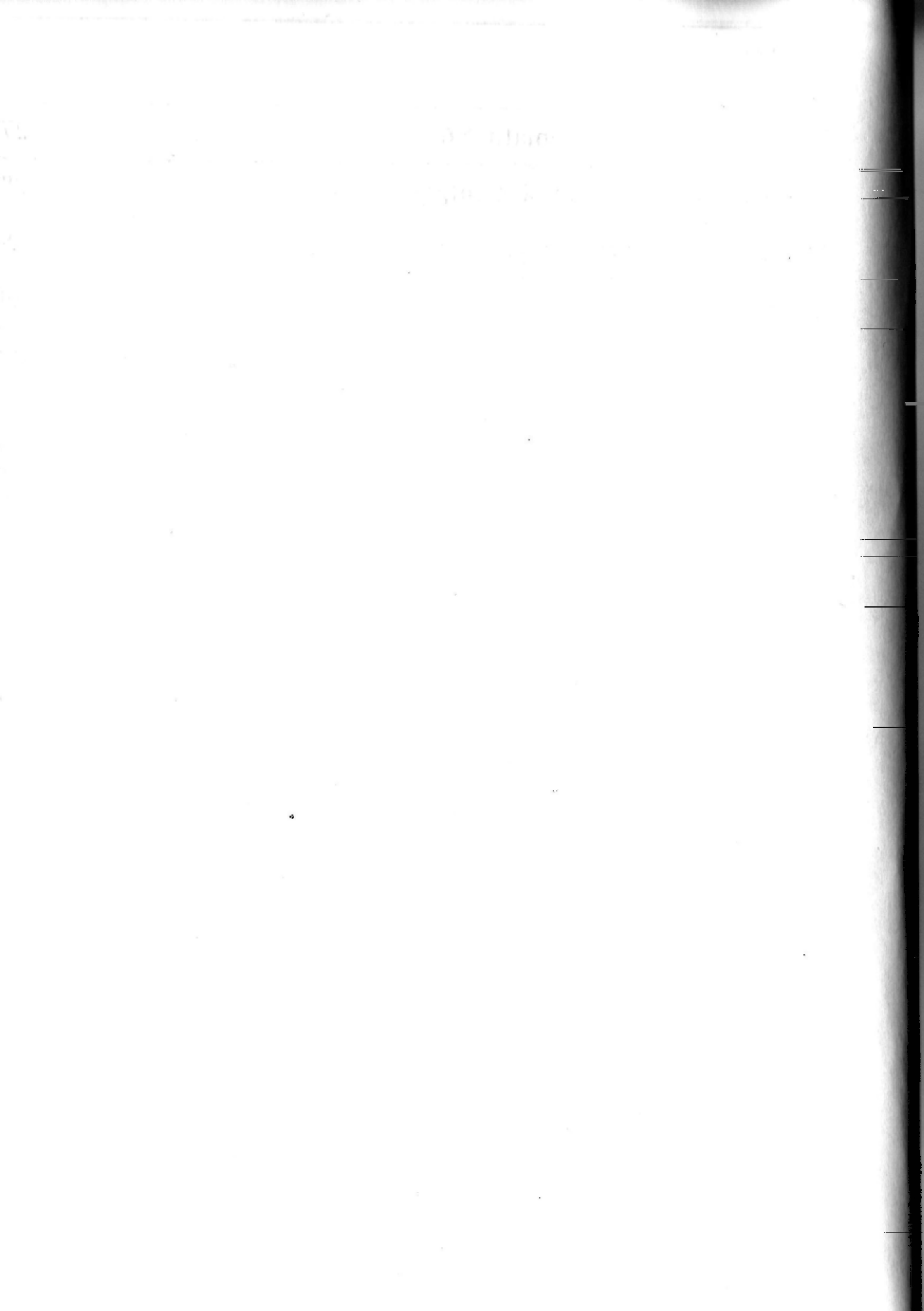
Współbieżność 217

1 Programy współbieżne . . . . . . . . . . . . . . . . . . . . . 217
2 Semantyka MAX . . . . . . . . . . . . . . . . . . . . . . . . . 219
3 Inne koncepcje semantyki . . . . . . . . . . . . . . . . . . . 223
4 Semantyki MAX i ARB w sieciach Petriego . . . . . . . . . 231
5 Logika modalna jako narzędzie analizy programów współbieżnych . . . . . . . . . . . . . . . . . . . . . . . . . . . . 235
6 Aksjomaty modalne obliczeń współbieżnych . . . . . . . . 237
7 Aksjomaty modalne definiują semantykę programu współbieżnego . . . . . . . . . . . . . . . . . . . . . . . . . . . . 243

8 O innych logikach programów 250

.1 Floyda opisy programów . . . . . . . . . . . . . . . . . . . . 250
.2 Logika Hoare'a . . . . . . . . . . . . . . . . . . . . . . . . . . 255
.3 O rachunku Dijkstry . . . . . . . . . . . . . . . . . . . . . . . 261
.4 Logika dynamiczna . . . . . . . . . . . . . . . . . . . . . . . . 266
.5 Logika temporalna . . . . . . . . . . . . . . . . . . . . . . . . 274

Dodatek A. Dowód lematu 5.6 . . . . . . . . . . . . . . . . . . . . 27
Dodatek B. Dziwna implementacja kolejek . . . . . . . . . . 28
Dodatek C. O współprogramach . . . . . . . . . . . . . . . . . . 28
Literatura 29
Skorowidz 29



# Przedmowa

Przedstawiamy czytelnikowi książkę o logice w programowaniu.* Logika jest dziś narzędziem używanym w wielu dziedzinach informatyki: w inżynierii oprogramowania, bazach danych, systemach doradczych (ang. expert systems), robotyce, teorii systemów współbieżnych i in. Czy istnieje jedna logika umożliwiająca formułowanie i rozwiązywanie kwestii z tak różnorodnych dziedzin? Oczywiście nie. Stosuje się m.in. logikę klasyczną, intuicjonistyczną, logiki modalne, logiki oparte na metodach rezolucji, logikę temporalną, różne logiki programów (m.in. logikę algorytmiczną, dynamiczną, Floyda-Hoare'a), logiki wielowartościowe, infinitystyczne, wyższych rzędów. A należy przewidywać powstanie i rozwój nowych rachunków logicznych. Do opisu zjawisk zachodzących w układach scalonych, np. VLSI, jest potrzebny zupełnie nowy formalizm logiczny, dwuwymiarowy, w odróżnieniu od obecnych, liniowych. Klasyczna logika predykatów z metodą rezolucji znajdują się u podstaw języka programowania Prolog i tzw. programowania w logice. Różne logiki niestandardowe, np. logika przekonań i domniemań, mają w przyszłości znaleźć zastosowanie w tzw. sztucznej inteligencji. Rozmaite warianty logiki temporalnej są, według rozpowszechnionej opinii, odpowiednim narzędziem do analizowania programów współbieżnych. Liczba prac poświęconych logikom w informatyce jest tak duża, że są organizowane oddzielne międzynarodowe konferencje poświęcone tylko temu tematowi. Dziedzina ta jest tak obszerna, że musieliśmy dokonać wyboru spośród wielu tematów.

W naszym przekonaniu wytwarzanie oprogramowania wymaga stosowania aparatury logicznej właściwej procesom specyfikacji, analizy i implementacji systemów programistycznych. Okazuje się, że klasyczna logika nie wystarcza do opisu zjawisk, z jakimi mamy do czynienia w programowaniu. Jest ona w swej istocie statyczna i nie oddaje dynamiki procesów algorytmicznych. Zdecydowaliśmy się więc na przedstawienie rachunku logicznego, jaki, naszym zdaniem, może przydać się programistom

---

* Książka ta była wspierana przez program badawczy RP.1.09 Ministerstwa Edukacji Narodowej.

w ich pracy. Prezentujemy logikę algorytmiczną najprostszych programów i jej zastosowania w specyfikowaniu struktur i algorytmów, w analizie semantycznych własności programów i w definiowaniu semantyki języka programowania.

Nie oczekuje się od czytelnika żadnej specjalnej wiedzy o tematyce poruszanej w tej książce. Zakładamy, że są mu znane podstawowe prawa logiki w zakresie odpowiadającym programowi szkoły średniej. Jest pożądana minimalna choćby znajomość algorytmów i programowania. Wierzymy, że książka, którą oddajemy, może wprowadzić czytelnika w problemy programowania, ale zdobycie osobistego doświadczenia w układaniu i uruhamianiu programów jest niezbędne.

Nie rozwijamy z osobna klasycznej logiki zdań ani klasycznego rachunku predykatów. Oba te systemy są częścią prezentowanej logiki. Ponadto, literatura na ten temat jest obszerna i w każdej bibliotece można znaleźć wiele cennych pozycji. Zaliczamy do nich zwięzłą monografię Lyndona [33], książki Grzegorczyka [21] i Rasiowej [44] (polecamy je szczególnie tym, którzy do informatyki przyszli bez matematycznego przygotowania).

Naszą książkę adresujemy do słuchaczy studiów informatycznych na uniwersytetach i politechnikach, do programistów mających ambicję ulepszenia swego warsztatu pracy, do projektantów dużych systemów informatycznych i twórców nowych języków programowania. Nie obiecujemy, że po przeczytaniu tej książki czytelnik zacznie pisać lepsze programy (bardziej efektywne, bardziej pomysłowe itp.). Z pewnością jednak będzie lepiej sobie zdawał sprawę z tego, jak przebiega jego praca i jakie narzędzia mogą być w razie potrzeby użyte. Mamy nadzieję, że znajomość problemów poruszanych w tej książce pozwoli czytelnikowi lepiej zrozumieć rolę specyfikacji i weryfikacji w dokumentowaniu prac programistycznych.

Oprócz logiki algorytmicznej prezentujemy pewną logikę modalną i jej semantykę opracowaną na wzór Kripkego. Logika modalna znalazła wiele różnych zastosowań. W tej książce prezentujemy jedno z nich: zastosowanie do budowy matematycznego modelu obliczeń współbieżnych. Sama logika algorytmiczna może być zresztą uważana za wariant logiki wielomodalnej.

Książka ta może być przydatna podczas wielu różnych zajęć na kierunku informatycznym. Może być uzupełnieniem wykładów: wstępu do informatyki, logiki z elementami teorii mnogości, metod programowania, algorytmów i struktur danych, semantyki języków programowania. Książka była przez nas używana jako podręcznik do zajęć z teorii programów. Podczas jednosemestralnego wykładu można przedstawić niemal cały materiał tu zawarty. Sugerowalibyśmy przerobienie większej liczby dowodów podczas ćwiczeń. Zauważyliśmy, że studenci znajdują sporo satysfakcji z udanych prób aksjomatyzacji struktur danych. Tym, których zainteresują

poruszone tutaj zagadnienia, zalecalibyśmy sięgnięcie po pełniejsze monografie dotyczące logiki i logik programów [3, 21, 23, 25, 29, 33, 40, 45].

Jesteśmy bardzo wdzięczni Recenzentom za krytyczne uwagi, które pomogły nam ulepszyć pierwotny maszynopis książki. Wiele cennych uwag przekazali nam także studenci. Pragniemy podziękować Wydawnictwu za cierpliwość i wyrozumiałość, z jaką przyjmowało nasze opóźnienia. Jest oczywiste jednak, że wszystkie usterki książki należy przypisać jej autorom. Będziemy bardzo zobowiązani za przesłanie nam wszelkich uwag.

G. MIRKOWSKA
A. SALWICKI

Warszawa, w sierpniu 1992

# Wstęp 1

## Dlaczego logika? 1.1

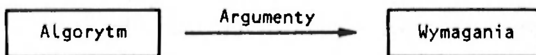
*Projektowanie i analiza systemów programistycznych muszą opierać się na obiektywnych metodach wnioskowania niezależnych od doświadczenia oraz osoby tworzącej algorytm. Metody te powinny w niezawodny sposób umożliwić wyprowadzenie prawdziwych własności algorytmów z prawdziwych przesłanek.*

Na pewno wielu programistów czuje potrzebę oparcia swej pracy na solidnych teoretycznych podstawach. Zanim przejdziemy do próby uzasadnienia naszego przekonania, że metody logiczne są niezbędne dla większości twórców oprogramowania, chcemy jednak zauważyć, że sami znamy kilku bardzo dobrych, wręcz wybitnych programistów i wiemy, że nie posługują się oni żadnymi metodami formalnymi. Sprzeczność, powiecie Państwo — jeśli tak, to autorzy zachęcają nas do czytania rzeczy zbędnej i niepotrzebnej. Nie jest to jednak tak proste, jak by się zdawało. Ci dobrzy fachowcy w dziedzinie programowania intuicyjnie stosują formalne prawa rządzące zachowaniem się programów, ich obliczeniami. Można w tym momencie dokonać porównania z matematyką. Wielu wybitnych matematyków prowadzi badania i uzyskuje fascynujące wyniki, chociaż nie studiowali logiki matematycznej i nie znają subtelnych wyników rozważań metamatematycznych. A jednak każdy dobry matematyk posługuje się logicznymi metodami wnioskowania, są one nieodłącznym atrybutem pracy matematyka. Zwróćmy uwagę na fakt, że matematyka rozwija się od tysiącleci, i na to także, że badania matematyczne przyczyniły się w znacznym stopniu do rozwoju nauki o wnioskowaniu, którą jest logika. Informatyce ten czas nie był dany. Powstała około czterdzieści lat temu i rozwija się gwałtownie, a co ważniejsze, jeszcze szybciej rozwijają się jej zastosowania. Język informatyki jest odmienny od języka matematyki. Ta sytuacja stwarza zupełnie inne potrzeby. Wielu badaczy o poważnym dorobku w dziedzinie oprogramowania (by wymienić tu tylko Hoare'a i Dijkstrę) dostrzegło konieczność stworzenia logicznych narzędzi właściwych informatyce.

Zauważmy, że ilość i znaczenie oprogramowania stale rośnie. Na światowym rynku informatycznym obroty oprogramowaniem są znacznie większe niż obroty sprzętem. Można wiele zyskać szybko produkując programy dobrej jakości. Z drugiej strony cena oprogramowania jest wysoka. Jest to spowodowane rozlicznymi czynnikami. Najkrócej mówiąc, oprogramowania (ang. software) — w odróżnieniu od sprzętu (ang. hardware) — nie wytwarza się w fabrykach.

Wiele osób sądzi, że w przyszłości oprogramowanie będzie się produkować metodami przemysłowymi. Uważa się, że podstawą takich metod będzie odpowiednia teoria. U podstaw techniki znajdują się różne teorie fizyczne i matematyczne. Dla rozwinięcia przemysłowej technologii produkcji oprogramowania jest niezbędny rozwój teorii programów. Cena, jaką płacimy za stosowanie programów z błędami, jest nie mniejsza niż np. koszt budowy mostu, który się zawali, ponieważ został zbudowany niezgodnie z prawami mechaniki. Na zwiększenie tej ceny w przypadku oprogramowania wpływają różnorodność i masowość zastosowań techniki obliczeniowej w dzisiejszym świecie, a także pośpiech, z jakim wprowadzamy nowinki techniczne do produkcji. Pragnienie oparcia produkcji programów na sformalizowanym rachunku matematycznym, w celu wyeliminowania groźnych i kosztownych błędów, jest celem minimum. Wiele się pisze i mówi o konieczności przyspieszenia produkcji oprogramowania, o jej zautomatyzowaniu. Nie przesądzając w tym miejscu, czy jest możliwa mechanizacja pracy programistów, zwróćmy uwagę na fakt, że niewiele mamy narzędzi usprawniających pracę ludzi tworzących oprogramowanie. Jest swego rodzaju paradoksem, że powstało dużo programów ułatwiających pracę projektantów maszyn i układów elektronicznych oraz architektów, lekarzy, kompozytorów i twórców niemal wszystkich profesji, a praca twórców oprogramowania nie doczekała się odpowiednich dla tego zawodu pomocy programistycznych.

Szybkie układanie i uruchamianie efektywnych algorytmów nie wyczerpuje całości problemów jakie napotykamy. Nie w samodzielnej pracy najlepszego nawet fachowca zasadza się problem tworzenia oprogramowania. Praca nad oprogramowaniem jest procesem społecznym. Oprócz autora programu (a często program jest tworzony przez spory zespół ludzki) i oczywiście komputera, w procesie tym biorą udział: użytkownik programu, pracownik pielęgnujący program i in. Proces ten wymaga swoistego języka do komunikacji pomiędzy jego uczestnikami. Język taki nie jest ani językiem programowania, ani językiem etnicznym jakiejś grupy ludzkiej. Tworzenie algorytmów prowadzi do powstawania pewnych tekstów. Mają one spełniać pewne semantyczne wymagania. Mamy więc zadanie, w którym występują co najmniej trzy elementy (rys. 1.1).




Rys. 1.1

Argumenty stanowią o tym, czy programista przekona użytkownika-zleceniodawcę, że jego algorytm spełnia postawione wymagania. Język, w jakim odbywa się proces wytwarzania oprogramowania, musi stwarzać możność formułowania wymagań stawianych programowi, zapisania programu i sformułowania argumentów — wyników analizy.

W związku z tymi rozważaniami pozostają nasze następujące poglądy:

Przekazanie algorytmu bądź systemu programistycznego do eksploatacji (lub sprzedaży) bez upewnienia się o jakości produktu jest wielce ryzykowne, grozi bowiem co najmniej stratami finansowymi, a może nawet spowodować zagrożenie życia ludzkiego.

Jakie jest wyjście? Przekonać nabywcę, że nasz produkt wykonano zgodnie ze stanem nauki i z zasadami sztuki. (Inżynier nie może ponieść odpowiedzialności za zawalenie się mostu, jeżeli zaprojektował go zgodnie z aktualnym stanem wiedzy; lekarz nie odpowiada za nieudaną operację, jeśli dokonano jej zgodnie ze stanem wiedzy medycznej itp.). Jest to oczywiście program minimum — właściwie to, co chcemy tu zaproponować, stwarza możliwość głębszej analizy oprogramowania (por. p. 5.1–5.8).

(1) Algorytm jest tylko i wyłącznie napisem, o jego znaczeniu decyduje interpretacja znaków w nim występujących (por. przykład 3.5).

(2) Interpretację tę można dokładnie scharakteryzować przyjmując pewne założenia (aksjomaty, por. rozdz. 4).

(3) Analizę programu można przeprowadzić na gruncie formalnym, dowodząc, że własności semantyczne programu wynikają z aksjomatów, no i oczywiście z tekstu programu (por. rozdz. 4).

Jakiego rodzaju pytania pojawiają się w trakcie prac nad oprogramowaniem? Sądzimy, że napotykane problemy można zaliczyć do jednej z trzech grup pytań:

(1) Czy zadanie jest rozwiązywalne za pomocą jakiegokolwiek programu?

(2) Czy program, który skonstruowaliśmy (bądź otrzymaliśmy od kogoś innego), jest poprawnym rozwiązaniem postawionego zadania?

(3) Czy dany program jest najlepszym rozwiązaniem zadania?

Wielokrotnie wskazywano na to, że odpowiedź na pytania (2) i (3) może być uzyskana jedynie w drodze dowodu. Dowodu również wymaga negatywna odpowiedź na pytanie (1). W przypadku pytania (1) odpowiednia konstrukcja — odpowiedni eksperyment — ma bardzo wielkie znaczenie. Chcemy zwrócić uwagę na to, że eksperyment powinien wesprzeć nasze teoretyczne spekulacje również w przypadku analizy pytań z grup (2) i (3). Ani teoretyczne rozważania, ani eksperyment nie mogą sobie rościć praw do bycia jedyną metodą pracy nad oprogramowaniem. Znaczenie eksperymentu wynika choćby stąd, że opracowane przez nas programy mają działać,

znaleźć zastosowanie w czyjejś pracy. Znaczenie teorii wynika z masowości zastosowań opracowanego przez nas oprogramowania, z tego, że żadnym eksperymentem nie jesteśmy w stanie stwierdzić: „ten oto program jest wolny od błędów i będzie działać w sposób pewny i niezawodny we wszelkich sytuacjach, w jakich może się znaleźć w przyszłych jego zastosowaniach". Eksperyment może pomóc w wykryciu błędu, ale nigdy nie możemy użyć go do argumentowania, że program jest wolny od błędu. Wracając do przykładu z budową mostów uważamy, że w analogiczny sposób trzeba postępować z programami, tzn. w trakcie ich projektowania i budowy należy, posługując się dostępną wiedzą teoretyczną, dowodzić, że mają określone własności, a potem należy dokonywać z nimi eksperymentów. Idzie tu nie o to, by konstruktor mostu stał pod nim podczas przejazdu pierwszych pojazdów, ale o to, by uzyskać wiedzę o parametrach produktu programistycznego. Na przykład powinniśmy dokładnie ocenić koszt działania programu jako funkcję rozmiaru danych, powinniśmy poznać stopień wielomianu i jego współczynniki (przyjmując, że funkcja kosztu okazała się wielomianem).

## Jaka logika? 1.2

*Wyrażenia, które występują podczas analizy programów, nie mogą się ograniczać tylko do formuł logiki pierwszego rzędu. Muszą zawierać programy, bo o własnościach programów mają mówić. Z drugiej strony wyrażenia niealgorytmiczne, nie zawierające programów, okazują się niewystarczające do zdefiniowania własności programów.*

*Istnieją systemy dedukcyjne odpowiednie do przeprowadzania w nich rozumowań o semantycznych własnościach programów. Są to logiki programów. Tu będziemy lansować logikę algorytmiczną, stworzoną i rozwijaną w Polsce od roku 1968. Nie dlatego, że jest starsza niż inne, lecz dlatego, że jest lepiej poznana i ma więcej udokumentowanych zastosowań.*

*Aparat pojęciowy i dedukcyjny logiki algorytmicznej nadaje się do specyfikowania struktur danych i odpowiadających im modułów programów. To zastosowanie logiki wydaje się nam ważniejsze nawet niż dowodzenie poprawności algorytmów. Logika algorytmiczna, jej aksjomaty i reguły wnioskowania pozwalają zdefiniować semantykę języków programowania.*

Stwierdzamy, że do pracy z programami nie wystarczają narzędzia oferowane przez logikę matematyczną pierwszego rzędu. Nie jest ona w stanie wyrazić własności programów. Okazuje się, że pewne własności algorytmów są równoważne własnościom matematycznym, o których od dawna wiemy, że nie można znaleźć dla nich takich formuł z języka pierwszego rzędu, które by wyrażały te własności. Ich lista jest długa

i znajduje się na niej wiele pojęć podstawowych dla matematyki, np. „być liczbą naturalną”, „być pierścieniem archimedesowskim”. W dalszym ciągu przedstawiamy te własności i pokazujemy, że są to własności algorytmiczne.

W trakcie rozwoju logiki matematycznej powstały różne jej warianty, m.in. logika formuł z nieskończonymi alternatywami i koniunkcjami, logiki wyższych rzędów umożliwiające wyrażenie tych własności, które nie dają się zdefiniować w języku logiki pierwszego rzędu. Większość znanych nam własności programów daje się wyrazić w językach tych logik. Na przykład własność „program **while** γ **do** K **od** nie ma obliczeń nieskończonych” daje się wyrazić jako nieskończona alternatywa formuł (coraz dłuższych) mówiących, że obliczenie programu nie będzie miało żadnego powtórzenia instrukcji K lub będzie miało jedno powtórzenie, lub dwa, lub ... Podobnie, można wyrazić tę własność programu operując kwantyfikatorami dotyczącymi skończonych zbiorów (tzw. słaba logika drugiego rzędu). Wydaje się nam jednak, że stosowanie takich technik nie jest specjalnie atrakcyjne dla programistów. Należałoby przecież problemy dotyczące konkretnych programów i warunków, jakie mają one spełniać, przetłumaczyć na np. język logiki drugiego rzędu, zanalizować problem w logice drugiego rzędu i potem przetłumaczyć wynik tej analizy na język oryginalnego programu, od którego to wszystko się zaczęło. Nawet jeśliby udało się nam zautomatyzować wszystkie kroki tej procedury, to byłaby ona dla programisty czymś nienaturalnym, sztucznym. Dlatego też proponujemy logikę algorytmiczną.

Logika algorytmiczna jest dla informatyki tym, czym logika matematyczna jest dla matematyki. Wyrażenia tej logiki są budowane z programów i formuł logicznych. Nie ma potrzeby stosowania nieskończonych alternatyw ani kwantyfikatorów drugiego rzędu. Formuły logiki algorytmicznej mają moc wyrażania wszystkich semantycznych własności programów, które są ważne w praktyce programistycznej. Logika algorytmiczna dostarcza narzędzi do rozumowań dedukcyjnych. W ten sposób umożliwiono badania programu a priori, przed wykonaniem eksperymentu obliczeniowego.

Autorom tego opracowania wydaje się, że znaczenie zaproponowanych przez nich narzędzi logicznych leży nie tyle w sferze dowodzenia własności programów, ile w obszarze dokumentowania (specyfikacji) dużych systemów programistycznych. W rozdziale 5 dajemy rozbudowany przykład takiego zastosowania logiki algorytmicznej. Wykazujemy, że w fazie projektowania systemu można wyodrębnić moduły, że zadania tych modułów mogą być opisane aksjomatycznie i że analiza takich aksjomatycznych teorii ma, dla wielu konkretnych typów danych, pozytywne znaczenie dla procesu programistycznego. Wskazujemy też na związek pojęcia interpretacji jednej teorii algorytmicznej w innej, z pojęciem implementacji struktur danych.

Przedstawiona w tej książce logika algorytmiczna zawiera w sobie klasyczną logikę pierwszego rzędu i klasyczny rachunek zdań. Jeżeli ograni-

czyć się do wyrażeń nie zawierających programów, to reguły wnioskowania R2-R5 wymienione w rozdz. 4 stają się zbędne. Można w tej sytuacji ograniczyć się jedynie do klasycznych reguł wnioskowania: modus ponens (m.p.) i praw wprowadzania klasycznych kwantyfikatorów w poprzedniku i następniku implikacji.

Spotykamy się z zarzutami, że logika algorytmiczna nie jest dla ludzi, bo zawiera reguły wnioskowania o nieskończonej liczbie przesłanek. Cóż odpowiedzieć? Po pierwsze, zapytajmy czy można skonstruować jakikolwiek system dedukcyjny opisujący zachowanie się obliczeń programów w sposób pełny i tak, by system ten nie zawierał infinitarnych reguł wnioskowania. Odpowiedź brzmi: nie. Po drugie, zauważmy, że proponowane finitarne systemy wnioskowania po bliższym przyjrzeniu okazują się systemami właśnie infinitarnymi (por. p. 8.3). Po trzecie, niektórzy badacze powiadają: unikniemy reguł o nieskończonej liczbie przesłanek przyjmując jako aksjomaty formuły pierwszego rzędu prawdziwe w badanej strukturze danych (lub w klasie struktur danych). No cóż, formalnie wszystko jest w porządku. Z definicjami można igrać do woli i bez widocznych ograniczeń. Ciekawe jednak czy zdajemy sobie sprawę co na co zamieniamy? Przecież, na ogół nie ma żadnej metody rozpoznawania czy formuła pierwszego rzędu jest prawdziwa czy nie w ustalonej klasie struktur danych. Zbiór formuł arytmetyki pierwszego rzędu prawdziwych w standardowym modelu arytmetyki jest zbiorem hiperarytmetycznym [31]. Naszym zdaniem mamy tu do czynienia z wykrętem. Idzie o to, by znaleźć argumenty dowodzące prawdziwości formuł, a nie o to, by przyjmować te formuły za aksjomaty. Warto zresztą zauważyć, że zbiór prawdziwych formuł algorytmicznych stwierdzających poprawność programu względem formuł wolnych od kwantyfikatorów jest znacznie niżej w hierarchii Kleene-Mostowskiego niż zbiór wszystkich formuł pierwszego rzędu prawdziwych w standardowym modelu arytmetyki.

Spróbujmy jeszcze inaczej wyrazić naszą wątpliwość. Są rozwijane tzw. abstrakcyjne typy danych, tzn. algebraiczne (najczęściej równościowe) teorie struktur danych. Przyjmuje się pewne równości lub (rzadziej) formuły pierwszego rzędu jako aksjomaty tych struktur. Wtedy okazuje się, że aksjomatów tych jest za mało, by określić dokładnie tę klasę, o którą nam chodziło. Tak jest dla prawie wszystkich struktur danych interesujących informatyków. (Fakt ten jest znany z badań nad klasycznym rachunkiem predykatów). Żaden skończony zbiór formuł pierwszego rzędu nie stanowi dostatecznie precyzyjnej aksjomatyzacji struktury liczb naturalnych. Podobnie jest dla innych struktur znaczących dla informatyki. W tej właśnie sytuacji powinniśmy poszukiwać takich narzędzi, by osiągnąć oba cele: możliwość opisu (aksjomatyzację) struktury danych i możliwość analizowania obliczeń programów w tej strukturze. Propozycja logiki algorytmicznej harmonijnie kojarzy pracę w obu tych kierunkach.

Niektórzy czytelnicy oczekiwali być może książki o logice matematycz-

nej w jej klasycznym już dziś kształcie. Zawiedzionych odsyłamy do obszernej literatury [21, 33, 41, 44]. Pragniemy jednak podkreślić, że

(1) dowody w logice klasycznej są dowodami w logice algorytmicznej, ponieważ logika algorytmiczna jest rozszerzeniem logiki klasycznej;

(2) do pracy z programami, z ich semantycznymi własnościami język logiki pierwszego rzędu ma zbyt małą moc wyrażania, np. własności „stopu" nie da się wyrazić w języku pierwszego rzędu.

Ponadto należy wspomnieć o tym, że nie da się zastosować głębszych wyników logiki matematycznej w teorii bądź praktyce programowania, ponieważ

(3) Metateoria logiki algorytmicznej różni się w istotny sposób od metateorii logiki klasycznej. Zdanie to jest prawdziwe zarówno w odniesieniu do teorii modeli (logikom programów nie przysługują własności tak popularne jak „górne", twierdzenie Skolema-Loevenheima, twierdzenie Łosia o ultraprodukcie i in., por. [44]), jak i teorii dowodu (logiki programów nie mają własności interpolacji Craiga, nie zachodzi w nich twierdzenie o eliminowaniu definicji uwikłanych Betha ani twierdzenie Robinsona, por. [33, 21]).

Chcemy zwrócić uwagę na podobieństwa łączące różne logiki programów. To co łączy wszystkie logiki programów to mniej lub bardziej uniwersalna zdolność wyrażania semantycznych własności programów formułami odpowiedniego języka i narzędzia dedukcyjne pozwalające zamienić proces semantycznej weryfikacji własności programu procesem dowodzenia formuły wyrażającej tę własność.

W przedstawianym czytelnikowi tomie zajmujemy się paroma z wielu przedstawionych wyżej problemów i nasuwających się pytań. Zaczniemy od omówienia struktury pewnego prostego języka programowania, a dokładniej pewnej klasy języków. Każdy język będziemy traktować jako parę złożoną z alfabetu języka i zbioru wyrażeń poprawnie zbudowanych. Alfabet zawiera: zmienne, funktory, predykaty, znaki operacji logicznych, znaki operacji programotwórczych, kwantyfikatory i znaki pomocnicze. W zbiorze wyrażeń poprawnie zbudowanych można wyróżnić trzy podzbiory: termów, formuł i programów. Analizując przykłady dostrzeżemy, iż program jest wyrażeniem, którego znaczenie nie jest zdeterminowane przez sam tylko tekst programu. Jeden i ten sam program może mieć wiele znaczeń w zależności od interpretacji, jaką nadajemy znakom występującym w programie. Przy ustalonej interpretacji znaków mówimy o strukturze danych, w jakiej dokonuje się obliczeń programu. Obserwacja, że program może być wykonywany w różnych strukturach danych, jest ważna nie tylko do zrozumienia jak program działa, ale i dlatego, że ma praktyczne znaczenie. Umożliwia ona mianowicie wielokrotne wykorzystanie jednego algorytmu w różnych kontekstach dla osiągnięcia całkowicie różnych celów. Ten sposób myślenia pozwala nawiązać do zaawansowanej metodologii programowania, do abstrakcyjnych typów danych itp.

W jaki sposób opisać znaczenie programu? Jesteśmy przekonani, że większość z nas opiera swoje intuicje związane z programowaniem na pojęciu procesu obliczeniowego. Postaramy się, tytułem przykładu, podać definicję obliczenia wyznaczonego przez program, strukturę danych (tzn. odpowiedni system algebraiczny) i wartościowanie początkowe zmiennych. Wartościowanie zmiennych jest formalnym odpowiednikiem pojęcia stanu pamięci. Struktura danych jest formalnym odpowiednikiem jednostki arytmetyczno-logicznej komputera, w którym wykonujemy obliczenia. Pojęcie struktury jest bardziej uniwersalne i dzięki temu można analizować zachowanie programu również w tzw. abstrakcyjnych strukturach danych. Obliczenie programu może być skończone lub nie, udane lub nieudane; wyniki obliczenia mogą być poprawne albo niepoprawne. Krótko mówiąc, obliczenia mają własności semantyczne i te własności są najbardziej interesujące dla użytkownika programu.

Wspomnieliśmy wyżej, że opieranie analizy programów tylko na wynikach eksperymentów jest nieuzasadnione z metodologicznego punktu widzenia. Eksperymenty, zwane często uruchamianiem, mogą pomóc w wykryciu faktów takich, jak nieograniczona długość obliczeń programu dla pewnych danych, niepoprawność wyników itp. Na drodze eksperymentalnej nie możemy wyprowadzić wniosku, że obliczenia badanego programu będą miały postulowane własności dla wszelkich danych, które mogą być rozważane. Jakie więc należy przyjąć rozwiązanie? Proponujemy, by własności semantyczne programów, a dokładniej — obliczeń programów, były wyrażane przez pewne formuły i by badać te formuły na drodze analitycznej, tzn. dowodzić ich prawdziwości. Będziemy mówić, że własność semantyczna *WS* jest wyrażalna pewną formułą $\alpha$, jeżeli formuła $\alpha$ jest prawdziwa wtedy i tylko wtedy, gdy* zachodzi własność semantyczna *WS*.

W dalszym ciągu skonstruujemy taki język, który zawiera programy i formuły logiczne wyrażające semantyczne własności programów. Kolejnym więc i naturalnym zadaniem stanie się opracowanie systemu dedukcyjnego umożliwiającego dowodzenie prawdziwości formuł, nazywamy je formułami algorytmicznymi. Rachunek logiczny, który skonstruujemy będzie oparty na dwu zbiorach: aksjomatów logicznych i reguł wnioskowania. Zbiory te muszą być tak dobrane, by posługiwanie się nimi było bezpieczne, by nie pozwalały na wyprowadzenie zdań fałszywych. Ponadto, należy tak dobrać aksjomaty i reguły wnioskowania, by każda formuła prawdziwa miała dowód wynikający z przyjętych aksjomatów. W ten sposób zadanie analizy własności semantycznych programów zostaje zastąpione zadaniem badania prawdziwości odpowiednich formuł, a to zadanie z kolei jest zastąpione zadaniem dowodzenia tych formuł.

* Oprócz sformułowań „wtedy i tylko wtedy, gdy” będziemy używać również skrótu wtw.

Skonstruowany rachunek logiczny znajduje różnorodne zastosowania: w analizie własności programów, w specyfikacji zadań programistycznych, w edukacji programistów. Okazało się, że formuły algorytmiczne dobrze opisują warunki wstępne i końcowe programów, mogą więc być użyte do sformułowania zadania przez zlecającego wykonanie pewnego programu. Formuły algorytmiczne mogą być także użyte do specyfikacji systemu programistycznego. Można skonstruować teorie algorytmiczne wielu powszechnie używanych struktur danych. Można także tworzyć systemy oprogramowania razem z ich specyfikacją. Przyjmie ona wtedy postać układów aksjomatów pozalogicznych (specyficznych) opisujących moduły projektowanego systemu. Zadanie konstrukcji systemu może być rozumiane jako zadanie znalezienia modelu dla teorii. Z drugiej strony, aksjomaty takie mogą być użyte jako kryterium poprawności implementacji systemu programistycznego.

Książka, którą przedkładamy, nie porusza wielu tematów. Nasz wybór jest subiektywny. Staraliśmy się wybrać to co, naszym zdaniem, ma znaczenie w praktyce programistycznej. Książkę tę można umownie podzielić na dwie części. W pierwszej znajdują się informacje wstępne (w rozdz. 2) przypominające podstawowe pojęcia algebry uniwersalnej i logiki pierwszego rzędu, prezentacja logiki algorytmicznej (rozdz. 3 i 4), zastosowania tej logiki (rozdz. 4 i 5). Także rozdz. 8 należałoby zaliczyć do tej części.

Umowna część druga stanowi swego rodzaju zaproszenie do wspólnego badania problemów, na które nie znamy dziś odpowiedzi. I tak, nie znamy zbyt wielu zastosowań aksjomatów procedur w dowodach własności programów. Wiemy, że system jest pełny. Wiemy, że może być użyty jako kryterium poprawności implementacji języka programowania. Nie są nam znane, niestety, ciekawsze dowody korzystające z aksjomatu procedury lub reguły pozwalającej wprowadzić instrukcję procedury w poprzedniku implikacji. W rozdziale 8 przedstawiamy oryginalne matematyczne modele obliczeń współbieżnych. Tu sytuacja jest bardziej surowa, nie znamy jeszcze pełnego zestawu aksjomatów charakteryzujących te semantyki.

Poza materiałem tej książki pozostaje niezmiernie ciekawe zadanie aksjomatycznego opisu nowoczesnych konstrukcji programistycznych takich, jak np. klasy, współprogramy, składanie klas (tzw. prefiksowanie), procesy, obsługa wyjątków. Do zadań, które powinny być rozwiązane w niedalekiej przyszłości, należy sprawdzenie czy uda się stworzyć systemy wspomagające proces wytwarzania oprogramowania, wykorzystujące ofertę logiki algorytmicznej.

# Struktury danych

# 2

## Wprowadzenie

## 2.1

Znaczenie programu, jego obliczenia zależą od struktury danych, w której jest on wykonywany. To własności działań i relacji w strukturze danych określają własności programów. Ten sam program realizowany w dwu różnych strukturach danych może się zachowywać w odmienny sposób (np. gdy realizujemy go w 16- i 32-bitowej arytmetyce). Z drugiej strony, dwa różne programy mogą zachowywać się identycznie dzięki własnościom struktury, w której są realizowane obliczenia (np. pewne działanie jest przemienne, a programy różnią się kolejnością argumentów tego działania).

W tej książce przyjmujemy pogląd, że struktury danych należy rozumieć jako systemy relacyjne, kładąc w ten sposób nacisk na algebraiczne własności struktur danych. Dla informatyków termin struktura danych wiąże się najczęściej nie tyle z abstrakcyjnymi operacjami w strukturze, ile z praktycznym sposobem reprezentacji danych w pamięci komputera. W dalszym ciągu wykażemy, że oba te podejścia daje się harmonijnie połączyć.

## O systemach relacyjnych

## 2.2

Niech $X_1, X_2, ..., X_n$ będą niepustymi zbiorami i niech $X_1 \times X_2 ... \times X_n$ oznacza $n$-argumentowy iloczyn kartezjański

$$X_1 \times X_2 \times ... \times X_n = \{(x_1, ..., x_n): x_i \in X_i \quad \text{dla} \quad 1 \leqslant i \leqslant n\}$$

Definicja 2.1

Każdy podzbiór iloczynu kartezjańskiego $X_1 \times X_2 \times ... \times X_n$ nazywamy *relacją n-członową*.
Jeżeli $f$ jest podzbiorem $X_1 \times X_2 \times ... \times X_n \times Y$, gdzie $Y$ jest pewnym niepustym zbiorem oraz dla dowolnych $x_i \in X_i$ $i \leqslant n$, $y, y' \in Y$, z tego, że $(x_1, ..., x_n, y)$, $(x_1, ..., x_n, y')$ są elementami $f$ wynika $y = y'$, to $f$ nazywamy

*n-argumentową funkcją częściową.* Jeżeli $f$ jest funkcją częściową oraz $(x_1, \ldots, x_n, y) \in f$, to piszemy $y = f(x_1, \ldots, x_n)$. Zbiór
$Dom(f) = \{(x_1, \ldots, x_n) \in X_1 \times \ldots \times X_n$: istnieje $y \in Y$, że $(x_1, \ldots, x_n, y) \in f\}$
nazywamy *dziedziną funkcji f.* Zbiór

$$Rg(f) = \{y \in X: \text{istnieje } x \in X_1 \times X_2 \times \ldots \times X_n, \text{ że } f(x) = y\}$$

nazywamy *zbiorem wartości funkcji f.* Jeżeli $Dom(f) \subset A$ i $Rg(f) \subset B$, to powiemy, że $f$ jest funkcją częściową (lub odwzorowaniem) ze zbioru $A$ w zbiór $B$, w skrócie $f: A \rightarrow B$. Jeżeli $Dom(f) = X_1 \times X_2 \times \ldots \times X_n$, to powiemy, że $f$ jest funkcją całkowitą lub po prostu funkcją. Funkcje całkowite i funkcje częściowe będziemy nazywać operacjami.

Oczywiście każda funkcja $n$-argumentowa jest relacją $n+1$-członową. Odwrotnie, każda relacja $n$-członowa $r$ wyznacza jednoznacznie pewną funkcję całkowitą $f$, zwaną funkcją charakterystyczną tej relacji, określoną następująco:

$$f(x_1, \ldots, x_n) = \begin{cases} 1 & \text{gdy} \quad (x_1, \ldots, x_n) \in r \\ 0 & \text{gdy} \quad (x_1, \ldots, x_n) \notin r \end{cases}$$

Jeżeli zbiory $X_1, X_2, \ldots, X_n$ są identyczne i równe $X$ oraz $f \subset X_1 \times \ldots \times X_n$ (tzn. $f \subset X^n$), to powiemy, że $f$ jest $n$-argumentową relacją w $X$. Jeżeli $f$ jest funkcją częściową oraz $f \subset X^{n+1}$, to powiemy, że $f$ jest $n$-argumentową funkcją częściową w $X$. ■

DEFINICJA 2.2

*Systemem relacyjnym* nazywamy dowolny układ postaci

$$\mathbf{A} = \langle A, f_1, \ldots, f_k; r_1, \ldots, r_l \rangle$$

taki, że

(1) $A$ jest niepustym zbiorem,

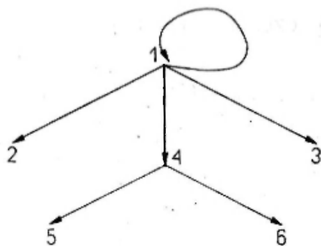
(2) dla dowolnego $1 \leqslant i \leqslant k$, $f_i$ jest $n_i$-argumentową operacją w $A$,

(3) dla dowolnego $1 \leqslant j \leqslant 1$, $r_j$ jest $m_j$-argumentową relacją w $A$.

Zbiór $A$ będziemy nazywać *uniwersum* systemu relacyjnego **A**. Układ liczb naturalnych $\langle n_1, \ldots, n_k; m_1, \ldots, m_l \rangle$ będziemy nazywać *sygnaturą systemu* **A**. Jeżeli w systemie relacyjnym nie występuje żadna relacja, to taki system będziemy nazywać *algebrą.* ■

PRZYKŁAD 2.1

Najprostszym przykładem systemu relacyjnego jest graf. Jest to system złożony ze zbioru i jednej relacji dwuczłonowej, np. $\langle V, E \rangle$. Zbiór $V$ nazywamy zbiorem wierzchołków grafu, a relację $E$ zbiorem jego krawędzi. System taki często przedstawia się w postaci graficznej, elementy zbioru $V$ są



RYS. 2.1

punktami na płaszczyźnie, a pary $(a, b)$ należące do relacji $E$ wektorami o początku w punkcie $a$ i końcu w punkcie $b$. Na rys. 2.1 przedstawiono graf, którego zbiorem wierzchołków jest zbiór $\{1, 2, 3, 4, 5, 6\}$, a zbiorem krawędzi zbiór $\{(1, 2), (1, 3), (1, 4), (1, 1), (4, 5), (4, 6)\}$. □

PRZYKŁAD 2.2

Dwuelementowa algebra Boole'a $\mathbf{B}_0$ jest to system relacyjny postaci

$$\langle B_0, \cup, \cap, -, \mathbf{1}, \mathbf{0} \rangle$$

o sygnaturze $\langle 2, 2, 1, 0, 0 \rangle$ taki, że $B_0 = \{\mathbf{1}, \mathbf{0}\}$ a operacje $\cup$, $\cap$, $-$ są określone dla dowolnych $a, b \in B_0$ następująco:

$$\begin{array}{llllll} a \cup b = \mathbf{1} & \text{wtw} & a = \mathbf{1} & \text{lub} & b = \mathbf{1} \\ a \cap b = \mathbf{1} & \text{wtw} & a = \mathbf{1} & \text{i} & b = \mathbf{1} \\ -a = \mathbf{1} & \text{wtw} & a = \mathbf{0} & & \end{array}$$

Elementy zbioru $B_0$ interpretujemy jako symbole prawdy ($\mathbf{1}$) i fałszu ($\mathbf{0}$). Operacje $\cup$, $\cap$, $-$ nazywamy odpowiednio alternatywą, koniunkcją i negacją. □

PRZYKŁAD 2.3

Niech $s$ oznacza operację następnika w zbiorze liczb naturalnych $N$, 0, stałą (tzn. zeroargumentową operację) w $N$, a $=$ relację równości w zbiorze $N$. Układ $\mathbf{N} = \langle N, s, 0; = \rangle$ jest systemem relacyjnym o sygnaturze $\langle 1, 0; 2 \rangle$. Będziemy go nazywać arytmetyką liczb naturalnych. □

DEFINICJA 2.3

Powiemy, że dwa systemy relacyjne

$$\mathbf{A} = \langle A, f_1, \ldots, f_k; r_1, \ldots, r_l \rangle \quad \text{i}$$
$$\mathbf{B} = \langle B, f'_1, \ldots, f'_s; r'_1, \ldots, r'_t \rangle$$

są podobne wtedy i tylko wtedy, gdy

(1) $k = s$, $l = t$ oraz

(2) $f'_i$ jest $n_i$-argumentową operacją w $B \Leftrightarrow f_i$ jest $n_i$-argumentową operacją w $A$, dla $i \leqslant k$,

(3) $r'_j$ jest $m_j$-członową relacją w $B \Leftrightarrow r_j$ jest $m_j$-członową relacją w $A$, dla $j \leqslant l$.

Jeżeli systemy **A** i **B**, określone jak w definicji 2.3, są podobne, to powiemy, że $f_i$ i $f'_i$, dla $i \leqslant k$ są odpowiadającymi sobie operacjami, a $r_j$ i $r'_j$, dla $j \leqslant l$ są odpowiadającymi sobie relacjami w **A** i w **B**. ■

PRZYKŁAD 2.4

Niech $\mathbf{X} = \langle X^*, \bullet; < \rangle$ będzie systemem relacyjnym takim, że

$X^*$ jest zbiorem słów nad alfabetem $X$ włączając słowo puste $\emptyset$ (tzn. $X^* = \bigcup_{n \in N} X^n$);

$\bullet$ jest dwuargumentową operacją konkatenacji (tzn. jeżeli $w_1, w_2 \in X^*$, to słowo $w_1 w_2$, powstające przez dopisanie słowa $w_2$ do słowa $w_1$, jest wynikiem zastosowania operacji $\bullet$ na słowach $w_1$ i $w_2$);

$<$ jest relacją dwuczłonową taką, że $w_1 < w_2$ wtw $w_2 = w_1 w_3$ dla pewnego $w_3 \in X^*$.

System $\mathbf{X} = \langle X^*, \bullet; < \rangle$ jest podobny do systemu $\langle R, +; \leqslant \rangle$, gdzie $R$ oznacza zbiór liczb rzeczywistych, $+$ operację dodawania w $R$, a $\leqslant$ relację liniowego porządku w $R$. System $\mathbf{X} = \langle X^*, \bullet; < \rangle$ nie jest podobny do systemu $\langle R, +, *; \leqslant \rangle$ ani do systemu $\langle R, {}^{-1}; = \rangle$, gdzie $*$ oznacza operację mnożenia w $R$, a ${}^{-1}$ jednoargumentową operację odwracania liczb rzeczywistych. □

Niech $h$ będzie odwzorowaniem zbioru $A$ w zbiór $B$, $h: A \to B$. Jeżeli dla dowolnych różnych elementów zbioru $A$ wartości funkcji $h$ są różne (tzn. jeżeli $a \neq b$ pociąga za sobą $h(a) \neq h(b)$), to $h$ nazywamy *funkcją różnowartościową*. Jeżeli każdy element zbioru $B$ jest wartością funkcji $h$, to mówimy, że $h$ jest odwzorowaniem na zbiór $B$. □

DEFINICJA 2.4

Niech **A** i **B** będą podobnymi systemami relacyjnymi

$$\mathbf{A} = \langle A, f_1, \dots, f_k; r_1, \dots, r_l \rangle,$$
$$\mathbf{B} = \langle B, g_1, \dots, g_k; s_1, \dots, s_l \rangle$$

Homomorfizmem systemu relacyjnego **A** w system relacyjny **B** nazywamy funkcję $h: A \to B$ taką, że

(1) dla dowolnej $n_i$-argumentowej operacji $f_i$ w $A$ i dowolnych argu-

mentów $a_1, \ldots, a_{ni} \in A$, jeżeli $(a_1, \ldots, a_{ni}) \in Dom(f_i)$, to $(h(a_1), \ldots, h(a_{ni})) \in Dom(g_i)$ oraz

$$h(f_i(a_1, \ldots, a_{ni})) = g_i(h(a_1), \ldots, h(a_{ni}));$$

(2) dla dowolnej $m_j$-argumentowej relacji $r_j$ w $A$ i dowolnych argumentów $a_1, \ldots, a_{mj} \in A$, jeżeli $(a_1, \ldots, a_{mj}) \in r_j$, to $(h(a_1), \ldots, h(a_{mj})) \in s_j$. ■

UWAGA

Własność (1) w definicji (2.4) będziemy nazywać własnością zachowywania operacji. ■

PRZYKŁAD 2.5

Rozważmy system $\langle X^*, \bullet; < \rangle$ (por. przykład 2.4) i system liczb naturalnych $\langle N, +; \leqslant \rangle$ ze zwykłą interpretacją operacji $+$ i relacji $\leqslant$. Funkcja $h: X^* \to N$ taka, że

$h(\emptyset) = 0$, gdzie $\emptyset$ oznacza słowo puste,
$h(x) = 1$ dla $x \in X$
$h(w \bullet x) = h(w) + h(x)$ dla dowolnych $w \in X^*$ i $x \in X$

jest homomorfizmem odwzorującym $X$ w $N$.

Warunek (1) definicji homomorfizmu (definicja 2.4) jest spełniony w sposób oczywisty. Dla dowodu warunku (2) weźmy $w_1$, $w_2 \in X^*$, mamy wtedy

$$w_1 < w_2 \Rightarrow (\exists w_3)\, w_2 = w_1 \bullet w_3 \Rightarrow (\exists w_3)\, h(w_2) = h(w_1) + h(w_3) \Rightarrow \Rightarrow h(w_1) \leqslant h(w_2)$$ □

DEFINICJA 2.5

Jeżeli $h$ jest homomorfizmem odwzorowującym **A** na **B** oraz $h$ jest funkcją różnowartościową taką, że

(1) dla dowolnych elementów $a_1, \ldots, a_{ni}$ struktury **A** i dla dowolnej funkcji $f_i$, $n_i$-argumentowej w **A**

$$(a_1, \ldots, a_{ni}) \in Dom(f_i) \quad \text{wtw} \quad (h(a_1), \ldots, h(a_{ni})) \in Dom(g_i)$$

(2) dla dowolnych elementów $a_1, \ldots, a_{mj}$ struktury **A** i dowolnej relacji $r_j$, $m_j$-argumentowej w **A**

$$(a_1, \ldots, a_{mj}) \in r_j \quad \text{wtw} \quad (h(a_1), \ldots, h(a_{mj})) \in s_j$$

to $h$ nazywamy izomorfizmem. Jeżeli istnieje izomorfizm struktury **A** na strukturę **B**, to mówimy, że struktury **A** i **B** są izomorficzne i piszemy **A** $\equiv$ **B**. ■

PRZYKŁAD 2.6

Niech $+_m$ będzie dodawaniem modulo $m$ w zbiorze liczb całkowitych $\{0, 1, ..., m-1\}$. Rozważmy algebrę reszt modulo $m$

$$Z_m = \langle \{0, 1, ..., m-1\}, +_m \rangle$$

oraz algebrę ciągów zerojedynkowych długości $(n+1)$ z działaniem dwuargumentowym *plus*

$$\langle \{0, 1\}^{n+1}, plus \rangle$$

gdzie operacja plus jest zdefiniowana następująco: dla danych ciągów $a = \{a_0, ..., a_n\}$ oraz $b = \{b_0, ..., b_n\}$ wynikiem operacji *plus* dla argumentów $a$ i $b$ jest ciąg $\{c_0, ..., c_n\}$ taki, że dla $j = 0, ..., n$ zachodzi $c_j = (a_j + b_j + p_{j-1}) mod 2$, gdzie wektor reszt $p_j$ jest zdefiniowany następująco $p_{-1} = 0$ i dla $j \geqslant 0$

$$p_j = \begin{cases} 1 & \text{gdy } (a_j + b_j + p_{j-1}) \geqslant 2 \\ 0 & \text{w przeciwnym razie} \end{cases}$$

Niech $h$ będzie funkcją odwzorowującą zbiór $\{0, 1\}^{n+1}$ w zbiór $Z_m$, gdzie $m = 2^{n+1}$, taką, że $h(f) = \sum_{0 \leqslant i \leqslant n} 2^i * f(i)$ dla $f \in \{0, 1\}^{n+1}$.

Oczywiście dla dowolnego $f \in \{0, 1\}^{n+1}$ mamy

$$\sum_{0 \leqslant i \leqslant n} 2^i * f(i) < 2^{n+1}$$

tzn. $h(f) \in Z_m$, a więc $h$ jest odwzorowaniem w zbiór $Z_m$. Co więcej, dla każdej liczby $k \in Z_m$ istnieje i jest zdefiniowany jednoznacznie ciąg $f \in \{0,1\}^{n+1}$ taki, że $h(f) = k$. Rzeczywiście, wystarczy przyjąć

$$f(i) = (k \ div \ 2^i) mod 2,$$

gdzie *div* oznacza dzielenie całkowite.

Funkcja $h$ jest różnowartościowa, bo dla różnych ciągów $f$, $g$, jeżeli $i$ jest największą liczbą naturalną, taką że $f(i) \neq g(i)$, to dla $f(i) = 1$ i $g(i) = 0$, mamy

$$h(f) - h(g) = 2^i + \sum_{j<i} 2^j f(j) - \sum_{j<i} 2^j g(j) > 0$$

Ponadto, jeżeli $f$ *plus* $g = (c_0, ..., c_n)$, to

$$h(f plus g) = \sum_{i \leqslant n} 2^i c_i = \sum_{i \leqslant n} 2^i (a_i + b_i + p_{i-1}) mod 2 =$$

$$= \sum_{i \leqslant n} 2^i (a_i + b_i + p_{i-1} - 2(a_i + b_i + p_{i-1})) div 2 =$$

$$= \sum_{i \leqslant n} 2^i a_i + \sum_{i \leqslant n} 2^i b_i + \sum_{i \leqslant n} 2^i (p_{i-1} - 2(a_i + b_i + p_{i-1}) div 2)$$

Zauważmy, że dla dowolnego $0 \leqslant i \leqslant n$,

$$p_i-(a_i+b_i+p_{i-1})\,div2 = 0$$

oraz

$$(a_n+b_n+p_{n-1})\,div2 = \begin{cases} 1 & \text{gdy } (\sum_{i\leqslant n} 2^i a_i + \sum_{i\leqslant n} 2^i b_i) \geqslant m \\ 0 & \text{w przeciwnym razie} \end{cases}$$

Ostatecznie

$$h(f\,plus\,g) = h(f)+h(g)-2^{n+1}(a_n+b_n+p_{n-1})\,div2 = h(f)+h(g)$$

Zatem funkcja $h$ jest izomorfizmem odwzorowującym system

$\langle\{0,1\}^{n+1}, plus\rangle$ na $Z_m$. □

DEFINICJA 2.6

Kongruencją w systemie relacyjnym $\mathbf{A} = \langle A, f_1, ..., f_k;\ r_1, ..., r_l\rangle$ nazywamy dowolną relację dwuczłonową $\approx$ w $A$ taką, że

(1) $\approx$ jest relacją równoważności, tzn. relacją zwrotną, symetryczną i przechodnią;

(2) dla dowolnej $n_i$-argumentowej operacji $f_i$ w $\mathbf{A}$ i dowolnych elementów $a_1, ..., a_{ni}, a'_1, ..., a'_{ni} \in A$, jeżeli $a_j \approx a'_j$ dla $j \leqslant n_i$, to

$$(a_1, ..., a_{ni}) \in Dom(f_i) \quad \text{wtw} \quad (a'_1, ..., a'_{ni}) \in Dom(f_i)$$

oraz dla $(a_1, ..., a_{ni}) \in Dom(f_i)$ mamy

$$f_i(a_1, ..., a_{ni}) \approx f_i(a'_1, ..., a'_{ni})$$

(3) dla dowolnej relacji $r_j$, $m_j$-argumentowej i dowolnych elementów $a_1, ..., a_{mj}, a'_1, ..., a'_{mj} \in A$, jeżeli $a_i \approx a'_i$, dla $i = 1, ..., m_j$, to

$$(a_1, ..., a_{mj}) \in r_j \quad \text{wtw} \quad (a'_1, ..., a'_{mj}) \in r_j$$ ■

PRZYKŁAD 2.7

Rozważmy system relacyjny $\langle N, +, *\rangle$ oraz dla pewnego ustalonego $k$ relację $\approx_k$ w $N$ taką, że

$$m \approx_k n \quad \text{wtw} \quad (m\,mod\,k) = (n\,mod\,k) \quad \text{dla dowolnych } n, m \in N$$

Relacja $\approx_k$ jest relacją równoważności w $N$. Proste sprawdzenie tego faktu pozostawiamy czytelnikowi. Co więcej, relacja ta jest kongruencją w systemie $\langle N, +, *, 0\rangle$. Rzeczywiście, niech $m_1 \approx_k n_1$ i $m_2 \approx_k n_2$, wtedy istnieją takie liczby naturalne $i_1, i_2, j_1, j_2, r_1, r_2$, że

$$m_1 = i_1 * k + r_1 \qquad m_2 = i_2 * k + r_2$$
$$n_1 = j_1 * k + r_1 \qquad n_2 = j_2 * k + r_2$$

Zatem

$$m_1+m_2 = (i_1+i_2)*k+r_1+r_2$$
$$n_1+n_2 = (j_1+j_2)*k+r_1+r_2$$

Reszta z dzielenia $m_1+m_2$ przez $k$ jest równa reszcie z dzielenia $r_1+r_2$ przez $k$, a więc jest równa reszcie z dzielenia $n_1+n_2$ przez $k$. Wynika stąd, że

$$(m_1+m_2)\, mod\, k = (n_1+n_2)\, mod\, k$$

Podobnie dla mnożenia

$$m_1*m_2 = ((i_1*i_2)*k+r_1*i_2+r_2*i_1)*k+r_1*r_2$$

$$n_1*n_2 = ((j_1*j_2)*k+r_1*j_2+r_2*j_1)*k+r_1*r_2$$

Reszta z dzielenia $m_1*m_2$ przez $k$ jest równa reszcie z dzielenia $r_1*r_2$ przez $k$, a więc jest równa reszcie z dzielenia $n_1*n_2$ przez $k$. Stąd

$$(m_1*m_2)\, mod\, k = (n_1*n_2)\, mod\, k$$ □

PRZYKŁAD 2.8

Rozważmy system $\langle X^*, \bullet; < \rangle$ i homomorfizm $h: X^* \rightarrow N$ zdefiniowane jak w przykładzie 2.5. Niech $\approx$ będzie relacją równoważności określoną następująco: dla dowolnych $u, w \in X$

$$u \approx w \quad \text{wtw} \quad h(u) = h(w)$$

Zauważmy, że jeżeli $u \approx u'$ oraz $w \approx w'$, to

$$h(u \bullet w) = h(u)+h(w) = h(u')+h(w') = h(u' \bullet w')$$

czyli $u \bullet w \approx u' \bullet w'$. Jednakże, relacja $\approx$ nie jest kongruencją, jeśli zbiór $X$ zawiera więcej niż jeden element. Rzeczywiście, jeżeli $x$, $y$ są różnymi elementami $X$, to chociaż $x \approx x$, $xx \approx yy$ i $x < xx$, to równocześnie nie jest prawdą, że $x < yy$. □

DEFINICJA 2.7

Niech **A** będzie systemem relacyjnym postaci

$$\langle A, f_1, ..., f_k; r_1, ..., r_l \rangle$$

o sygnaturze $\langle n_1, ..., n_k; m_1, ..., m_l \rangle$ i niech $\approx$ będzie kongruencją w **A**. System

$$\mathbf{A}/\approx = \langle A/\approx, f_1^*, ..., f_k^*; r_1^*, ..., r_l^* \rangle$$

nazywamy *systemem ilorazowym*, jeżeli

(1) $A/\approx$ jest zbiorem klas abstrakcji relacji $\approx$, tzn. $A/\approx = \{[a] : a \in A\}$, gdzie $[a] = \{b \in A : a \approx b\}$;

(2) $f_i^*$ jest $n_i$-argumentową operacją w $A/\approx$ taką, że dla dowolnych $a_1, \ldots, a_{ni} \in A$

$$([a_1], \ldots, [a_{ni}]) \in Dom(f_i^*) \quad \text{wtw} \quad (a_1, \ldots, a_{ni}) \in Dom(f_i)$$

oraz dla dowolnych $([a_1], \ldots, [a_{ni}]) \in Dom(f_i^*)$

$$f_i^*([a_1], \ldots, [a_{ni}]) = [f_i(a_1, \ldots, a_{ni})]$$

(3) $r_j^*$ jest $m_j$-argumentową relacją w $A/\approx$ taką, że dla dowolnych $a_1, \ldots, a_{mj} \in A$

$$([a_1], \ldots, [a_{mj}]) \in r_j^* \quad \text{wtw} \quad (a_1, \ldots, a_{mj}) \in r_j$$ ■

UWAGA

Definicja funkcji $f_i^*$ nie zależy od wyboru reprezentantów klas $[a_1], \ldots, [a_{ni}]$, gdyż na mocy definicji 2.6 dla dowolnych elementów $x_i \in [a_i]$, $i = 1, \ldots, n_i$, $f_i(a_1, \ldots, a_{ni}) \approx f_i(x_1, \ldots, x_{ni})$, a zatem

$$[f_i(a_1, \ldots, a_{ni})] = [f_i(x_1, \ldots, x_{ni})]$$

Podobnie, definicja relacji $r_j^*$ nie zależy od wyboru reprezentantów klas $[a_1], \ldots, [a_{mj}]$, gdyż dla dowolnych $x_1, \ldots, x_{mj}$ takich, że $x_i \in [a_i]$

$$(a_1, \ldots, a_{mj}) \in r_j \quad \text{wtw} \quad (x_1, \ldots, x_{mj}) \in r_j$$ ■

PRZYKŁAD 2.9

Rozważmy kongruencję $\approx_m$ z przykładu 2.7 oraz system $\langle N, + \rangle$. System ilorazowy $\langle N/\approx_m, \otimes \rangle$ jest izomorficzny z systemem $Z_m$ zdefiniowanym w przykładzie 2.6. Jedyna operacja systemu, $\otimes$, jest określona następująco: dla dowolnych $a, b \in N$

$$[a] \otimes [b] = [a+b]$$

Po przyporządkowaniu każdej klasie $[a]$ liczby naturalnej będącej resztą z dzielenia $a$ przez $m$, otrzymujemy wzajemnie jednoznaczną funkcję $h$ z $N/\approx_m$ na $Z_m$

$$h([a]) = a \bmod m$$

która jest izomorfizmem rozważanych systemów. □

W dalszym ciągu, jeżeli nazwy relacji i funkcji w systemie relacyjnym nie będą miały istotnego znaczenia, będziemy stosowali notację $\mathbf{A} = \langle A, \Omega_F; \Omega_R \rangle$, gdzie $\Omega_F$ oznacza zbiór funkcji, a $\Omega_R$ zbiór relacji systemu $\mathbf{A}$.

W większości zastosowań, systemy relacyjne, z którymi będziemy mieli do czynienia, będą tzw. *systemami wielosortowymi*. Różne argumenty funkcji

i relacji w takim systemie mogą należeć do różnych (rozłącznych) zbiorów. Przykładem może być przestrzeń wektorowa, w której występują dwa typy zbiorów: wektory i skalary oraz operacja mnożenia wektora przez skalar określona w iloczynie *Wektory* × *Skalary*.

DEFINICJA 2.8

System relacyjny $\mathbf{A} = \langle A, \Omega_F; \Omega_R \rangle$ będziemy nazywać wielosortowym, dokładniej $t$-sortowym, jeżeli

(1) istnieją zbiory $A_1, \ldots, A_t$, zwane sortami, takie, że

$$A = A_1 \cup \ldots \cup A_t \quad \text{oraz } A_i \cap A_j = \emptyset \quad \text{dla } i \neq j,\ 1 \leqslant i, j \leqslant t$$

(2) dla dowolnej operacji $f \in \Omega_F$ $n$-argumentowej jest określony typ operacji $(i_1 \times \ldots \times i_n \to i_{(n+1)})$ taki, że

$$Dom(f) \subset A_{i1} \times \ldots \times A_{in} \quad \text{oraz} \quad Rg(f) \subset A_{i(n+1)})$$

(3) dla dowolnej relacji $r \in \Omega_R$ $m$-argumentowej jest określony typ relacji $(j_1 \times \ldots \times j_m)$ taki, że $r \subset (A_{j1} \times \ldots \times A_{jm})$. ■

Przykłady systemów wielosortowych podamy w następnym punkcie. Teraz zauważmy tylko, że każdy system wielosortowy jest systemem relacyjnym w sensie podanym na początku tego punktu. Jeżeli $f$ jest operacją typu $(i_1 \times \ldots \times i_n \to i_{n+1})$ w pewnym systemie wielosortowym, to przyjmując dla dowolnych $x_1, \ldots, x_n$

$$f'(x_1, \ldots, x_n) = \begin{cases} f(x_1, \ldots, x_n) & \text{dla } (x_1, \ldots, x_n) \in Dom(f) \\ \text{nieokreślone} & \text{w przeciwnym razie} \end{cases}$$

definiujemy odpowiadającą jej $n$-argumentową funkcję częściową.

Niech $\mathbf{A} = \langle A, \Omega_F; \Omega_R \rangle$ i $\mathbf{B} = \langle B, \Omega'_F; \Omega'_R \rangle$ będą dwoma systemami wielosortowymi odpowiednio o sortach $A_1, \ldots, A_t$ i $B_1, \ldots, B_{t'}$.

DEFINICJA 2.9

Powiemy, że systemy wielosortowe **A**, **B** są podobne, jeżeli

(1) **A**, **B** są podobnymi systemami relacyjnymi w sensie definicji 2.3;

(2) $t = t'$ oraz

(3) odpowiadające sobie operacje i relacje mają takie same typy. ■

Pojęcia homomorfizmu i izomorfizmu systemów wielosortowych są prawie identyczne z odpowiednimi pojęciami dla systemów relacyjnych. Drobne zmiany wynikają z podziału uniwersum systemu na podzbiory różnych typów.

DEFINICJA 2.10

Powiemy, że $h$ jest homomorfizmem (izomorfizmem) odwzorowującym system $t$-sortowy **A** w (na) system podobny **B**, jeżeli $h$ jest homomorfizmem (izomorfizmem) systemu relacyjnego **A** w system relacyjny **B** (por. definicja 2.4) i ponadto dla dowolnego $i \leqslant t$, $h(A_i) \subset B_i$ $\big(h(A_i) = B_i\big)$. ■

LEMAT 2.1

Dla dowolnych systemów wielosortowych podobnych **A**, **B**, **C**, jeżeli $h_1$ jest homomorfizmem odwzorowującym system **A** w system **B** oraz $h_2$ homomorfizmem odwzorowującym system **B** w system **C**, to funkcja $h$ taka, że $h(a) = h_2(h_1(a))$ dla dowolnego $a \in A$, będąca złożeniem funkcji $h_1$ i $h_2$, jest homomorfizmem odwzorowującym system **A** w system **C**. ■

Bardzo istotne dla naszych dalszych rozważań będzie pojęcie kongruencji. W systemach wielosortowych definicja kongruencji jest nieco inna niż w systemach relacyjnych. Musimy zadbać o to, by podział implikowany przez relację kongruencji nie łączył w jednej klasie abstrakcji elementów różnych typów.

DEFINICJA 2.11

Kongruencją w systemie wielosortowym **A** będziemy nazywać relację równoważności w $A$ spełniającą warunki definicji 2.6 oraz taką, że dla dowolnego $a \in A$, jeżeli $a \in A_i$, to klasa abstrakcji wyznaczona przez element $a$ jest zawarta w $A_i$, $[a] \subset A_i$. ■

Niech **A** będzie systemem wielosortowym o sortach $A_1, \ldots, A_t$ i niech $\approx$ będzie kongruencją w **A**. Oznaczmy przez $B_i$ zbiór $A_i/\approx$. Na mocy definicji kongruencji $B_i \cap B_j = \emptyset$ dla dowolnych $i \neq j$. Niech $f$ będzie dowolną $n$-argumentową operacją w **A** typu $(i_1 \times \ldots \times i_n \to i_{n+1})$. Zdefiniujemy, odpowiadającą $f$, operację $f^*$ w $B = B_1 \cup \ldots \cup B_t$ następująco:

$$Dom(f^*) = \{([a_1], \ldots, [a_n]) : (a_1, \ldots, a_n) \in Dom(f)\}$$

oraz dla $([a_1], \ldots, [a_n]) \in Dom(f^*)$

$$f^*([a_1], \ldots, [a_n]) = [f(a_1, \ldots, a_n)]$$

Ponieważ dla dowolnego $i \leqslant t$ i dla dowolnego $a \in A_i$, $[a] \subset A_i$, więc $[a] \in B_i$. Wynika stąd, że $f^* \subset B_1 \times \ldots \times B_n \times B_{n+1}$ i w konsekwencji typ funkcji $f^*$ jest taki sam jak funkcji $f$.

Podobnie, dla dowolnej relacji $r$ typu $(j_1 \times \ldots \times j_n)$ w $A$ zdefiniujemy relację $r^*$ następująco: dla dowolnych $(a_1, \ldots, a_n) \subset A_{j1} \times \ldots \times A_{jn}$,

$$([a_1], \ldots, [a_n]) \in r^* \quad \text{wtw} \quad (a_1, \ldots, a_n) \in r$$

Z przyjętej definicji wynika, że $r^* \subset B_{j1} \times \ldots \times B_{jn}$, a więc $r^*$ ma taki sam typ jak $r$.

Oznaczmy przez $\Omega_F^*$ zbiór wszystkich funkcji $f^*$ odpowiadających funkcjom $f \in \Omega_F$ i przez $\Omega_R^*$ zbiór wszystkich relacji $r^*$ odpowiadających relacjom $r \in \Omega_R$.

DEFINICJA 2.12

System $\mathbf{A}/\approx\, = \langle B, \Omega_F^*; \Omega_R^* \rangle$, gdzie $B = B_1 \cup \ldots \cup B_t$, nazywamy wielosortowym systemem ilorazowym. ■

Oczywiście, na mocy przyjętych definicji, system $\mathbf{A}/\approx$ jest podobny do systemu $\mathbf{A}$. Ponadto zachodzi następująca, wielokrotnie wykorzystywana własność.

LEMAT 2.2

Niech $\mathbf{A}$ będzie systemem wielosortowym, a $\mathbf{A}/\approx$ system ilorazowym zdefiniowanym jak wyżej. Funkcja $h: \mathbf{A} \to \mathbf{A}/\approx$ taka, że $h(a) = [a]$ dla $a \in \mathbf{A}$, jest homomorfizmem odwzorowującym $\mathbf{A}$ na $\mathbf{A}/\approx$. ■

DOWÓD

Odwzorowanie $h$ jest niewątpliwie funkcją całkowitą odwzorowującą $\mathbf{A}$ na $\mathbf{A}/\approx$ (zauważmy, że nie jest to jednak funkcja różnowartościowa, gdyż dla wszystkich $b \in [a]$ mamy $h(b) = [a]$).

Rozważmy dowolną operację $f$ systemu $\mathbf{A}$. Niech $(i_1 \times \ldots \times i_n \to i_{n+1})$ będzie typem $f$ oraz niech $(a_1, \ldots, a_n) \in Dom(f)$. Wtedy na mocy przyjętych definicji mamy $([a_1], \ldots, [a_n]) \in Dom(f^*)$ oraz

$$h(f(a_1, \ldots, a_n)) = f^*(h(a_1), \ldots, h(a_n)) = f^*([a_1], \ldots, [a_n]) = [f(a_1, \ldots, a_n)]$$

Rozważmy dowolną relację $r$ typu $(i_1 \times \ldots \times i_n)$ w $\mathbf{A}$. Przypuśćmy, że $(a_1, \ldots, a_n) \in r$. Wtedy $([a_1], \ldots, [a_n]) \in r^*$ na mocy definicji systemu ilorazowego (definicja 2.12), a więc $(h(a_1), \ldots, h(a_n)) \in r^*$. Z powyższych rozważań wynika, że $h$ jest homomorfizmem. Homomorfizm ten będziemy nazywać homomorfizmem naturalnym. □

UWAGA

System wielosortowy $\mathbf{A} = \langle A, \Omega_F; \Omega_R \rangle$ o sortach $A_1, \ldots, A_t$ może być traktowany jako system relacyjny jednosortowy $\mathbf{A}'$

$$\mathbf{A}' = \langle A, \Omega_F; \Omega_R \cup \{typ_i : i \leqslant t\} \rangle$$

w którym sygnaturę rozszerzono o $t$ relacji jednoargumentowych $typ_i$ określonych następująco:

$$typ_i(a) \equiv a \in A_i$$

Relacje $typ_i$ dla $i \leqslant t$ definiują występujące w strukturze **A** typy. Założenie rozłączności typów można zdefiniować następującą implikacją

$$typ_i(a) \rightarrow non\ typ_j(a)$$

dla dowolnych $a \in A$, $i \neq j$ i $i,j \leqslant t$. Jeżeli $f$ jest funkcją typu $A_{i1} \times \ldots$ $\ldots \times A_{in} \rightarrow A_{i(n+1)}$, to warunkiem koniecznym należenia ciągu $(a_1, \ldots, a_n)$ do dziedziny funkcji $f$ jest wówczas koniunkcja

$$typ_{i1}(a_i) \cap \ldots \cap typ_{in}(a_n)$$

Podobny warunek można sformułować dla relacji.

Zauważmy ponadto, że jeżeli $\approx$ jest kongruencją w systemie $A'$, to $\approx$ jest również kongruencją w **A**. ■

## Systemy relacyjne do reprezentacji zbiorów i ciągów skończonych 2.3

Punkt ten zawiera kilka wybranych przykładów struktur danych pozwalających operować na zbiorach i ciągach skończonych.

DEFINICJA 2.13

*Standardową strukturą słowników* nazywamy system relacyjny dwusortowy

$$\mathbf{D} = \langle E \cup D, insert, delete; member, empty \rangle$$

taki, że $E \cap D = \emptyset$ i $D = Fin(E)$, tzn. $D$ jest zbiorem wszystkich skończonych podzbiorów $E$,

$insert$ oraz $delete$ są funkcjami typu $(E \times D \rightarrow D)$,
$member$ jest relacją dwuczłonową typu $(E \times D)$,
$empty$ jest relacją jednoczłonową typu $(D)$,
$Dom(delete) = E \times (D - \{\emptyset\})$, $Dom(insert) = E \times D$

oraz dla dowolnych $e \in E$ i $d \in D$

$insert(e, d) = d \cup \{e\}$,
$delete(e, d) = d - \{e\}$, jeżeli $d \neq \emptyset$
$d \in empty$ wtw $d = \emptyset$
$(e, d) \in member$ wtw $e \in d$. ■

Struktura słowników jest ważną i często używaną w informatyce strukturą danych. Stosujemy ją, gdy mamy do czynienia ze skończonymi zbiorami, a akcje, które zamierzamy wykonywać, polegają na zwiększaniu zbioru o nowy element lub usuwaniu elementu ze zbioru.

DEFINICJA 2.14

*Standardową strukturą stosów* nazywamy system relacyjny dwusortowy

$$S = \langle E \cup S, push, pop, top; empty, \underset{E}{=} \rangle$$

taki, że $E$ jest dowolnym niepustym zbiorem, $S$ jest zbiorem wszystkich skończonych ciągów o elementach ze zbioru $E$ oraz

*push* jest operacją typu $(E \times S \to S)$,
*pop* jest operacją typu $(S \to S)$,
*top* jest operacją jednoargumentową typu $(S \to E)$,
*empty* jest jednoczłonową relacją typu $(S)$,
$\underset{E}{=}$ jest dwuczłonową relacją typu $(E \times E)$

oraz dla dowolnych $e \in E$ i $s \in S$, jeżeli $s = (e_1, \ldots, e_n)$ to

$pop(s) = (e_2, \ldots, e_n)$
$push(e, s) = (e, e_1, \ldots, e_n)$,
$top(s) = e_1$.

Jeżeli $s$ jest ciągiem pustym, to wartości $pop(s)$ i $top(s)$ nie są określone.

$$\underset{E}{=} \; = \{(e, e): e \in E\}$$

$s \in empty$ wtw $s$ jest pustym ciągiem.

Elementy zbioru $S$ będziemy nazywać stosami. ■

Stosy odgrywają podstawową rolę w konstrukcji kompilatorów języków programowania. Struktura ta jest równie ważna podczas analizy syntaktycznej tekstu programu źródłowego jak i podczas wykonywania obliczeń. Stos, którego elementami są tzw. rekordy aktywacji, jest podstawowym narzędziem realizacji procedur.

Inną często stosowaną strukturą, która również dotyczy ciągów skończonych, jest struktura kolejek.

DEFINICJA 2.15

*Standardowa strukturą kolejek* nazywamy system dwusortowy

$$\langle E \cup Q, put, out, first; em, \underset{E}{=}, \underset{Q}{=} \rangle$$

gdzie $E$ jest zbiorem niepustym elementów, $Q$ jest zbiorem ciągów skoń-

czonych o elementach ze zbioru $E$, włączając zbiór pusty $\emptyset$, a operacje i relacje systemu są określone jak następuje:

*put* jest dwuargumentową operacją typu $(Q \times E \to Q)$
*out* jest jednoargumentową operacją typu $(Q \to Q)$
*first* jest jednoargumentową operacją typu $(Q \to E)$
*em* jest relacją typu $(Q)$
$=_E$ jest relacją identyczności typu $(E \times E)$
$=_Q$ jest relacją identyczności typu $(Q \times Q)$

oraz dla dowolnych $q = (e_1, ..., e_n) \in Q$ oraz $e \in E$ mamy

$put(q,e) = (e_1, ..., e_n, e)$ $Dom(put) = Q \times E$
$out(q) = (e_2, ..., e_n)$, $Dom(out) = Q - \emptyset$
$first(q) = e_1$ $Dom(first) = Q - \emptyset$
$q' \in em$ wtedy i tylko wtedy, gdy $q'$ jest ciągiem pustym $\emptyset$. ■

Kolejka jest strukturą używaną często w programach symulacyjnych. W systemach operacyjnych tworzy się kolejki procesów oczekujących na dostęp, np. do drukarki.

W bardzo wielu zastosowaniach używa się struktur zwanych drzewami. Poniżej przedstawimy dwa typy struktur działających na drzewach.

DEFINICJA 2.16

Niech $At$ będzie zbiorem niepustym, którego elementy będziemy nazywać atomami. Niech *Tree* będzie najmniejszym zbiorem wyrażeń zawierającym dla każdego $a \in At$ wyrażenie postaci $(a)$ oraz element specjalny *none* i takim, że dla dowolnych dwu elementów $t_1, t_2 \in Tree$, $t_1, t_2 \neq none$ wyrażenie $(t_1 \bullet t_2)$ jest elementem zbioru *Tree*. Elementy zbioru *Tree* będziemy nazywać drzewami binarnymi. ■

DEFINICJA 2.17

*Standardową strukturą drzew binarnych* nazywamy system dwusortowy

$$\langle At \cup Tree, cons, left, right; atom, empty \rangle$$

taki, że

*cons* jest operacją typu $(Tree \times Tree \to Tree)$
*left* jest operacją typu $(Tree \to Tree)$
*right* jest operacją typu $(Tree \to Tree)$
*empty* i *atom* są relacjami jednoargumentowymi typu $(Tree)$, tzn. pewnymi podzbiorami zbioru *Tree*

oraz dla dowolnych $t \in Tree$ i $t_1, t_2 \in Tree - \{none\}$ spełnione są następujące warunki:

$cons(t_1, t_2) = (t_1 \bullet t_2)$, $\quad Dom(cons) = (Tree - \{none\}) \times (Tree - \{none\})$
$left((t_1 \bullet t_2)) = t_1$, $\quad Dom(left) = Tree - \{none\}$
$right((t_1 \bullet t_2)) = t_2$, $\quad Dom(right) = Tree - \{none\}$
$right(t) = left(t) = none$, $\quad$ gdy $t \in atom$
$t \in atom$ wtw $t = (a)$ $\quad$ dla pewnego $a \in At$
$t \in empty$ wtw $t = none$. ■

Dalej omówimy nieco inny typ drzew binarnych, które znajdują duże zastosowanie w zadaniach związanych z porządkowaniem zbiorów.

Niech $Et$ będzie zbiorem liniowo uporządkowanym przez pewną relację $\leqslant$. Elementy zbioru $Et$ będziemy nazywać etykietami. Niech $Tr$ będzie najmniejszym zbiorem zawierającym wyrażenie ( ) i takim, że jeżeli $e \in Et$ i $t_1, t_2 \in Tr$, to $(t_1\, e\, t_2) \in Tr$. Elementy zbioru $Tr$ będziemy nazywali etykietowanymi drzewami binarnymi. Jeżeli drzewo $t$ ma postać $(t_1\, e\, t_2)$, to $t_1$ nazywamy jego lewym poddrzewem, a $t_2$ prawym poddrzewem. Element $e$ nazywamy etykietą drzewa $t$.

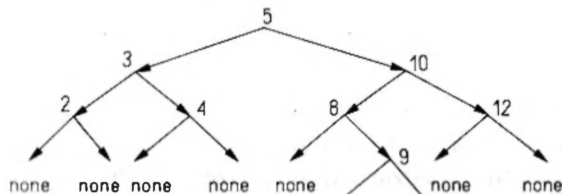
Drzewo $t \in Tr$ nazywamy *drzewem binarnych poszukiwań*, jeżeli jest postaci ( ) lub gdy jest postaci $(t_1\, e\, t_2)$ i są spełnione następujące warunki:

(1) dla dowolnej etykiety $e'$ występującej w $t_1$, $e' \leqslant e$,
(2) dla dowolnej etykiety $e'$ występującej w $t_2$, $e < e'$,
(3) $t_1$ i $t_2$ są drzewami binarnych poszukiwań.

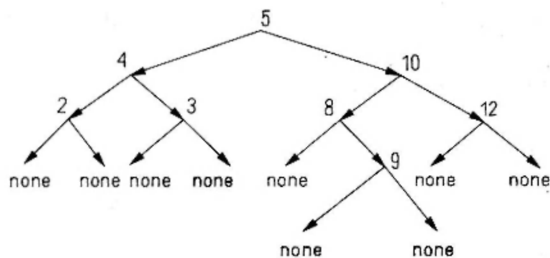
Niech $BST$ będzie zbiorem drzew binarnych poszukiwań. Każdy element $t$ tego zbioru jest więc drzewem binarnym skończonym. Wierzchołki takiego drzewa są etykietowane elementami zbioru $Et$. Dla każdego poddrzewa $t'$ drzewa $t$ etykieta przypisana korzeniowi tego poddrzewa jest większa od etykiety przyporządkowanej każdemu wierzchołkowi lewego poddrzewa drzewa $t'$. Podobnie etykieta przyporządkowana dowolnemu wierzchołkowi z prawego poddrzewa drzewa $t'$ jest większa od etykiety przyporządkowanej korzeniowi drzewa $t'$.

PRZYKŁAD 2.10

Rysunek 2.2 przedstawia drzewo binarnych poszukiwań etykietowane liczbami naturalnymi. Drzewo binarne przedstawione na rys. 2.3 nie jest drzewem binarnych poszukiwań. □




Rys. 2.2




RYS. 2.3

DEFINICJA 2.18

*Standardową strukturą drzew binarnych poszukiwań* nazywamy system dwusortowy

$$\langle Et \cup BST, val, left, right, new, upl, upr; isnone, \leqslant, = \rangle$$

gdzie

*val* jest jednoargumentową operacją typu $(BST \rightarrow Et)$
*left*, *right* są operacjami typu $(BST \rightarrow BST)$
*new* jest jednoargumentową operacją typu $(Et \rightarrow BST)$
*upl* i *upr* są dwuargumentowymi operacjami typu $(BST \times BST \rightarrow BST)$

oraz dla dowolnego $e \in Et$ i dowolnych drzew $t_1, t_2 \in BST$

$val(t_1\, e\, t_2) = e$, $Dom(val) = BST - \{()\}$
$left(t_1\, e\, t_2) = t_1$
$right(t_1\, e\, t_2) = t_2$, $Dom(left) = Dom(right) = BST - \{()\}$,
$new(e) = (()\, e\, ())$ dla dowolnego $e \in E$

$$upl(t, t_1\, e\, t_2) = \begin{cases} (t\, e\, t_2) & \text{wtw } (t\, e\, t_2) \text{ jest elementem } BST \\ \text{nieokreślony} & \text{w pozostałych przypadkach} \end{cases}$$

$$upr(t, t_1\, e\, t_2) = \begin{cases} (t_1\, e\, t) & \text{wtw } (t_1\, e\, t) \text{ jest elementem } BST \\ \text{nieokreślony} & \text{w pozostałych przypadkach} \end{cases}$$

*isnone* jest relacją jednoczłonową w zbiorze *BST*, spełnioną tylko dla elementu (),

$\leqslant$ jest relacją liniowego porządku w zbiorze *Et*, a $=$ jest identycznością w *Et*. ■

# O języku formalnym 2.4

Badanie własności konkretnych systemów relacyjnych wymaga języka odpowiedniego do wyrażania własności operacji i relacji systemu. Językiem, którego zwykle używamy w matematyce i który będzie stanowił bazę naszych dalszych rozważań, jest język sformalizowany pierwszego rzędu, por. [33, 44, 45]. Formalna definicja tego języka jest treścią tego punktu.

DEFINICJA 2.19

Alfabetem języka pierwszego rzędu nazywamy zbiór $A$ będący sumą rozłącznych zbiorów

$V$ — zmiennych indywiduowych,
$V_0$ — zmiennych zdaniowych,
$P$ — symboli relacyjnych (predykatów), $P = (\varrho_j)_{j \in J}$
$\Phi$ — symboli funkcyjnych (funktorów), $\Phi = (\varphi_i)_{i \in I}$
$\{\wedge, \vee, \neg, \Rightarrow\}$ — symboli logicznych, nazywanych odpowiednio alternatywą, koniunkcją, negacją, implikacją,
$\{\forall, \exists\}$ — symboli dla kwantyfikatorów nazywanych odpowiednio kwantyfikatorem ogólnym i szczegółowym,
$\{), (\}$ — symboli pomocniczych. ■

W dalszym ciągu będziemy zakładać, że zbiór zmiennych indywiduowych jest sumą niepustych, rozłącznych zbiorów $V_i$, $1 \leqslant i \leqslant tt$ dla pewnej liczby naturalnej $tt$. Jeżeli $x \in V_i$, to powiemy, że $x$ jest zmienną typu $i$. Każdy predykat $\varrho \in P$ ma określony typ $t(\varrho)$ postaci $(i_1 \times \ldots \times i_n \rightarrow 0)$, gdzie $n$ jest liczbą argumentów $\varrho$, a $i_1, \ldots, i_n$ są typami jego argumentów, $1 \leqslant i_1, \ldots, i_n \leqslant tt$. Każdy funktor $\varphi \in \Phi$ ma określony typ $t(\varphi)$ postaci $(i_1 \times \ldots \times i_n \rightarrow i_{n+1})$, gdzie $n$ jest liczbą argumentów funktora $\varphi$, a $i_1, \ldots, i_n$ są typami jego argumentów, $1 \leqslant i_1, \ldots, i_n, i_{n+1} \leqslant tt$. Zakładamy ponadto, że alfabet $A$ jest zbiorem co najwyżej przeliczalnym.

DEFINICJA 2.20

Układ $\langle tt, \{t(\varphi)\}_{\varphi \in \Phi}, \{t(\varrho)\}_{\varrho \in P} \rangle$ nazywamy sygnaturą języka. ■

UWAGA

Definicja sformalizowanego języka, podana dalej, różni się nieco od zwykle przyjmowanej dla języka pierwszego rzędu. Ze względu na późniejsze zastosowania, włączyliśmy do alfabetu zmienne różnych typów i w szczególności zbiór zmiennych zdaniowych. ■

Zbiór wyrażeń poprawnych języka pierwszego rzędu składa się ze zbioru termów i zbioru formuł. Termy umożliwią konstruowanie złożonych wyrażeń funkcyjnych, a formuły będą służyły do wyrażania własności opisywanych pojęć.

DEFINICJA 2.21

Zbiór termów $T$ (inaczej: wyrażeń algebraicznych) jest to najmniejszy zbiór zawierający zbiór zmiennych indywiduowych $V$ i taki, że jeśli $\varphi$ jest

funktorem typu $(i_1 \times \ldots \times i_n \to i_{n+1})$, a $\tau_1, \ldots, \tau_n$ są termami ze zbioru $T$ odpowiednio typów $i_1, \ldots, i_n$, to $\varphi(\tau_1, \ldots, \tau_n)$ jest termem typu $i_{n+1}$. ■

DEFINICJA 2.22

Zbiór formuł $F$ jest to najmniejszy zbiór zawierający zbiór zmiennych zdaniowych oraz zbiór formuł elementarnych postaci $\varrho(\tau_1, \ldots, \tau_m)$, gdzie $\varrho$ jest predykatem typu $(i_1 \times \ldots \times i_m \to 0)$, a $\tau_1, \ldots, \tau_m$ są dowolnymi termami odpowiednio typów $i_1, \ldots, i_m$, i taki, że

(1) jeżeli $\alpha, \beta$ są formułami, to $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $\neg\alpha$, $(\alpha \Rightarrow \beta)$ są formułami,

(2) jeżeli $\alpha$ jest formułą, a $x$ jest zmienną indywiduową, to $(\forall x)\alpha$, $(\exists x)\alpha$ są formułami.

Formuły, w których nie występują symbole kwantyfikatorów, nazywamy *formułami otwartymi.* ■

PRZYKŁAD 2.11

Jeżeli $x, y, z \in V$ oraz $+$, $*$ są dwuargumentowymi symbolami funkcyjnymi, a $>$ jest dwuczłonowym symbolem relacyjnym, w jednosortowym języku $L$, to wyrażenia

$$*(+(x,y),z) \qquad +(*(x,z),*(y,z))$$

są termami, a wyrażenie

$$(\forall y)(\exists x) > (*(+(x,y),z), +(*(x,z),*(y,z)))$$

jest formułą rozważanego języka.

Jeśli zastosujemy zwykłą notację, w której symbol operacji / relacji dwuczłonowej występuje nie przed a między argumentami, to powyższe wyrażenia przyjmą postać

$$(x+y)*z \qquad (x*z)+(y*z)$$
$$(\forall y)(\exists x)(x+y)*z > ((x*z)+(y*z))$$

W dalszym ciągu we wszystkich przykładach, w których będą występowały symbole relacji lub operacji dwuczłonowych, będziemy stosować tę ostatnią konwencję. □

Jeżeli formuła ma postać $(\exists x)\alpha$ lub $(\forall x)\alpha$, to $\alpha$ będziemy nazywać odpowiednio zakresem kwantyfikatora szczegółowego lub ogólnego. Jeżeli zmienna indywiduowa $x$ pojawia się w zakresie kwantyfikatora, to mówimy, że $x$ jest zmienną związaną lub dokładniej zmienną związaną przez kwantyfikator ogólny w przypadku formuły $(\forall x)\alpha$ i zmienną związaną przez kwantyfikator szczegółowy w przypadku formuły $(\exists x)\alpha$. O kwantyfikatorach mówimy, że wiążą wystąpienia zmiennej $x$ w formule $\alpha$.

Jeżeli zmienna indywiduowa $x$ występuje w formule $\beta$ i nie jest związana przez żaden kwantyfikator, to powiemy, że $x$ jest zmienną wolną w $\beta$ i piszemy $\beta(x)$. Napis $\beta(x_1,\dots,x_n)$ oznacza, że $x_1,\dots,x_n$ są zmiennymi wolnymi w formule $\beta$. Zbiór wszystkich zmiennych wolnych występujących w formule $\beta$ będziemy oznaczać przez $FV(\beta)$. Zbiór wszystkich zmiennych występujących w termie $\tau$ lub w formule $\alpha$ będziemy oznaczać odpowiednio przez $V(\tau)$ i $V(\alpha)$.

PRZYKŁAD 2.12

Rozważmy język pierwszego rzędu jak w przykładzie 2.11 i formułę $\alpha$ postaci $(\beta \vee \gamma)$, gdzie

$$\beta = (\forall x)(\exists y)((x+y)*z > y), \quad \gamma = (x > y*(x+y))$$

W formule $\beta$ występuje jedna zmienna wolna $z$ i dwie zmienne związane $x, y$. Wszystkie zmienne występujące w formule $\gamma$ są zmiennymi wolnymi. W formule $\alpha$ występują trzy zmienne wolne $x$, $y$, $z$ i dwie zmienne związane $x$, $y$. Zauważmy, że pewne wystąpienia, np. zmiennej $x$, są w formule $\alpha$ związane, a inne wolne:

$$((\forall x)\ (\exists y)((x+y)*z > y) \vee (x > y*(x+y)))$$

związane wystąpienia $x$ wolne wystąpienia $x$ □

Formalne przedstawienie języka pierwszego rzędu zakończymy następującą definicją.

DEFINICJA 2.23

Językiem pierwszego rzędu będziemy nazywać układ $L = \langle A, T, F\rangle$, w którym A jest alfabetem, T jest zbiorem termów, a F jest zbiorem formuł zbudowanych nad alfabetem A. ■

UWAGA

Zmieniając alfabet otrzymujemy inny zbiór termów i inny zbiór formuł, a więc inny język pierwszego rzędu. Powyższa definicja jest w istocie opisem klasy języków pierwszego rzędu. Dwa języki tej klasy mogą mieć istotnie różne alfabety. W dalszym ciągu będziemy stosować następujące skróty: **true** dla formuły $(p \vee \neg p)$ i **false** dla formuły $(p \wedge \neg p)$, gdzie $p$ jest zmienną zdaniową w rozważanym języku. Jeżeli wśród predykatów języka L występuje dwuargumentowy predykat $=$, to fakt ten będziemy podkreślać pisząc $L_=$. ■

Zarówno termy, jak i formuły są jedynie wyrażeniami formalnymi,

którym nadamy sens, jeżeli ustalimy interpretację dla wszystkich symboli w nich występujących. Interpretację wyrażeń poprawnych rozważanego języka będziemy nazywać *semantyką*. Semantyka języka — to sposób rozumienia jego wyrażeń.

Niech L będzie językiem pierwszego rzędu o sygnaturze $\langle tt, \{t(\varphi)\}_{\varphi\in\Phi}, \{t(\varrho)\}_{\varrho\in P}\rangle$, a $A$ niepustym zbiorem będącym sumą rozłącznych zbiorów $A_i$ dla $1 \leqslant i \leqslant tt$. Dla dowolnego funktora $\varphi$ języka L typu $t(\varphi) = (i_1 \times \ldots \times i_n \to i_{n+1})$ niech $\varphi_A$ oznacza $n$-argumentową funkcję częściową w $A$,

$$\varphi_A : A_{i1} \times \ldots \times A_{in} \to A_{i(n+1)}$$

i dla dowolnego predykatu $\varrho$ typu $t(\varrho) = (i_1 \times \ldots \times i_m \to 0)$ niech $\varrho_A$ oznacza $m$-członową relację w $A$,

$$\varrho_A \subset A_{i1} \times \ldots \times A_{im}$$

Otrzymany w ten sposób system relacyjny

$$\mathbf{A} = \langle A, (\varphi_A)_{\varphi\in\Phi}, (\varrho_A)_{\varrho\in P}\rangle$$

ma sygnaturę zgodną z sygnaturą języka L. Funkcję $\varphi_A$ i relację $\varrho_A$ będziemy nazywać interpretacjami symbolu funkcyjnego $\varphi$ i symbolu relacyjnego $\varrho$ w systemie **A**. O takim systemie relacyjnym wielosortowym **A** będziemy mówili krótko, że jest *strukturą danych dla języka* L.

Jeżeli $L_=$ jest językiem, w którym występuje $=$, to w strukturze danych dla $L_=$ predykat $=$ będzie zwykle interpretowany jako identyczność.

## PRZYKŁAD 2.13

Niech $\varphi$, $\psi$ będą dwuargumentowymi funktorami języka L, a $x$, $y$, $z$ zmiennymi indywiduowymi. Niech **A** będzie systemem relacyjnym, którego uniwersum stanowi zbiór liczb rzeczywistych, a operacjami systemu są operacja dodawania $+$ i operacja mnożenia $*$. Ustalmy, że interpretacją symbolu $\varphi$ jest $+$, a interpretacją symbolu $\psi$ jest $*$. Przy powyższych założeniach term $\psi(\varphi(x, y), z)$ można rozumieć w **A** jako funkcję trójargumentową $\psi_A(\varphi_A(x, y), z)$ określoną w zbiorze liczb rzeczywistych, która trójce liczb $a_1$, $a_2$, $a_3$ przyporządkowuje liczbę rzeczywistą będącą wartością wyrażenia arytmetycznego $(a_1 + a_2) * a_3$. □

Uogólniając te obserwacje można powiedzieć, że każdy term jest wzorcem pewnej funkcji częściowej zdeterminowanej przez znaczenie funktorów w nim występujących. Wartości tej funkcji zależą od przyjętych wartości zmiennych, od tzw. wartościowania zmiennych. Dalej przedstawimy dokładną definicję tego pojęcia.

DEFINICJA 2.24

Wartościowaniem w strukturze danych **A** dla języka L nazywamy dowolną funkcję

$$v: V \cup V_0 \to A \cup B_0$$

taką, że $v(x) \in A_i$ dla $x \in V_i$, $1 \leqslant i \leqslant tt$ oraz $v(q) \in B_0$ dla $q \in V_0$, tzn. wartościowanie przypisuje zmiennej indywiduowej element odpowiedniego typu z uniwersum systemu **A**, a zmiennej zdaniowej element dwuelementowej algebry Boole'a $B_0$. Zbiór wszystkich wartościowań w strukturze **A** oznaczymy przez W(**A**). ■

DEFINICJA 2.25

Dla dowolnego termu $\tau$ języka L i dla dowolnej struktury danych **A** znaczeniem (semantyką) termu $\tau$ w **A** jest funkcja częściowa $\tau_A$, taka, że

$$\tau_A : \mathrm{W}(\mathbf{A}) \to \mathbf{A}$$

oraz dla dowolnego wartościowania $v$ w **A**

$$x_A(v) = v(x) \quad \text{dla} \quad x \in V \cup V_0$$

$$\varphi(\tau_1, ..., \tau_n)_A(v) = \begin{cases} \varphi_A(\tau_{1A}(v), ..., \tau_{nA}(v)), \text{ jeżeli } \tau_{1A}(v), ..., \tau_{nA}(v) \text{ oraz} \\ \qquad \varphi_A(\tau_{1A}(v), ..., \tau_{nA}(v)) \text{ są określone} \\ \text{nieokreślone w przeciwnym razie} \end{cases}$$

gdzie $\tau_1, ..., \tau_n$ są dowolnymi termami, a $\varphi$ $n$-argumentowym funktorem. ■

*Powtórzmy jeszcze raz*

(1) Dowolny term $\tau$ wyznacza w strukturze danych **A** funkcję częściową $\tau_A$, która dowolnemu wartościowaniu $v$ z dziedziny funkcji $\tau_A$, przyporządkowuje element $\tau_A(v)$ należący do uniwersum struktury **A**;

(2) Wartością termu postaci $x$ jest, przy wartościowaniu $v$ w strukturze **A**, wartość funkcji $v$ dla $x$;

(3) Wartością termu $\tau$ postaci $\varphi(\tau_1, ..., \tau_n)$ jest wynik operacji $\varphi_A$ dla argumentów $a_1, ..., a_n$, jeżeli wartości funkcji $\tau_{1A}, ..., \tau_{nA}$ są określone dla $v$ i równe odpowiednio $a_1, ..., a_n$ oraz $(a_1, ..., a_n)$ należy do dziedziny funkcji $\varphi_A$;

(4) Wartość termu nie zależy od wartości zmiennych, które w nim nie występują, możemy zatem każdy term traktować jako funkcję częściową o skończonej liczbie argumentów. Term, w którym występują zmienne $x_1, ..., x_n$, wyznacza w strukturze danych $n$-argumentową funkcję częściową zależną od zmiennych $x_1, ..., x_n$.

W podobny, rekurencyjny sposób przypisujemy znaczenie formułom. Dokładną definicję poprzedzimy prostym przykładem.

PRZYKŁAD 2.14

Niech L będzie językiem, a **A** strukturą danych, opisanymi jak w przykładzie 2.13. Załóżmy dodatkowo, że w języku L istnieje predykat $\varrho$ dwuczłonowy i że odpowiada mu w systemie **A** relacja $<$. Rozważmy formułę

$$(\varrho(\psi(x, x), \psi(y, y)) \Rightarrow \varrho(x, y))$$

Jeżeli zinterpretujemy ją w systemie **A**, to otrzymamy wyrażenie $(x * x < < y * y \Rightarrow x < y)$, które może być rozumiane jako relacja dwuczłonowa zachodząca między liczbami rzeczywistymi $a$, $b$, wtedy i tylko wtedy, gdy $a < b$ lub $|a| \geqslant |b|$, lub jako funkcja dwu zmiennych $x$, $y$, która przyjmuje wartość prawda, gdy $x < y$ lub $|x| \geqslant |y|$. □

Uogólniając obserwację zawartą w tym przykładzie przyjmujemy następującą definicję:

DEFINICJA 2.26

Znaczeniem (semantyką) dowolnej formuły $\alpha$ w strukturze danych **A** będziemy nazywać funkcję całkowitą $\alpha_A$

$$\alpha_A \colon W(\mathbf{A}) \to \mathbf{B}_0$$

która dowolnemu wartościowaniu $v$ przyporządkowuje element dwuelementowej algebry Boole'a $\mathbf{B}_0$. Wartość funkcji $\alpha_A$ dla ustalonego argumentu $v$ nazywamy wartością formuły $\alpha$ przy wartościowaniu $v$ w strukturze **A** i definiujemy rekurencyjnie w następujący sposób:

$$q_A(v) = v(q) \quad \text{dla} \quad q \in V_0,$$

$$\varrho(\tau_1, \ldots, \tau_n)_A(v) = \begin{cases} \varrho_A(\tau_{1A}(v), \ldots, \tau_{nA}(v)), & \text{gdy } \tau_{1A}(v), \ldots, \tau_{nA}(v) \text{ są określone} \\ 0 & \text{w przeciwnym razie} \end{cases}$$

$$(\alpha \wedge \beta)_A(v) = \alpha_A(v) \cap \beta_A(v)$$
$$(\alpha \vee \beta)_A(v) = \alpha_A(v) \cup \beta_A(v)$$
$$(\alpha \Rightarrow \beta)_A(v) = -\alpha_A(v) \cup \beta_A(v)$$
$$(\neg\alpha)_A(v) = -\alpha_A(v)$$
$$((\forall x)\,\alpha)_A(v) = \mathbf{1} \quad \text{wtw dla każdego } a \in A,\ \alpha_A(v_a^x) = \mathbf{1}$$
$$((\exists x)\,\alpha)_A(v) = \mathbf{1} \quad \text{wtw istnieje } a \in A \text{ takie, że } \alpha_A(v_a^x) = \mathbf{1},$$
$$\text{gdzie} \quad v_a^x(z) = v(z) \quad \text{dla} \quad z \neq x \quad \text{i} \quad v_a^x(x) = a$$

przy czym $\tau_1, \ldots, \tau_n$ są dowolnymi termami, $\varrho$ jest $n$-argumentowym predykatem, a $\alpha$ i $\beta$ są dowolnymi formułami. ■

*Zapamiętajmy*

(1) Wartość formuły jest określona przy dowolnym wartościowaniu w każdej strukturze danych.

(2) Wartością formuły elementarnej, w której występuje term o wartości nieokreślonej dla danego wartościowania, jest **0** (fałsz).

(3) Kwantyfikator ogólny jest interpretowany jako kres dolny

$$((\forall x)\,\alpha)_A(v) = \mathit{inf}\,\{\alpha_A(v_a^x) : a \in A\}$$

a kwantyfikator szczegółowy jako kres górny w algebrze Boole'a

$$((\exists x)\,\alpha)_A(v) = \mathit{sup}\,\{\alpha_A(v_a^x) : a \in A\}$$

(4) Wartość formuły przy wartościowaniu $v$ w strukturze **A** zależy jedynie od wartości zmiennych wolnych występujących w tej formule.

UWAGA

Jeżeli rozważany język zawiera predykat $=$ oraz $\tau$ jest termem tego języka, to dla dowolnej struktury danych **A** i dla dowolnego wartościowania $v$

$$\mathbf{A}, v \models \tau = \tau \text{ wtedy i tylko wtedy, gdy } v \in Dom(\tau)$$

Inaczej mówiąc formuła $\tau = \tau$ przyjmuje wartość **1** tylko dla tych wartościowań, dla których wartości termu $\tau$ w strukturze $A$ są określone. ■

DEFINICJA 2.27

Powiemy, że formuła $\alpha$ jest spełniona przez wartościowanie $v$ w strukturze **A** wtedy i tylko wtedy, gdy $\alpha_A(v) = \mathbf{1}$, krótko $\mathbf{A}, v \models \alpha$. Jeżeli formuła $\alpha$ jest spełniona przez każde wartościowanie w strukturze danych **A**, to powiemy, że $\alpha$ jest prawdziwa w **A** i piszemy $\mathbf{A} \models \alpha$. Jeżeli formuła $\alpha$ jest prawdziwa w każdej strukturze, to powiemy, że jest tautologią, symbolicznie $\models \alpha$. ■

PRZYKŁAD 2.15

(1) Formuła $(\forall y)\, x \leqslant y$ jest spełniona w strukturze liczb naturalnych ze zwykłą interpretacją predykatu $\leqslant$ przez wartościowanie $v$, w którym $v(x) = 0$, i nie jest prawdziwa w tej strukturze.

(2) Formuła $(x \leqslant y \vee y \leqslant x)$ pewnego języka L jest prawdziwa w każdej strukturze danych dla tego języka, w której symbol $\leqslant$ jest interpretowany jako relacja liniowego porządku.

(3) Formuła postaci $((x:z) \leqslant y \vee \neg (x:z) \leqslant y)$ jest tautologią, tzn. jest prawdziwa niezależnie od wartościowania i interpretacji symbolu relacyjnego $\leqslant$ i funkcyjnego $:$. Gdy wartość termu $(x:z)$ nie jest określona, formuła elementarna $(x:z) \leqslant y$ ma wartość określoną — jest fałszywa. W konsekwencji formuła $((x:z) \leqslant y \vee \neg (x:z) \leqslant y)$ jest prawdziwa przy każdym wartościowaniu w dowolnej strukturze danych. □

**LEMAT 2.3**

Następujące formuły w języku pierwszego rzędu L są prawdziwe w każdej strukturze danych dla L. Nazywamy je prawami logiki klasycznej.

(1) $(\alpha \vee \neg\alpha)$ prawo wyłączonego środka

(2) $(\neg(\neg\alpha) \equiv \alpha)$ prawo podwójnego przeczenia

(3) $((\alpha \Rightarrow \beta) \Rightarrow (\neg\beta \Rightarrow \neg\alpha))$ prawo transpozycji

(4) $((\forall x)\,\neg\alpha = \neg(\exists x)\,\alpha)$ prawo de Morgana ■

**DEFINICJA 2.28**

Niech $Z$ będzie dowolnym zbiorem formuł w języku L. Mówimy, że struktura danych **A** jest modelem zbioru formuł $Z$, w skrócie $A \models Z$, jeżeli dla każdej formuły $\alpha$ ze zbioru $Z$, $\mathbf{A} \models \alpha$. ■

**PRZYKŁAD 2.16**

Rozważmy język pierwszego rzędu L, w którym występują zmienne indywiduowe dwóch typów $E$ i $S$, symbole funkcyjne *top, pop* i *push* typów $(S \to E)$, $(S \to S)$, $(E \times S \to S)$ oraz symbole relacyjne *empty*, = odpowiednio typów $(S)$ i $(S \times S)$.

Struktura stosów zdefiniowana w p. 2.3 jest modelem następującego zbioru formuł:

$$(\forall e)\,(\forall s) \quad \neg\, empty(push(e,s))$$

$$(\forall e)\,(\forall s) \quad e = top(push(e,s))$$

$$(\forall e)\,(\forall s) \quad s = pop(push(e,s))$$

$$(\forall e)\,(\forall s) \quad (\neg\, empty(s) \Rightarrow s = push(top(s), pop(s))).$$

□

**PRZYKŁAD 2.17**

Niech $\alpha(x)$ oznacza formułę w języku $L_{=}$ z jedną zmienną wolną $x$ i niech $\alpha(x/\tau)$ będzie formułą otrzymaną z $\alpha$ przez zastąpienie wszystkich wystąpień $x$ przez $\tau$. Pokażemy, że każda struktura danych dla języka L jest modelem zbioru formuł postaci

$$((\forall x)\,\alpha(x) \Rightarrow (\tau = \tau \Rightarrow \alpha(x/\tau))$$

gdzie $\tau$ jest dowolnym termem tego samego typu co zmienna $x$. Rozważmy dowolną strukturę danych **A** dla L. Niech $v$ będzie dowolnie ustalonym wartościowaniem w **A**, a $\tau$ dowolnie ustalonym termem zgodnego z $x$ typu. Przypuśćmy, że

$$\mathbf{A}, v \models (\forall x)\,\alpha(x) \quad \text{oraz} \quad \mathbf{A}, v \models \tau = \tau$$

Z drugiego warunku wynika, że wartość termu $\tau$ jest określona w **A** przy wartościowaniu $v$. Niech $a = \tau_A(v)$. Na mocy definicji semantyki oraz pierwszego warunku **A**, $v_a^x \models \alpha(x)$, czyli

$$\mathbf{A}, v \models \alpha(x/\tau).$$ □

# Problemy wyrażalności w języku pierwszego rzędu 2.5

DEFINICJA 2.29

Powiemy, że własność $w$ jest wyrażalna w języku L wtedy i tylko wtedy, gdy istnieje formuła $\alpha$ w języku L taka, że dla dowolnej struktury danych **A**

**A** ma własność $w$ wtw $\mathbf{A} \models \alpha$ ■

PRZYKŁAD 2.18

Własność „być grupą abelową” jest wyrażalna w języku $L_=$, w którym $\varphi$ jest funktorem dwuargumentowym, a zero jest stałą. Niech $\alpha$ oznacza koniunkcję następujących formuł:

$$(\forall x)\,(\exists y)\,\varphi(x,y) = 0$$
$$(\forall x,y)\,\varphi(x,y) = \varphi(y,x)$$
$$(\forall x,y)\,\varphi(x,\varphi(y,z)) = \varphi(\varphi(x,y),z)$$
$$(\forall x)\;\varphi(x,0) = x$$

**A** jest grupą abelową wtedy i tylko wtedy, gdy $\mathbf{A} \models \alpha$. □

PRZYKŁAD 2.19

Niech L będzie dowolnym językiem pierwszego rzędu, w którym $P$ jest zbiorem predykatów, a $\Phi$ zbiorem funktorów. Niech $eq \in P$. Rozważmy następujący zbiór formuł *Axeq* w języku L:

$$(\forall x)\; eq(x,x)$$
$$(\forall x,y)(eq(x,y) \Rightarrow eq(y,x))$$
$$(\forall x,y,z)\;((eq(x,y) \wedge eq(y,z)) \Rightarrow eq(x,z))$$
$$(\forall \mathbf{x},\mathbf{y})(eq(\varphi(\mathbf{x}),\varphi(\mathbf{x})) \wedge eq(\varphi(\mathbf{y}),\varphi(\mathbf{y})) \wedge eq(x_1,y_1) \wedge \dots \wedge eq(x_n,y_n)) \Rightarrow$$
$$\Rightarrow eq(\varphi(\mathbf{x}),\varphi(\mathbf{y})))$$
$$(\forall \mathbf{x},\mathbf{y})((eq(x_1,y_1) \wedge \dots \wedge eq(x_n,y_n)) \Rightarrow \varrho(\mathbf{x}) = \varrho(\mathbf{y}))$$

dla dowolnego $n$-argumentowego funktora $\varphi \in \Phi$ i dowolnego $n$-argumentowego predykatu $\varrho \in$ P. Przez **x** oznaczyliśmy wektor $(x_1, \dots, x_n)$, a przez **y** wektor $(y_1, \dots, y_n)$.

W dowolnej strukturze danych **A** dla języka L zachodzi następująca własność:

$\mathbf{A} \models Axeq$ wtw $eq_{\mathbf{A}}$ jest kongruencją w **A**

Pierwsze trzy warunki gwarantują, że $eq_{\mathbf{A}}$ jest relacją równoważności w **A**, a pozostałe wyrażają własność ekstensjonalności relacji $eq_{\mathbf{A}}$.

W dalszym ciągu zbiór formuł *Axeq* będziemy nazywać aksjomatami równości. □

PRZYKŁAD 2.20

(1) Oznaczmy przez $\alpha$ formułę postaci

$$(\exists x_1)\ldots(\exists x_n)(\forall x)(x = x_1 \vee \ldots \vee x = x_n)$$

Formuła $\alpha$ ma następującą własność: dla dowolnej struktury danych **A** dla języka $L_=$, $\mathbf{A} \models \alpha$ wtedy i tylko wtedy, gdy uniwersum systemu relacyjnego **A** zawiera co najwyżej $n$ elementów. Rzeczywiście, rozważmy system **A**, którego uniwersum **A** ma moc większą niż $n$. Niech $v$ będzie dowolnym wartościowaniem w **A** i niech $a$ będzie takim elementem zbioru $A$, że $v(x_i) \neq a$ dla $1 \leqslant i \leqslant n$. Wtedy

$$\mathbf{A}, v_a^x \models (x \neq x_1 \wedge \ldots \wedge x \neq x_n)$$

i w konsekwencji

$$non\, \mathbf{A}, v \models (\forall x)(x = x_1 \vee \ldots \vee x = x_n)$$

Ponieważ $v$ było dowolnym wartościowaniem, to

$$non\, \mathbf{A} \models (\exists x_1)\ldots(\exists x_n)(\forall x)(x = x_1 \vee \ldots \vee x = x_n)$$

czyli $non\, \mathbf{A} \models \alpha$.

Odwrotnie, jeżeli $non\, \mathbf{A} \models \alpha$, to (ponieważ $\alpha$ jest formułą zamkniętą) dla dowolnego wartościowania $v$, $non\, \mathbf{A}, v \models \alpha$.
Stąd

$$non\, \mathbf{A}, v \models (\forall x)(x = x_1 \vee \ldots \vee x = x_n) \text{ dla dowolnego } v.$$

Zgodnie z przyjętym znaczeniem kwantyfikatorów, dla każdego $v$ istnieje taki element $a$, że $non\, \mathbf{A}$, $v_a^x \models (x = x_1 \vee \ldots \vee x = x_n)$. Ostatecznie, dla dowolnych wartości $v(x_1), \ldots, v(x_n)$ istnieje taki element $a$, że $a \neq v(x_i)$, dla $i \leqslant n$, tzn. $card(A) > n$.

(2) Rozważmy formułę $\beta$ języka L postaci

$$(\exists x_1)\ldots(\exists x_n)(x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \ldots \wedge x_1 \neq x_n \wedge x_2 \neq x_3 \wedge x_2 \neq x_4 \\ \wedge \ldots \wedge x_2 \neq x_n \wedge \ldots \wedge x_{n-1} \neq x_n)$$

Formuła $\beta$ jest prawdziwa tylko w tych strukturach dla języka L, których

uniwersum ma co najmniej $n$ elementów. Formuła $\beta$ wyraża więc własność „uniwersum systemu ma co najmniej $n$ elementów". Dowód tego faktu pozostawiamy czytelnikowi.

Podsumowując oba wymienione przykłady zauważmy, że formuła $(\alpha \wedge \beta)$ wyraża własność „być zbiorem $n$ elementowym", gdyż jest prawdziwa tylko w tych strukturach, które jednocześnie mają własność $\alpha$ i własność $\beta$. □

TWIERDZENIE 2.1

Niech **A** i **B** będą strukturami danych dla języka L. Jeżeli $h: \mathbf{A} \to \mathbf{B}$ jest izomorfizmem odwzorowującym strukturę **A** na strukturę **B**, to dla dowolnego termu $\tau$ i dla dowolnego wartościowania $v$ w **A**, $\tau_{\mathbf{A}}$ jest określone przy wartościowaniu $v$ wtedy i tylko wtedy, gdy $\tau_{\mathbf{B}}$ jest określone przy wartościowaniu $h \circ v$ oraz dla określonych $\tau_{\mathbf{A}}(v)$, $\tau_{\mathbf{B}}(h \circ v)$.

$$h(\tau_{\mathbf{A}}(v)) = \tau_{\mathbf{B}}(h \circ v) \tag{2.1}$$

Co więcej, dla dowolnej formuły $\alpha$ i dowolnego wartościowania $v$

$$\mathbf{A}, v \models \alpha \equiv \mathbf{B}, (h \circ v) \models \alpha \tag{2.2}$$

DOWÓD

Dowód twierdzenia przebiega przez indukcję ze względu na stopień komplikacji termu i formuły. Jeżeli $\tau$ jest zmienną indywiduową $x$, to

$$h(\tau_{\mathbf{A}}(v)) = h(v(x)) = (h \circ v)(x) = x_{\mathbf{B}}(h \circ v) = \tau_{\mathbf{B}}(h \circ v)$$

Jeżeli $\tau$ jest postaci $\varphi(\tau_1, \ldots, \tau_n)$ oraz własność (2.1) jest spełniona dla termów $\tau_1, \ldots, \tau_n$, to z definicji izomorfizmu struktur i na mocy założenia indukcyjnego

$$h(\varphi(\tau_1, \ldots, \tau_n)_{\mathbf{A}}(v)) = h(\varphi_{\mathbf{A}}(\tau_{1\mathbf{A}}(v), \ldots, \tau_{n\mathbf{A}}(v))) =$$
$$= \varphi_{\mathbf{B}}(h(\tau_{1\mathbf{A}}(v)), \ldots, h(\tau_{n\mathbf{A}}(v))) = \varphi_{\mathbf{B}}(\tau_{1\mathbf{B}}(h \circ v), \ldots, \tau_{n\mathbf{B}}(h \circ v)) =$$
$$= \varphi(\tau_1, \ldots, \tau_n)_{\mathbf{B}}(h \circ v)$$

Zatem własność (2.1) została udowodniona dla wszystkich termów języka L.

Rozważmy formuły. Niech $\alpha$ będzie formułą postaci $\varrho(\tau_1, \ldots, \tau_m)$.

$\mathbf{A}, v \models \alpha$ wtw $\mathbf{A}, v \models \varrho(\tau_1, \ldots, \tau_m)$ wtw $\tau_{i\mathbf{A}}$ określone dla $i \leqslant m$ oraz $(\tau_{1\mathbf{A}}(v) \ldots, \tau_{m\mathbf{A}}(v)) \in \varrho_{\mathbf{A}}$

Stąd i z definicji izomorfizmu

$\mathbf{A}, v \models \alpha$ wtw $(h(\tau_{1\mathbf{A}}(v)), \ldots, h(\tau_{m\mathbf{A}}(v))) \in \varrho_{\mathbf{B}}$ wtw $(\tau_{1\mathbf{B}}(h \circ v), \ldots, \tau_{m\mathbf{B}}(h \circ v)) \in \varrho_{\mathbf{B}}$ wtw $\mathbf{B}, h \circ v \models \varrho(\tau_1, \ldots, \tau_m)$

Załóżmy, że własność (2.2) jest prawdziwa dla formuł $\beta$, $\beta_1$, $\beta_2$ i rozważmy formułę postaci $(\beta_1 \wedge \beta_2)$. Mamy wtedy

$$\mathbf{A}, v \models (\beta_1 \wedge \beta_2) \quad \text{wtw} \quad \mathbf{A}, v \models \beta_1 \quad \text{i} \quad \mathbf{A}, v \models \beta_2 \quad \text{wtw}$$
$$\mathbf{B}, h \circ v \models \beta_1 \quad \text{i} \quad \mathbf{B}, h \circ v \models \beta_2 \quad \text{wtw} \quad \mathbf{B}, h \circ v \models (\beta_1 \wedge \beta_2)$$

Niech $\alpha$ będzie formułą postaci $(\exists x)\,\beta(x)$. Jeżeli $a$ jest zmienną typu zgodnego z $x$, to $b = h(a)$ też ma typ zgodny z $x$, a ponadto

$$\mathbf{A}, v \models (\exists x)\,\beta(x) \quad \text{wtw}$$
$$\text{dla pewnego } a, \ \mathbf{A}, v_a^x \models \beta(x) \quad \text{wtw}$$
$$\text{dla pewnego } a \in \mathbf{A}, \quad \mathbf{B}, h \circ v_a^x \models \beta(x) \quad \text{wtw}$$
$$\text{dla pewnego } a \in \mathbf{A}, \quad \mathbf{B}, (h \circ v)_{h(a)}^x \models \beta(x) \quad \text{wtw}$$
$$\text{dla pewnego } b \in \mathbf{B}, \quad \mathbf{B}, (h \circ v)_b^x = \beta(x) \quad \text{wtw}$$
$$\mathbf{B}, h \circ v \models (\exists x)\,\beta(x)$$

Dowód własności (2.2) w przypadku innych postaci formuł przebiega analogicznie. □

Na mocy twierdzenia 2.1, jeżeli $\mathbf{A} \models \alpha$ oraz $\mathbf{B}$ jest strukturą izomorficzną z $\mathbf{A}$, to $\mathbf{B} \models \alpha$. Inaczej mówiąc, jeżeli struktura $\mathbf{A}$ jest modelem pewnego zbioru formuł $Z$, to każda struktura izomorficzna z $\mathbf{A}$ też jest modelem zbioru $Z$. Zatem, nie jest możliwa jednoznaczna charakteryzacja jakiejkolwiek struktury za pomocą formuły. Co najwyżej możemy poszukiwać charakteryzacji klas struktur izomorficznych. Niestety i to zdanie nie zawsze może być uwieńczone sukcesem.

PRZYKŁAD 2.21

Rozważmy strukturę liczb naturalnych

$$\mathbf{N} = \langle N, s, 0; = \rangle$$

Własność „być liczbą naturalną” czy też „być strukturą liczb naturalnych” nie jest wyrażalna w języku pierwszego rzędu z równością. Oczywiście możemy scharakteryzować stałą 0 i funkcję następnika $s$ za pomocą następującej formuły $\alpha$:

$$(\forall x)(s(x) = s(x) \wedge \neg s(x) = 0) \wedge (\forall x, y)(s(x) = s(y) \Rightarrow x = y)$$

Jednakże z tego, że $\mathbf{N} \models \alpha$ nie wynika, że uniwersum struktury $\mathbf{N}$ składa się z liczb naturalnych lub że $\mathbf{N}$ jest izomorficzne ze strukturą liczb naturalnych. Własności, które gwarantuje nam prawdziwość formuły $\alpha$ w strukturze $\mathbf{N}$ są następujące:

(1) interpretacja funktora $s$ jest funkcją całkowitą;
(2) funkcja ta nie przyjmuje wartości 0;
(3) interpretacja funktora $s$ jest funkcją różnowartościową.

Okazuje się, że dodanie do formuły $\alpha$, tzw. schematu indukcji

$$\big(\gamma(x/0) \wedge (\forall x)\big(\gamma(x) \Rightarrow \gamma(x/s(x))\big)\big) \Rightarrow (\forall x)\ \gamma(x)$$

nadal nie prowadzi do charakteryzacji zbioru liczb naturalnych. Aksjomaty podane poprzednio, aksjomaty Peano, nie pozwalają odrzucić modeli, które oprócz liczb naturalnych zawierają tzw. elementy niestandardowe. Okazuje się, że zbiór przedstawionych wyżej aksjomatów ma modele nieprzeliczalne, są to tzw. niestandardowe modele arytmetyki [21]. □

Niestety, większość interesujących nas własności nie daje się wyrazić za pomocą formuł pierwszego rzędu. Do takich własności należą m.in.:

(1) być zbiorem skończonym;
(2) być relacją dobrego porządku;
(3) być liczbą naturalną;
(4) być stosem;
(5) być kolejką;
(6) być drzewem binarnym, skończonym.

Aby zakończyć ten rozdział pozytywnym akcentem, rozważmy sprawę częściowości funkcji będących interpretacjami funktorów języka. Zgodnie z przyjętą semantyką, niektóre termy mogą mieć nieokreślone wartości w pewnych strukturach danych i przy pewnych wartościowaniach. W konsekwencji formuły elementarne, w których wystąpił taki term mają, zgodnie z przyjętą semantyką, wartość fałsz. Dla dalszych rozważań istotne będzie wykrywanie i definiowanie tych „nienormalnych" sytuacji w obliczaniu wartości termu czy formuły otwartej (tzn. formuły nie zawierającej kwantyfikatorów). Przedstawimy rekurencyjną definicję własności, która wyraża określoność termów i formuł otwartych pierwszego rzędu.

DEFINICJA 2.30

Dla dowolnych termów $\tau, \tau_1, \ldots, \tau_n$ i dowolnych formuł otwartych $\alpha, \beta$ języka pierwszego rzędu $L_=$ niech

$$ok(\tau) \stackrel{df}{=} (\tau = \tau)$$

$$ok(\varrho(\tau_1, \ldots, \tau_n)) \stackrel{df}{=} (ok(\tau_1) \wedge \ldots \wedge ok(\tau_n))$$

$$ok(\alpha \wedge \beta) \stackrel{df}{=} ok(\alpha \vee \beta) \stackrel{df}{=} ok(\alpha \Rightarrow \beta) \stackrel{df}{=} (ok(\alpha) \wedge ok(\beta))$$

$$ok(\neg\alpha) \stackrel{df}{=} ok(\alpha)$$

LEMAT 2.4

Dla dowolnego termu lub formuły otwartej $w$, dla dowolnej struktury danych $\mathbf{A}$ i dowolnego wartościowania $v$.

**A**, $v \vdash ok(w)$ wtedy i tylko wtedy, gdy dla każdego termu postaci $\varphi(\tau_1, ..., \tau_n)$ występującego w wyrażeniu *w*, funkcje częściowe $\tau_{1A}, ..., \tau_{nA}$ mają określone wartości przy wartościowaniu *v* oraz $(\tau_{1A}, ..., \tau_{nA}) \in Dom(\varphi)$.

Oczywisty dowód lematu pomijamy. ■

UWAGA:

W pewnych przypadkach (np. p. 3.3) termin *ok* (*w*) będzie użyty jako skrót zastępujący zdanie: „wartość wyrażenia *w* jest określona". □

# Deterministyczne programy iteracyjne 3

## Wprowadzenie 3.1

W tym rozdziale omówimy najprostszą klasę programów $\pi$, zwaną klasą programów iteracyjnych lub klasą deterministycznych **while**-programów. Klasa ta jest dostatecznie bogata, by można było z jej pomocą zdefiniować każdą funkcję obliczalną [21]. Na przykładzie klasy $\pi$ pokażemy, jak pewne metody logiki mogą być użyte do analizy programów i badania struktur danych.

Podstawą naszych rozważań będzie język rachunku kwantyfikatorów z równością, tzn. język pierwszego rzędu (por. p. 2.3). Jednak język taki jest zbyt ubogi, by mówić o bardziej skomplikowanych własnościach programów, np.: własność stopu nie jest w nim wyrażalna. W konsekwencji rozszerzymy język pierwszego rzędu dopuszczając wyrażenia, w których oprócz zwykłych formuł klasycznego rachunku logicznego będą występować formuły algorytmiczne. Formuły algorytmiczne pozwolą wyrazić własność stopu i wiele innych własności programów i struktur danych niewyrażalnych w języku pierwszego rzędu.

## Język programów 3.2

Rozwojowi maszyn cyfrowych towarzyszył od początku szybki rozwój języków programowania. Jedne języki mają charakter uniwersalny, inne służą wyspecjalizowanym zadaniom. Często różnią się znacznie nie tylko pod względem formalnym, ale także zestawem oferowanych instrukcji. Co więcej, ten proces trwa nadal. Pojawiają się nowe języki wyposażone w nowe konstrukcje mające ułatwić proces programowania, np. procesy równoległe, klasy czy współprogramy. W tej mnogości języków można wyodrębnić pewne wspólne cechy. I tak, w większości języków programowania program jest rozumiany jako skończony ciąg instrukcji. Łącząc instrukcje za pomocą pewnych dopuszczalnych konstrukcji buduje się instrukcje bardziej skomplikowane.

W tym rozdziale przedstawiamy klasę deterministycznych programów iteracyjnych, jednak definicja ta będzie abstrahować od pewnych szczegółów, takich jak np. deklaracje zmiennych, które nie odgrywają istotnej roli w analizie programów z tej klasy.

Niech L będzie ustalonym językiem pierwszego rzędu. Termy (por. definicję 2.21) i formuły (por. definicję 2.22) tego języka będą służyć jako wyrażenia arytmetyczne i wyrażenia booleowskie w programach zdefiniowanych w tym rozdziale.

DEFINICJA 3.1

Instrukcją przypisania będziemy nazywać dowolny napis postaci

$x := \tau$ lub $q := \gamma$

gdzie $x$ jest zmienną indywiduową, $\tau$ jest termem tego samego typu co zmienna $x$, $q$ jest zmienną zdaniową, a $\gamma$ jest formułą otwartą. ■

PRZYKŁAD 3.1

Jeżeli w języku L jednosortowym występują zmienne $x$, $y$, symbole operacji dwuargumentowych $+$, $-$ oraz symbole relacji dwuargumentowych $\geqslant$, $\leqslant$ to

$z := (x+y)-z$ oraz $q := (x \leqslant y \Rightarrow x \geqslant (y-z))$

są instrukcjami przypisania. □

Instrukcje przypisania będziemy nazywać *programami atomowymi* lub *elementarnymi.*

DEFINICJA 3.2

Zbiór $\pi$ programów deterministycznych iteracyjnych w języku L jest to najmniejszy zbiór wyrażeń spełniających następujące warunki:

(1) każda instrukcja przypisania w języku L jest programem klasy $\pi$;

(2) jeżeli $\gamma$ jest formułą otwartą oraz K i M są programami należącymi do $\pi$, to wyrażenie **if** $\gamma$ **then** K **else** M **fi** jest programem klasy $\pi$;

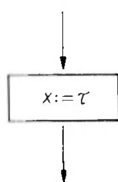
(3) jeżeli $\gamma$ jest formułą otwartą oraz M jest programem klasy $\pi$, to wyrażenie **while** $\gamma$ **do** M **od** jest programem klasy $\pi$;

(4) jeżeli K, M są programami klasy $\pi$, to wyrażenie **begin** K; M **end** jest programem klasy $\pi$. ■

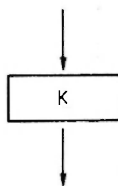
Przyjęto ilustrować programy w postaci tzw. grafów przepływu, diagramów. Każdy diagram jest grafem z jednym wejściem i jednym wyjściem, wierzchołkami grafu są testy (formuły otwarte) lub instrukcje przypisania.

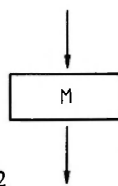
Diagram przedstawiający najprostszy program atomowy ma postać jak na rys. 3.1.

Jeżeli dane są diagramy odpowiadające programom K i M (rys. 3.2), to identyfikując krawędź wyjściową pierwszego programu z krawędzią wejściową drugiego otrzymamy diagram programu złożonego **begin** K; M **end** (rys. 3.3).

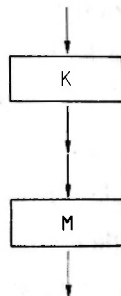


Rys. 3.1





Rys. 3.2



Rys. 3.3 begin K; M end

Analogicznie, jeżeli diagramy programów K i M są takie, jak przedstawiono na rys. 3.2, to grafy z rys. 3.4 i 3.5 są odpowiednio diagramami programów

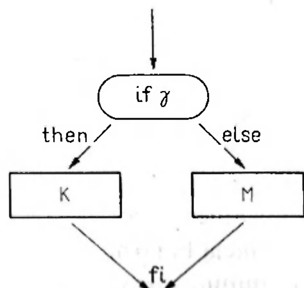
**if** $\gamma$ **then** K **else** M **fi** i **while** $\gamma$ **do** K **od**

Zauważmy, że zbiór programów $\pi$ tworzy algebrę z dwoma dwuargumentowymi operacjami składania $\circ$ i rozgałęzienia $\mathbf{v}_\gamma$ oraz jednoargumentową operacją iteracji $*_\gamma$ dla $\gamma \in F_0$, określonymi następująco:

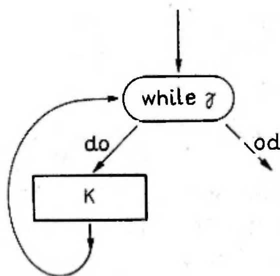
K $\circ$ M $\stackrel{df}{=}$ **begin** K; M **end**

K$\mathbf{v}_\gamma$M $\stackrel{df}{=}$ **if** $\gamma$ **then** K **else** M **fi**

$*_\gamma$M $\stackrel{df}{=}$ **while** $\gamma$ **do** M **od**



if γ then K else M fi Rys. 3.4



while γ do K od Rys. 3.5

Zbiorem generatorów tej algebry jest zbiór wszystkich instrukcji przypisania, tzn. każdy program można otrzymać z instrukcji przypisania stosując (być może wielokrotnie) operacje składania, rozgałęziania i iteracji.

Algebraiczny charakter zbioru $\pi$ podkreśla modularną strukturę rozważanej klasy programów. Operacje składania, rozgałęziania i iteracji są przykładami operacji programotwórczych.

W dalszym ciągu będziemy stosować następujące uproszczenia:

(1) zamiast $i$-krotnego złożenia programów

$$\underbrace{\textbf{begin } K; \textbf{ begin } K; \ldots K \textbf{ end end}}_{i \text{ razy}}$$

będziemy pisać krótko $K^i$;

(2) dla dowolnej zmiennej $x$, zamiast instrukcji

**if** $\gamma$ **then** K **else** $x := x$ **fi**

będziemy pisać krótko **if** $\gamma$ **then** K **fi**.

# Semantyka 3.3

Zadanie określenia semantyki języka programowania sprowadza się do podania interpretacji wszystkich symboli występujących w rozważanym języku. Po pierwsze, musimy wybrać zbiór, którego elementy będą reprezentowały wartości zmiennych, a po drugie, musimy przypisać symbolom funkcyjnym i relacyjnym odpowiednie operacje i relacje w wybranym zbiorze. Oznacza to, że musimy najpierw ustalić system relacyjny odpowiadający językowi pierwszego rzędu lub inaczej, strukturę danych dla języka, nad którym zbudowano język programowania. Wszystkie termy i formuły pojawiające się w programach będą miały przypisane znaczenie zgodnie z definicjami przyjętymi w p. 2.3. Pozostała jeszcze zasadnicza sprawa przypisania znaczenia konstrukcjom programotwórczym. Korzystając z modularnej struktury programów podamy interpretację programów atomowych i określimy sposób tworzenia interpretacji konstrukcji bardziej złożonych z interpretacji programów prostszych.

Chociaż nie ma jednoznacznej opinii, jak rozumieć poszczególne konstrukcje programotwórcze, to jest powszechnie przyjęte, by wiązać z programem pewną relację binarną określającą związek między danymi (tzn. stanem pamięci przed wykonaniem programu) a wynikami (tzn. stanem pamięci po wykonaniu programu). Relację taką nazywa się relacją wejścia-wyjścia. Ponieważ stan pamięci będzie dla nas wartościowaniem (por. z definicją 2.24), zatem relacja przypisana programowi będzie relacją binarną w zbiorze wszystkich wartościowań. Będziemy używali terminu *wartościowanie początkowe* do określenia stanu pamięci przed wykonaniem pro-

gramu i *wartościowanie końcowe* do określenia stanu pamięci po wykonaniu programu.

Przedstawiony dalej sposób wyznaczania relacji wejścia-wyjścia dla danego programu klasyfikuje się jako przykład semantyki operacyjnej.

Niech M będzie programem, a **A** strukturą danych języka L. Jeżeli wartościowania $v$, $v'$ należą do relacji wejścia-wyjścia wyznaczonej przez program M w strukturze **A**, to fakt ten zapiszemy w postaci

$$v \xrightarrow{M_A} v'$$

ewentualnie pomijając indeks **A**, jeśli struktura danych jest ustalona.

Relację wejścia-wyjścia definiujemy rekurencyjnie następująco:

$$v \xrightarrow{x:=\omega} v_1 \quad \text{wtw} \quad \text{wartość } \omega_A(v) \text{ jest określona, } v_1(x) = \omega_A(v)$$

$$\text{i} \quad v_1(z) = v(z) \quad \text{dla} \quad z \neq x$$

$$v \xrightarrow{\text{begin K; M end}_A} v_1 \quad \text{wtw } (\exists v')\, v \xrightarrow{K_A} v' \quad \text{i} \quad v' \xrightarrow{M_A} v_1$$

$$v \xrightarrow{\text{if } \gamma \text{ then K else M fi}_A} v_1 \quad \text{wtw } \mathbf{A}, v \models (\gamma \wedge ok(\gamma)) \quad \text{i} \quad v \xrightarrow{K_A} v_1$$

$$\text{lub} \quad \mathbf{A}, v \models (\neg\gamma \wedge ok(\gamma)) \quad \text{i} \quad v \xrightarrow{M_A} v_1$$

$$v \xrightarrow{\text{while } \gamma \text{ do M od}_A} v_1 \quad \text{wtw } \mathbf{A}, v \models (\neg\gamma \wedge ok(\gamma)) \quad \text{i} \quad v_1 = v$$

$$\text{lub } \mathbf{A}, v \models (\gamma \wedge ok(\gamma)) \quad \text{i} \quad (\exists v')\, v \xrightarrow{M_A} v' \quad \text{i} \quad v' \xrightarrow{\text{while } \gamma \text{ do M od}_A} v_1$$

dla dowolnych $v$, $v_1$ w strukturze danych **A**.

Głębszy wgląd w metodę wyznaczania znaczenia programu w strukturze danych pozwala na zrozumienie procesu, w którym stan początkowy pamięci, jest przekształcany w wynik. Proces ten będziemy nazywać *obliczeniem programu*. Poniżej przedstawimy formalne pojęcie obliczenia oraz dwa pojęcia pomocnicze: konfiguracji i relacji bezpośredniego następstwa.

DEFINICJA 3.3

Konfiguracją (lub stanem obliczenia) w ustalonej strukturze danych **A** będziemy nazywać uporządkowaną parę $\langle v, K_1; \ldots; K_n \rangle$ gdzie $v$ jest wartościowaniem w **A**, a $K_1, \ldots, K_n$ jest skończoną listą programów, które będą kolejno wykonywane. (Dla uproszczenia nazwa struktury nie występuje w zapisie konfiguracji). ■

W zbiorze wszystkich konfiguracji w ustalonej strukturze **A** określimy relację bezpośredniego następstwa. Za jej pomocą zdefiniujemy pojęcie obliczenia.

DEFINICJA 3.4

Definicją bezpośredniego następstwa nazywamy relację binarną $\underset{A}{\rightarrow}$(znak będziemy pomijać, jeżeli nie będzie prowadziło to do nieporozumień) określoną w zbiorze wszystkich konfiguracji w strukturze **A** w sposób następujący:

$\langle v, z := \omega; K_1;...;K_n\rangle \underset{A}{\rightarrow} \langle v', K_1;...;K_n\rangle$ jeżeli $\mathbf{A}, v \models ok(\omega)$ oraz $v'(x) = v(x)$ dla $x \neq z$ i $v'(z) = \omega_A(v)$

$\langle v, \textbf{if}\ \gamma\ \textbf{then}\ K\ \textbf{else}\ M\ \textbf{fi};\ K_1;...\ K_n\rangle \underset{A}{\rightarrow} \langle v, K, K_1;...;K_n\rangle$ jeżeli $\mathbf{A}, v \models (\gamma \wedge ok(\gamma))$

$\langle v, \textbf{if}\ \gamma\ \textbf{then}\ K\ \textbf{else}\ M\ \textbf{fi};\ K_1;...\ K_n\rangle \underset{A}{\rightarrow} \langle v, M, M_1;...;M_n\rangle$ jeżeli $\mathbf{A}, v \models (\neg\gamma \wedge ok(\gamma))$

$\langle v, \textbf{begin}\ K;\ M\ \textbf{end};\ K_1;...;K_n\rangle \underset{A}{\rightarrow} \langle v, K; M; K_1;...;K_n\rangle$

$\langle v, \textbf{while}\ \gamma\ \textbf{do}\ M\ \textbf{od};\ K_1;...;K_n\rangle \underset{A}{\rightarrow} \langle v, K_1;...;K_n\rangle$ jeżeli $\mathbf{A}, v \models (\neg\gamma \wedge ok(\gamma))$

$\langle v, \textbf{while}\ \gamma\ \textbf{do}\ M\ \textbf{od};\ K_1;...;K_n\rangle \underset{A}{\rightarrow} \langle v, M; \textbf{while}\ \gamma\ \textbf{do}\ M\ \textbf{od};\ K_1;...;K_n\rangle$ gdy $\mathbf{A}, v \models (\gamma \wedge ok(\gamma))$

■

Zauważmy, że dla danej konfiguracji istnieje co najwyżej jedna konfiguracja będąca jej bezpośrednim następnikiem. Co więcej, pewne konfiguracje nie są w relacji bezpośredniego następstwa z żadną inną konfiguracją. Należą do nich na przykład konfiguracje postaci $\langle v, \rangle$ oraz $\langle v, x := \gamma; M\rangle$, gdy $v$ nie należy do $Dom(\gamma)$.

DEFINICJA 3.5

Obliczeniem programu M w strukturze danych **A** przy wartościowaniu początkowym $v$ będziemy nazywać ciąg konfiguracji $\theta$ taki, że

(1) pierwszym elementem ciągu jest konfiguracja $\theta_0 = \langle v, M\rangle$

(2) jeżeli jest określony $i$-ty element ciągu $\theta_i$ oraz

(a) istnieje konfiguracja *Konf* taka, że $\theta_i \underset{A}{\rightarrow} Konf$, to jest określony $i+1$ element ciągu $\theta$ oraz $\theta_{i+1} \stackrel{df}{=} Konf$,

(b) w przeciwnym razie element $\theta_{i+1}$ nie jest określony i $\theta_i$ jest ostatnim elementem ciągu. ■

PRZYKŁAD 3.2

Rozważmy program K postaci

**begin**
  $y := 0$;
  **while** $\neg y = z$ **do** M **od**
**end**

oraz M = **begin** $y := y+1;\ x := 1+(1/(1+x))$ **end** w języku L. Niech $\langle R, 0, 1, +, / \rangle$ będzie strukturą danych dla jednosortowego języka L taką, że $R$ jest zbiorem liczb rzeczywistych, a 0, 1, +, / są naturalnymi operacjami w tym zbiorze.

(1) Dla danych początkowych $v$ takich, że $v(x), v(z) \in N$ program K ma obliczenia skończone. Jeżeli $v(x) = 1$, $v(z) = 2$, to obliczenie programu K wygląda następująco (wartość zmiennej $y$ w wartościowaniu $v$ pomijamy, gdyż jest nieistotna w przedstawionym obliczeniu):

$$\left\langle \frac{x\ y\ z}{1\ -\ 2}, \mathrm{K} \right\rangle$$

$$\left\langle \frac{x\ y\ z}{1\ -\ 2}, y := 0; \textbf{ while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ y\ z}{1\ 0\ 2}, \textbf{ while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ y\ z}{1\ 0\ 2}, \mathrm{M}; \textbf{ while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ y\ z}{1\ 0\ 2}, y := y+1; x := 1+(1/(1+x)); \textbf{ while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ y\ z}{1\ 1\ 2}, x := 1+(1/(1+x)); \textbf{ while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ \ y\ z}{3/2\ 1\ 2}, \textbf{ while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ \ y\ z}{3/2\ 1\ 2}, \mathrm{M}; \textbf{ while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ \ y\ z}{3/2\ 1\ 2}, y := y+1; x := 1+(1/(1+x)); \textbf{ while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ \ y\ z}{3/2\ 2\ 2}, x := 1+(1/(1+x)); \textbf{ while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ \ y\ z}{7/5\ 2\ 2}, \textbf{ while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ \ y\ z}{7/5\ 2\ 2}, \right\rangle$$

(2) Rozważmy wartościowanie początkowe $v$ takie, że $v(x) \in N$, $v(z) < 0$, np. $v(x) = 0$, $v(z) = -2$. Program K ma wtedy obliczenie nieskończone. Będzie tak, ponieważ w każdym kroku obliczenia wartość zmiennej $y$ rośnie, a wartość zmiennej $z$ nie ulega zmianie. Zatem nigdy nie zajdzie warunek $y = z$. Oznacza to, że pętla **while** będzie wykonywana nieskończenie wiele razy. Inaczej mówiąc, obliczenie programu jest nieskończone.

(3) Niech $v$ będzie wartościowaniem takim, że $v(x) = -1, v(z) = 2$. Przy takim wartościowaniu początkowym program K ma obliczenie skończone nieudane.

$$\left\langle \frac{x\ y\ z}{-1-2}, \mathrm{K} \right\rangle$$

$$\left\langle \frac{x\ y\ z}{-1-2}, y := 0; \textbf{while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ y\ z}{-1\ 0\ 2}, \textbf{while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ y\ z}{-1\ 0\ 2}, \mathrm{M}; \textbf{while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ y\ z}{-1\ 0\ 2}, y := y+1;\ x := 1+(1/(1+x)); \textbf{while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

$$\left\langle \frac{x\ y\ z}{-1\ 1\ 2}, x := 1+(1/(1+x)); \textbf{while } \neg y = z \textbf{ do } \mathrm{M} \textbf{ od} \right\rangle$$

Obliczenie urywa się nie dając wyniku, ponieważ aktualna instrukcja wymaga dzielenia przez zero. □

Jeżeli program M ma obliczenie $\theta$ skończone w strukturze **A** przy wartościowaniu $v$ i konfiguracja końcowa ma postać $\langle v', \emptyset \rangle$, tzn. lista programów do wykonania jest pusta, to wartościowanie $v'$ nazywamy wynikiem programu M dla danych $v$.

Jeżeli obliczenie $\theta$ jest ciągiem nieskończonym rozpoczynającym się od pary $\langle v, M \rangle$, to powiemy, że $\theta$ jest nieskończonym obliczeniem programu M dla danych $v$.

Jeżeli obliczenie $\theta$ jest ciągiem skończonym, ale lista instrukcji do wykonania w ostatniej konfiguracji nie jest pusta, to $\theta$ jest *obliczeniem nieudanym*. Mamy więc do czynienia z obliczeniem nieudanym wtedy i tylko wtedy, gdy w trakcie obliczenia pojawi się (w termie lub w formule) funkcja częściowa, której wartość, dla aktualnego stanu zmiennych, nie jest określona.

Niech $\mathrm{M}_{\mathbf{A}}$ oznacza relację w zbiorze wartościowań taką, że $(v, v') \in \mathrm{M}_{\mathbf{A}}$ wtedy i tylko wtedy, gdy istnieje skończone obliczenie $\theta$ programu M w strukturze **A** przy wartościowaniu początkowym $v$, którego wynikiem jest $v'$. Poniżej przedstawimy własności tej relacji.

UWAGA

Dla dowolnej struktury **A**, dla dowolnego wartościowania $v$ i dla dowolnego programu M

$$(v, v') \in \mathrm{M_A} \quad \text{wtw} \quad v \xrightarrow{\mathrm{M}} v'$$

Oczywisty dowód pomijamy. ■

LEMAT 3.1

Dla dowolnego programu M, dowolnej struktury **A**, $\mathrm{M_A}$ jest funkcją częściową oraz

(1) jeżeli M jest postaci $x := \omega$, to $Dom(\mathrm{M_A}) = \{v : \mathbf{A}, v \models ok(\omega)\}$ i dla $v \in Dom(\mathrm{M_A})$ mamy

$\mathrm{M_A}(v) = v'$, gdzie $v'(x) = \omega_\mathbf{A}(v)$, $v'(z) = v(z)$ dla $z \neq x$;

(2) jeżeli M jest postaci **begin** $\mathrm{K_1}$; $\mathrm{K_2}$ **end**, to

$$Dom(\mathrm{M_A}) = \{v : v \in Dom(\mathrm{K_{1A}}) \ \text{ i } \ \mathrm{K_{1A}}(v) \in Dom(\mathrm{K_{2A}})\}$$

oraz

$$\mathrm{M_A}(v) = \mathrm{K_{2A}}(\mathrm{K_{1A}}(v)) \quad \text{dla} \quad v \in Dom(\mathrm{M_A})$$

(3) jeżeli M jest postaci **if** $\gamma$ **then** $\mathrm{K_1}$ **else** $\mathrm{K_2}$ **fi**, to

$$Dom(\mathrm{M_A}) = \{v : v \in Dom(\mathrm{K_{1A}}) \ \text{ i } \ \mathbf{A}, v \models (ok(\gamma) \wedge \gamma)\} \cup \{v : v \in Dom(\mathrm{K_{2A}}) \ \text{ i } \ \mathbf{A}, v \models (ok(\gamma) \wedge \neg\gamma)\}$$

i dla $v \in Dom(\mathrm{M_A})$

$$\mathrm{M_A}(v) = \begin{cases} \mathrm{K_{1A}}(v), & \text{gdy } \ \mathbf{A}, v \models (ok(\gamma) \wedge \gamma) \\ \mathrm{K_{2A}}(v), & \text{gdy } \ \mathbf{A}, v \models (ok(\gamma) \wedge \neg\gamma) \end{cases}$$

(4) jeżeli M jest postaci **while** $\gamma$ **do** K **od**, to

$$\mathrm{M_A}(v) = \mathrm{K_A^i}(v),$$

gdzie $i$ jest najmniejszą liczbą naturalną taką, że dla $j < i$, $v \in Dom(\mathrm{K_A^j})$ i $\mathbf{A}, \mathrm{K_A^j}(v) \models (ok(\gamma) \wedge \gamma)$ oraz $\mathbf{A}, \mathrm{K_A^i}(v) \models (ok(\gamma) \wedge \neg\gamma)$. Jeżeli takie $i$ nie istnieje, to $v \notin Dom(\mathrm{M_A})$

Dowód lematu pozostawiamy czytelnikowi. ■

DEFINICJA 3.6

Niech L będzie ustalonym językiem pierwszego rzędu, $\pi$ klasą programów iteracyjnych nad tym językiem, **A** zaś strukturą danych dla L. Powiemy, że funkcja częściowa $n$-zmiennych $f(x_1, \ldots, x_n)$ jest programowalna w strukturze **A** w klasie $\pi$ wtedy i tylko wtedy, gdy istnieje program $\mathrm{M} \in \pi$ zależny (co najmniej) od zmiennych $x_1, \ldots, x_n$, $x_{n+1}$ i taki, że dla dowolnych $a_j \in A$ i dowolnego wartościowania $v$ takiego, że $v(x_j) = a_j$ dla $j \leqslant n$,

(1) $(a_1, \ldots, a_n) \in Dom(f)$ wtedy i tylko wtedy, gdy istnieje skończone,

udane obliczenie programu M przy wartościowaniu początkowym $v$ w **A** oraz

(2) jeżeli $(a_1, \ldots, a_n) \in Dom(f)$ i $v' = M_A(v)$, to $v'(x_{n+1}) = f(a_1, \ldots, a_n)$ ■

UWAGA

Każda funkcja częściowo rekurencyjna [21] jest programowalna w strukturze liczb naturalnych $\mathbf{N} = \langle N, 0, s; = \rangle$ w klasie deterministycznych programów iteracyjnych. ■

PRZYKŁAD 3.3

Rozważamy język pierwszego rzędu o sygnaturze $\langle 0, 1; 2, 2 \rangle$ oraz funkcję $div(x, y)$ określoną w zbiorze liczb naturalnych, której wynikiem jest część całkowita ilorazu $x/y$. Funkcja $div$ jest programowalna w strukturze $\langle N, O, s; \leqslant, = \rangle$ przez program M następującej postaci:

```
begin
  r := x; wynik := 0;
  while y ⩽ r
  do
    u := y; j := 0;
    while ¬r = u
    do
      u := s(u); j := s(j)
    od;
    r := j; wynik := s(wynik)
  od
end
```

Rzeczywiście, dla dowolnego wartościowania $v$, jeżeli $v(x) < v(y)$, to wartością zmiennej *wynik* po wykonaniu programu jest 0, a w przeciwnym razie, wartością zmiennej *wynik* jest największa liczba naturalna $n$ taka, że $n * v(y) \leqslant v(x)$. □

DEFINICJA 3.7

Niech L będzie ustalonym językiem pierwszego rzędu, $\pi$ klasą programów iteracyjnych nad tym językiem, **A** zaś strukturą danych dla L. Powiemy, że relacja $r(x_1 \ldots, x_m)$ typu $(i_1 \times \ldots \times i_m)$ jest programowalna w strukturze **A** w klasie $\pi$, jeżeli istnieje program $M \in \pi$ zależny od (co najmniej) zmiennych $x_1 \ldots, x_m$ i zmiennej zdaniowej $q$ taki, że dla dowolnych $a_j \in A$ i dowolnego wartościowania $v$ takiego, że $v(x_j) = a_j$ dla $j \leqslant m$, $(a_1 \ldots, a_m) \in r$ wtedy i tylko wtedy, gdy istnieje skończone, udane obliczenie programu M przy wartościowaniu początkowym $v$ oraz dla $v' = M_A(v)$, $v'(q) = \mathbf{1}$. ■

UWAGA

Każda relacja częściowo rekurencyjna [21] jest programowalna w strukturze liczb naturalnych $\mathbf{N} = \langle N, 0, s; = \rangle$ w klasie deterministycznych programów iteracyjnych. ■

PRZYKŁAD 3.4

Niech L będzie ustalonym językiem pierwszego rzędu o sygnaturze $\langle 0, 1; 2 \rangle$, a $\pi$ klasą programów iteracyjnych nad L. Rozważmy relację $\leqslant$ w zbiorze liczb naturalnych. Relacja ta jest programowalna w strukturze liczb naturalnych $\mathbf{N} = \langle N, 0, s; = \rangle$ przez program M następującej postaci:

```
begin
  u := x;
  w := y;
  q := true;
  while (q ∧ ¬u = y ∨ ¬q ∧ ¬x = w)
    do
      q := ¬q;
      if q then u := s(u) else w := s(w) fi
    od
  end
```

Zauważmy, że program M zatrzymuje się przy dowolnych danych początkowych w strukturze N. □

Na zakończenie przypomnijmy jeszcze raz, że program jest tworem formalnym. Sposób, w jaki będzie wykonywany, funkcja, jaką przedstawia, zależą od struktury danych, są zdeterminowane przez strukturę danych i ustalony wcześniej sposób rozumienia operacji programotwórczych.

PRZYKŁAD 3.5

Rozważmy program M postaci

```
begin
  z := x;
  u := y;
  while ¬z = 0 ∧ ¬u = 0
    do
      if z > u then z := z − u else u := u − z fi
    od
  if z = 0 then z := u fi
end
```

Wszystkie zmienne występujące w programie są tego samego typu, > jest dwuczłonowym predykatem, – dwuargumentowym funktorem, a 0 stałą.

(1) Niech

$$\mathbf{A}_1 = \langle C, 0, -; >, = \rangle$$

będzie strukturą liczb całkowitych ze stałą 0, jedną funkcją dwuargumentową – (odejmowanie) i dwoma relacjami > (relacja większości) i = (relacja równości). W strukturze $\mathbf{A}_1$, jeżeli dane spełniają warunek $v(x) \neq 0$ i $v(y) \neq 0$, to wartością zmiennej $z$ po wykonaniu programu M jest największy wspólny dzielnik liczb $v(x)$ i $v(y)$.

(2) Niech $\mathbf{A}_2$ będzie strukturą danych

$$\mathbf{A}_2 = \langle W[x], 0, -; = \rangle$$

gdzie $W[x]$ jest pierścieniem wielomianów nad ciałem liczb wymiernych, 0 jest zerem tego pierścienia, – jest operacją dwuargumentową taką, że $w_1 - w_2$ jest resztą z dzielenia $w_1$ przez $w_2$, > jest relacją porządkującą zbiór wielomianów ($w_1 > w_2$ wtedy i tylko wtedy, gdy wielomian $w_1$ ma wyższy stopień niż $w_2$ lub gdy współczynnik przy najwyższej potędze różniącej te wielomiany jest większy w wielomianie $w_1$). Po wykonaniu programu M w strukturze $\mathbf{A}_2$ wartością zmiennej $z$ jest największy wspólny dzielnik wielomianów będących wartościami zmiennych $x$ i $y$.

(3) Niech

$$\mathbf{A}_3 = \langle O, 0, -; >, = \rangle$$

będzie strukturą danych o uniwersum będącym zbiorem wszystkich odcinków $O$ na płaszczyźnie rzeczywistej i taką, że 0 jest odcinkiem pustym, – jest różnicą odcinków, a > relacją porównywania długości odcinków. Po wykonaniu programu M w strukturze $\mathbf{A}_3$ wartością zmiennej $z$ jest najdłuższy odcinek, który mieści się całkowitą ilość razy w odcinkach będących wartościami zmiennych $x$ i $y$. Zauważmy, że w tej strukturze program M nie zawsze ma obliczenie skończone. □

Definicja 3.8

Niech $Z$ będzie dowolnym zbiorem zmiennych. Powiemy, że dwa wartościowania $v$, $v'$ pokrywają się z dokładnością do zbioru $Z$, symbolicznie

$$v = v' \ \mathit{off}\ Z$$

jeżeli $v(x) = v'(x)$ dla wszystkich tych zmiennych $x$, które nie należą do $Z$. ■

Chociaż dane programu są opisane w naszej formalizacji za pomocą wartościowania początkowego, które jest w gruncie rzeczy nieskończonym ciągiem wartości zmiennych, to wynik programu nie w jednakowym stopniu

zależy od wszystkich jego elementów. Od czego więc zależy wynik programu K? Oczywiście nie zależy od zmiennych, które w nim nie występują. Zatem wynik programu K przy wartościowaniu początkowym $v$ w strukturze **A**, jeżeli istnieje, spełnia własność

$$v = \mathrm{K}_{\mathbf{A}}(v)\ \mathit{off}\ V(K)$$

Oznaczmy przez $V_{out}(K)$ zbiór zmiennych występujących w programie K po lewej stronie instrukcji przypisania. Jeżeli program K ma określony wynik w strukturze **A** przy wartościowaniu początkowym $v$, to na mocy definicji obliczenia, wartościowanie początkowe i wartościowanie wynikowe różnić się mogą co najwyżej wartościami zmiennych należących do zbioru $V_{out}(K)$, tzn.

$$\mathrm{K}_{\mathbf{A}}(v) = v\ \mathit{off}\ V_{out}(K)$$

Łatwo zauważyć, że wynik programu nie zależy od wartości początkowej tych zmiennych, które występują jedynie po lewej stronie instrukcji przypisania. Pełnią one rolę zmiennych pomocniczych programu. Przez analogię do pojęcia zmiennej związanej w formule, możemy je nazywać zmiennymi *związanymi programu.*

Niech $V_{in}(\mathrm{K})$ oznacza te zmienne programu K, które występują w testach lub po prawej stronie instrukcji przypisania. Od wartości zmiennych ze zbioru $V_{in}(\mathrm{K})$ może zależeć wynik programu, co więcej

$$v = v'\ \mathit{off}\,(V - V_{in}(\mathrm{K}))\quad \text{implikuje}\quad \mathrm{K}_{\mathbf{A}}(v) = \mathrm{K}_{\mathbf{A}}(v')$$

Zmienne ze zbioru $V_{in}(\mathrm{K})$ mają charakter *zmiennych wolnych programu.* Oczywiście zbiory $V_{in}(\mathrm{K})$ i $V_{out}(\mathrm{K})$ nie muszą być zbiorami rozłącznymi.

# Semantyczne własności programów 3.4

## *Problem stopu*

Jednym z podstawowych pytań, na które musi odpowiedzieć programista, jest następujące: czy napisany przez niego program M ma skończone obliczenie dla wszystkich danych początkowych i czy wynik programu jest zawsze określony? Inaczej mówiąc, czy program M ma obliczenia udane dla dowolnych danych. Sytuacja, w której program „zapętla się” (tzn. ma obliczenia nieskończone) lub w której wynik programu nie jest określony, jest (najczęściej) niedopuszczalna z punktu widzenia użytkownika oczekującego na wynik. Problem, o którym tu mówimy nosi nazwę problemu stopu. Można go rozważyć w trzech wariantach.

DEFINICJA 3.9

STOP(M) ≡ program M ma obliczenia skończone i udane dla wszystkich danych we wszystkich strukturach danych dla języka L.
STOP(M, **A**) ≡ program M ma obliczenia skończone i udane dla wszystkich danych początkowych w strukturze **A** dla języka L.
STOP(M, **A**, $v$) ≡ wynik programu M jest określony dla danych początkowych $v$ w strukturze **A**. ■

PRZYKŁAD 3.6

Niech $M_1$, $M_2$, $M_3$ będą programami w języku L takimi, że

$M_1$: **while** $q$ **do** $q := \neg q$ **od**
$M_2$: **while** $\neg x = 0$ **do** $x := x - 1$ **od**
$M_3$: **while** $\neg x = y$ **do** $x := x + 1$ **od**

Program $M_1$ zatrzymuje się i ma określony wynik w każdej strukturze dla dowolnych danych początkowych.
Program $M_2$ zatrzymuje się i ma określony wynik dla dowolnych danych początkowych w strukturze liczb naturalnych z zerem i poprzednikiem.
Program $M_3$ zatrzymuje się i ma określony wynik w strukturze liczb naturalnych z następnikiem, tylko dla takich danych początkowych $v$, dla których $v(x) \leqslant v(y)$. □

## *Poprawność programu*

Wyjaśnienia, czym jest poprawność programu, rozpoczniemy od przykładu.

PRZYKŁAD 3.7

Niech zadanie polega na znalezieniu programu M, który oblicza pierwiastek kwadratowy z danej liczby dodatniej $x$ z zadaną dokładnością $\varepsilon$ w strukturze liczb rzeczywistych **R**. Wartość pierwiastka niech będzie zapamiętana na zmiennej $y$.

Powiemy, że program M jest poprawnym rozwiązaniem naszego zadania, jeżeli dla wszystkich danych początkowych $v$ w strukturze **R** takich, że wartość zmiennej $x$ jest dodatnia, $v(x) > 0$, wynik $y$ programu M spełnia warunek $(\sqrt{x} - \varepsilon) < y < (\sqrt{x} + \varepsilon)$. Rozważmy program M postaci

**begin**
  $z := 0$;
  **if** $x < 1$ **then** $y := 1$ **else** $y := x$ **fi**;
  **while** $\neg |z - y| < \delta$

```
        do z := y;
           y := (z + x/z)/2
        od
    end
```

gdzie $\delta = \varepsilon/2 \times \max(1, x)$.

Program M jest poprawnym rozwiązaniem postawionego zadania. Rzeczywiście, kolejne kroki obliczenia programu M konstruują ciąg $y_i$ taki, że

$$y_0 = max(v(x), 1)$$
$$y_{i+1} = (y_i + v(x)/y_i)/2$$

Zauważmy, że dla każdego $i$, jeżeli $v(x) < 1$, to $v(x) \leqslant y_i \leqslant 1$, jeżeli $v(x) \geqslant 1$, to $1 \leqslant y_i \leqslant v(x)$. Ponadto $(\forall i)\ y_{i+1} \leqslant y_i$, a zatem ciąg ten jest zbieżny. Granicą ciągu $(y_i)$ jest $\sqrt{v(x)}$. Zatem program M nie zapętla się — ma obliczenie skończone, jeśli $v(x) \geqslant 0$. Co więcej, warunek wyjścia z pętli w programie M gwarantuje, że różnica między dwoma kolejnymi wyrazami ciągu jest dostatecznie mała $|y_{i+1} - y_i| < \delta$, a w konsekwencji $|y - \sqrt{x}| < \varepsilon$.

□

DEFINICJA 3.10

Program jest poprawny ze względu na warunek początkowy $\alpha$ i warunek końcowy $\beta$ w strukturze danych **A** wtedy i tylko wtedy, gdy dla dowolnego wartościowania $v$ w **A** spełnienie warunku $\alpha$ przez $v$, implikuje istnienie wyniku programu spełniającego warunek $\beta$. ■

Zwróćmy uwagę, że jeżeli program M jest poprawny ze względu na warunek początkowy $\alpha$ i warunek końcowy $\beta$, to spełnienie warunku początkowego gwarantuje istnienie skończonego, udanego obliczenia programu M. Znalezienie takiego warunku początkowego jest często trudne. Zadowalamy się wówczas nieco słabszą formą poprawności, tzw. częściową poprawnością programu.

DEFINICJA 3.11

Program M jest częściowo poprawny ze względu na warunek początkowy $\alpha$ i warunek końcowy $\beta$ w strukturze **A** wtedy i tylko wtedy, gdy dla dowolnych danych, dla których jest określony wynik programu M, spełnienie warunku $\alpha$ przez te dane implikuje spełnienie warunku $\beta$ przez wynik programu. ■

PRZYKŁAD 3.8

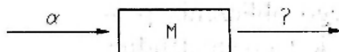
Niech M będzie następującym programem

```
begin
  z := 1;
  u := x; w := y;
  while y > 0
    do
      if even(w)
        then
          u := u * u; w := w/2
        else
          z := z * u; w := w - 1
      fi
    od
end
```

Rozważmy naturalną interpretację występujących w programie funktorów i predykatów w strukturze liczb rzeczywistych. Program M jest częściowo poprawny ze względu na warunek początkowy **true** i warunek końcowy $(z = x^y)$ oraz jest poprawny ze względu na warunek początkowy „$y$ jest liczbą naturalną” i warunek końcowy $(z = x^y)$. □

## *Najmocniejszy następnik*

Niech **A** będzie ustaloną strukturą danych, a M programem. Jakie własności będą miały wyniki programu, jeżeli wiadomo, że dane początkowe spełniają warunek $\alpha$, a program M ma dla tych danych obliczenie skończone, udane? Pytamy więc o własności przeciwdziedziny funkcji częściowej $M_{\mathbf{A}}$ (rys. 3.6).




*Najmocniejszy następnik* RYS. 3.6

DEFINICJA 3.12

Najmocniejszym następnikiem formuły $\alpha$ ze względu na program M w strukturze **A** będziemy nazywać taką formułę $\beta$, która spełnia następujące warunki:

(1) dla wszystkich danych początkowych, jeżeli obliczenie programu M ma określony wynik i warunek $\alpha$ jest spełniony, to wynik programu M spełnia warunek $\beta$ (krótko, $\beta$ jest następnikiem formuły $\alpha$ ze względu na program M);

(2) dla dowolnego warunku $\delta$, jeżeli $\delta$ jest następnikiem $\alpha$ ze względu na program M, to w strukturze **A**, $\beta$ implikuje $\delta$. ■

PRZYKŁAD 3.9

Rozważmy program M postaci

```
begin
  z := x; y := 1;
  while z - y ≥ 0
  do z := z - y
     y := y + 2
  od
end
```

w strukturze **R** liczb rzeczywistych ze zwykłą interpretacją symboli $+$, $-$, $0$, $1$, $2$, $=$, $\geqslant$. Zauważmy, że zmienna $y$ przyjmuje jako swoje wartości kolejne liczby nieparzyste. W rezultacie, po wykonaniu $i$-tej iteracji, zmienne $z$, $y$ przyjmują, odpowiednio wartości

$$v(x)-1-3-5-\ldots-(2i-1) \quad \text{oraz} \quad (2i+1)$$

Ponieważ $\sum_{0<j\leqslant i}(2j-1)=i^2$, więc wartością zmiennej $z$ po $i$-tej iteracji jest $v(x)-i^2$. Instrukcja **while** jest wykonywana tak długo aż różnica $v(x)-i^2-(2i+1)$ będzie mniejsza niż zero, tzn. aż znajdziemy taką liczbę naturalną $n$, że

$$(n+1)^2 > v(x) \quad \text{oraz} \quad n^2 < v(x)$$

Wartością zmiennej $z$ jest wtedy $v(x)-[\sqrt{v(x)}]^2$, a wartością $y$ jest $2[\sqrt{v(x)}]+1$. Mówiąc krócej, jeżeli dane początkowe $v$ spełniają warunek $x>0$, to program M ma udane obliczenie skończone, a wartością zmiennej $z$ jest, po wykonaniu programu M, odległość $v(x)$ od największego kwadratu liczby naturalnej, mniejszego niż $v(x)$.
Formuła

$$\beta = (z = x-[\sqrt{x}]^2)\wedge(y = 2[\sqrt{x}]+1)\wedge x>0$$

jest najmocniejszym następnikiem formuły $\alpha \equiv x>0$, ze względu na program M w rozważanej strukturze danych.

Rzeczywiście, jeżeli **R**, $v \models x>0$, to po wykonaniu programu M wartościowanie $M_{\mathbf{R}}(v)$ spełnia formułę $\beta$ zgodnie z przeprowadzoną wyżej analizą. Warunek (1) definicji 3.12 najmocniejszego następnika jest więc spełniony.

Niech $\delta$ będzie formułą taką, że dla dowolnego wartościowania $v$, **R**, $v \models \alpha$ i $v \in Dom(M_{\mathbf{R}})$ implikuje **R**, $M_{\mathbf{R}}(v) \models \delta$. Rozważmy dowolne wartościowanie $v'$ i niech **R**, $v' \models \beta$.

$$v'(y) = 2[\sqrt{v'(x)}]+1$$

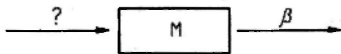
$$v'(z) = v'(x)-[\sqrt{v'(x)}]^2$$

$$v'(x) > 0$$

Zatem **R**, $v' \models \alpha$, a co za tym idzie program M ma, przy wartościowaniu początkowym $v'$, obliczenie skończone, czyli $v' \in Dom(M_R)$. Na mocy założenia mamy więc **R**, $M_R(v') \models (\delta \wedge \beta)$. Ponieważ zachowanie programu M zależy wyłącznie od zmiennej $x$, która zresztą nie ulega w czasie obliczenia żadnym zmianom, to wartościowanie wynikowe $M_R(v')$ może się różnić od wartościowania początkowego jedynie wartościami zmiennych $y, z$. Zgodnie z przeprowadzoną wcześniej analizą wartości zmiennych $z, y$, po wykonaniu programu M, spełniają warunki zawarte w $\beta$. W konsekwencji $M_R(v') = v'$ i **R**, $v' \models \delta$. Ostatecznie **R**, $v' \models (\beta \Rightarrow \delta)$ dla dowolnego wartościowania $v'$, tzn. **R** $\models (\beta \Rightarrow \delta)$. □

## *Najsłabszy warunek wstępny*

Niech M będzie programem a $\beta$ pewnym ustalonym warunkiem. Jakie warunki muszą spełniać dane początkowe programu M w strukturze **A**, by wyniki programu spełniały warunek $\beta$ (rys. 3.7)?
Inaczej mówiąc, jak scharakteryzować dziedzinę funkcji $M_A$?




*Najsłabszy warunek wstępny* RYS. 3.7

DEFINICJA 3.13

Najsłabszym warunkiem wstępnym formuły $\beta$ względem programu M w strukturze **A** nazywamy formułę $\alpha$ spełniającą następujące warunki:

(1) dla dowolnych danych początkowych w **A**, które spełniają warunek $\alpha$, wyniki programu M istnieją i spełniają warunek $\beta$. tzn. $\alpha$ jest warunkiem wstępnym formuły $\beta$ względem programu M;

(2) dla dowolnej formuły $\delta$, jeżeli $\delta$ jest warunkiem wstępnym formuły $\beta$ względem programu M, to w strukturze **A**, $\delta$ implikuje $\alpha$. ■

PRZYKŁAD 3.10

Rozważmy program M postaci

w strukturze kolejek **Q** (por. definicję 2.15). Zakładamy, że $x$ jest zmienną typu *kolejka*, a $y$ zmienną typu *element kolejki*.

Najsłabszym warunkiem wstępnym formuły $\neg em(x)$ względem programu M jest formuła $\neg em(put(x, y))$. Zauważmy jeszcze, że w rozważanej strukturze danych **Q**, $\neg em(put(x, y))$ jest formułą prawdziwą przy każdym wartościowaniu.

Sytuacja nie zawsze jest tak prosta jak w przedstawionym przykładzie. Niech K będzie programem postaci:

```
while ¬first(x) = y ∧ ¬em(x)
  do
    x := out(x)
  od
```

Najsłabszym warunkiem wstępnym formuły $\alpha$ postaci

$$(\neg em(x) \wedge first(x) = y)$$

względem programu K w strukturze kolejek **Q** jest warunek, który mówi, że $y$ jest elementem kolejki $x$. Warunek ten można zapisać w postaci nieskończonej alternatywy (nie jest to formuła pierwszego rzędu)

$$(\neg em(x) \wedge first(x) = y) \vee$$
$$(\neg em(out(x)) \wedge first(out(x)) = y) \vee \ldots \vee$$
$$(\neg em(out^{n-1}(x)) \wedge first(out^{n-1}(x)) = y) \vee \ldots$$

Jeżeli wartościowanie początkowe programu jest takie, że $v(x)$ jest $n$-elementową kolejką oraz $v(y) \in v(x)$, to po usunięciu co najwyżej $(n-1)$ pierwszych elementów otrzymamy kolejkę, której pierwszym elementem jest $v(y)$. □

## *Niezmienniki*

W bardzo wielu zadaniach interesujemy się nie tym co będzie po zakończeniu obliczenia programu, lecz tym co dzieje się w trakcie obliczenia. Nie tym, jak zmienią się warunki zadane na początku, lecz raczej jakie własności nie zmienią się podczas obliczeń. Do takiej grupy należą np.: programy symulacyjne i systemy operacyjne.

PRZYKŁAD 3.11

Rozważmy program, którego zadaniem jest prowadzenie kontroli rezerwacji miejsc w samolotach. Własnością, która musi być stale spełniona, jest. np.:

*liczba pasażerów* $\leqslant$ *liczba miejsc* □

DEFINICJA 3.14

Niezmiennikiem programu M w strukturze danych **A** nazywamy formułę $\alpha$ taką, że dla dowolnego obliczenia programu M w strukturze **A**, jeżeli wartościowanie początkowe spełnia $\alpha$, to dowolne wartościowanie uzyskane w czasie obliczenia też spełnia warunek $\alpha$. ■

## *Równoważność programów*

Niech K i M będą dwoma programami otrzymanymi jako rozwiązanie tego samego problemu. Czy realizują one rzeczywiście to samo zamierzenie? Kiedy można nazwać te programy równoważnymi?

Jedno z możliwych kryteriów równoważności programów może być następujące: dla dowolnego $x \in V$, wartość zmiennej $x$ po wykonaniu programu K jest taka sama jak wartość zmiennej $x$ po wykonaniu programu M dla tych samych danych początkowych.

PRZYKŁAD 3.12

Rozważmy dwa programy K, M w strukturze liczb rzeczywistych **R** ze zwykłą interpretacją występujących w programach symboli.

```
K:  begin
      z := 0,
      while ¬z = y
        do
          z := z+1;
          x := x+1
        od
    end

M:  begin
      z := y;
      while ¬z = 0
        do
          z := z−1;
          x := x+1
        od
    end
```

W sensie kryterium sformułowanego powyżej, programy te nie są równoważne, chociaż nie jest to zgodne z naszą intuicją. Oba programy liczą sumę wartości zmiennych $x$ i $y$, jeżeli wartością zmiennej $y$ jest liczba

naturalna i zapętlają się, jeżeli ten warunek nie jest spełniony. Programy K, M różnią się tylko zachowaniem („nieistotnej” dla obliczania sumy) zmiennej pomocniczej $z$. □

Z przykładu wynika, że sformułowane tutaj kryterium jest zbyt silne. Omawiany przykład sugeruje, że powinniśmy raczej ograniczyć się do pewnych wybranych zmiennych występujących w programach, uznać je za istotne do rozwiązywanego zadania i ze względu na te właśnie zmienne badać zachowanie programów. Kryterium równoważności zgodne z tą ideą mogłoby mieć następujące brzmienie:

DEFINICJA 3.16

Programy K, M są równoważne ze względu na zbiór zmiennych $X$ w strukturze A wtedy i tylko wtedy, gdy

(1) dla dowolnych danych początkowych $v$, K ma obliczenie skończone wtedy i tylko wtedy, gdy M ma obliczenie skończone oraz

(2) jeżeli dla dowolnie ustalonych danych początkowych oba programy mają obliczenia skończone i udane, to wyniki są identyczne na zbiorze zmiennych $X$. ■

PRZYKŁAD 3.13

Niech K i M będą programami obliczającymi pierwiastek kwadratowy z $x$, jeśli wartością zmiennej $x$ jest liczba dodatnia większa od 1. Wynik obliczenia jest zapamiętany na zmiennej $y$.

```
K: begin
     a := 1;   b := x;
     while   ¬(b − a) < δ
       do
          y := (a + b)/2;
          if (a² − x)(y² − x) ≤ 0
             then b := y   else a := y   fi
       od
   end

M: begin
     z := 0;   y := x;
     while   ¬|z − y| < δ
       do
          z := y;
          y := (z + x/z)/2
       od
   end
```

Program K oblicza pierwiastek kwadratowy metodą Newtona, zwaną też metodą bisekcji. Program M natomiast, oblicza pierwiastek kwadratowy z $x$ inną metodą iteracji. W sensie ostatnio rozważanego kryterium, programy K i M nie są równoważne ze względu na zmienną $y$; uzyskane przybliżone wartości pierwiastka kwadratowego z $x$ mogą się różnić dokładnością. Mimo to programy te można uznać za równoważne, gdyż wyliczają tę samą funkcję. □

Powyższy przykład prowadzi do innej definicji równoważności.

DEFINICJA 3.17

Programy K i M są równoważne w strukturze **A** ze względu na zbiór własności $Z$ wtedy i tylko wtedy, gdy dla dowolnego $\alpha \in Z$ i dla dowolnych danych początkowych wyniki jednego programu spełniają warunek $\alpha$ wtedy i tylko wtedy, gdy wyniki drugiego spełniają warunek $\alpha$. ■

Rozważane tutaj kryteria równoważności są jedynie przykładami możliwych definicji.

W punkcie tym chodziło nam jedynie o przedstawienie pewnych interesujących semantycznych własności programów. W dalszym ciągu będziemy się starali nie tylko wyrażać takie własności formułami w sformalizowanym języku, ale także dowodzić, że własności opisywane przez te formuły przysługują lub nie przysługują konkretnym programom.

# Język algorytmiczny 3.5

Zauważyliśmy wcześniej, że język pierwszego rzędu nie wystarcza na to, by móc wyrazić własności pewnych systemów relacyjnych. Co więcej, wielu podstawowych własności programów nie można wyrazić za pomocą formuł pierwszego rzędu.

PRZYKŁAD 3.14

Własność stopu nie jest wyrażalna w języku pierwszego rzędu.

DOWÓD

Przypuśćmy przeciwnie. Załóżmy, że dla dowolnego programu M istnieje formuła $\alpha_{\mathbf{M}}$ taka, że

$\models \alpha_{\mathbf{M}} \equiv$ program M zatrzymuje się dla dowolnych danych w każdej strukturze danych (3.1)

Rozważmy program K postaci

**begin** $x := 0$; **while** $\neg y = x$ **do** $x := succ(x)$ **od end**

oraz klasę struktur *Nat* podobnych do $\mathbf{N} = \langle N, 0; S; = \rangle$, które są modelami (por. definicję 2.28) następujących formuł pierwszego rzędu

$(\forall x)\ \neg succ(x) = 0$
$(\forall x)\ (\forall y)\ (succ(x) = succ(y) \Rightarrow x = y)$
$(\forall x)\ ok(succ(x))$

W każdym modelu Z klasy *Nat* interpretacją funktora *succ* musi być funkcja całkowitą i różnowartościową (por. przykład 2.20), która nie przyjmuje wartości odpowiadającej stałej 0.

Zanim przystąpimy do właściwego dowodu, zauważmy następujący pomocniczy fakt:

Jeżeli $\mathbf{A} \in Nat$ oraz program K zatrzymuje się dla dowolnych danych początkowych w strukturze **A**, to **A** jest strukturą izomorficzną z **N** (por. twierdzenie 4.9).

Ponieważ program K zatrzymuje się dla wszystkich danych początkowych w strukturze **N**, zatem w dowolnej strukturze izomorficznej z **N**, K ma wszystkie obliczenia skończone (por. twierdzenia 2.1 oraz 4.8). Stąd i z własności (3.1) mamy

$\mathbf{A} \models \alpha_K$ wtw
$Stop(\mathrm{K}, \mathbf{A})$ wtw
K zatrzymuje się dla dowolnych danych w strukturze **A** wtw
**A** jest izomorficzne ze strukturą standardowych liczb naturalnych N.

Ponieważ ta ostatnia własność nie jest wyrażalna w języku pierwszego rzędu, [21], więc nie może istnieć formuła $\alpha_K$ wyrażająca własność stopu w klasie struktur danych *Nat*. □

Przykład ten dowodzi, że aby mówić o własnościach programów czy o własnościach struktur danych, musimy rozszerzyć język zdefiniowany w poprzednim rozdziale [40].

Niech $\pi$ oznacza klasę programów iteracyjnych nad językiem pierwszego rzędu $L = \langle A, T, F \rangle$.

DEFINICJA 3.18

Zbiorem formuł algorytmicznych nad $\pi$ będziemy nazywać zbiór $F(\pi)$, który jest najmniejszym zbiorem spełniającym następujące warunki:

(1) $F \subset F(\pi)$;

(2) jeżeli $\alpha, \beta \in F(\pi)$, to wyrażenia $\neg\alpha$, $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \Rightarrow \beta)$, $(\exists x)\alpha$, $(\forall x)\alpha$ są elementami $F(\pi)$;

(3) jeżeli $\alpha \in F(\pi)$ a $K \in \pi$, to wyrażenia $(K\alpha)$, $\bigcup K\alpha$, $\bigcap K\alpha$ należą do zbioru $F(\pi)$.
Formuły, w których występują programy, np. formuły postaci $(K\alpha)$, nazywamy algorytmicznymi. Symbole $\bigcap$ i $\bigcup$ nazywamy kwantyfikatorami iteracji w odróżnieniu od kwantyfikatorów klasycznych $\exists$ i $\forall$. ∎

DEFINICJA 3.19

Językiem algorytmicznym nad klasą programów $\pi$ będziemy nazywać system $L(\pi) = \langle A(\pi), T, F(\pi), \pi \rangle$ taki, że $A(\pi)$ jest rozszerzeniem alfabetu języka L o symbole konstrukcji programotwórczych **begin**, **end**, **if**, **then**, **else**, **fi**, **while**, **do**, **od**, T jest zbiorem termów (por. p. 2.3), $F(\pi)$ jest zbiorem formuł, a $\pi$ jest zbiorem programów. ∎

Znaczenie termów i formuł klasycznych języka $L(\pi)$ w strukturze danych **A** jest określone tak jak dla języka pierwszego rzędu (por. p. 2.3). Dla pełnego określenia semantyki języka $L(\pi)$ wystarczy zdefiniować znaczenie formuł algorytmicznych.

DEFINICJA 3.20

Niech M będzie dowolnym programem, $M \in \pi$, $\alpha$ dowolną formułą, $\alpha \in F(\pi)$, a $v$ dowolnym wartościowaniem w strukturze danych **A**.

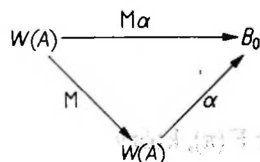
$\mathbf{A}, v \models (M\alpha)$ wtedy i tylko wtedy, gdy obliczenie programu M dla danych początkowych $v$ jest skończone i udane, a wynik tego obliczenia spełnia formułę $\alpha$.
Znaczenie kwantyfikatorów iteracji określają następujące równoważności:

$\mathbf{A}, v \models \bigcup M\alpha$ wtw $(\exists i)$ $\mathbf{A}, v \models (M^i\alpha)$

$\mathbf{A}, v \models \bigcap M\alpha$ wtw $(\forall i)$ $\mathbf{A}, v \models (M^i\alpha)$ ∎

Na mocy definicji 3.20, znaczenie formuły algorytmicznej $(M\alpha)$ otrzymujemy przez złożenie znaczenia programu M ze znaczeniem formuły $\alpha$ (rys. 3.8).

Podobnie jak w rozdz. 2 (por. definicję 2.27), przyjmujemy następujące definicje:




RYS. 3.8

Formuła $\alpha$ jest spełniona w strukturze **A** przez wartościowanie $v$ wtedy i tylko wtedy, gdy $\mathbf{A}, v \models \alpha$ (lub inaczej $\alpha_{\mathbf{A}}(v) = \mathbf{1}$). Formuła $\alpha$ jest prawdziwa w strukturze **A** , gdy jest spełniona przez dowolne wartościowanie w **A**, tzn.

$\mathbf{A} \models \alpha$ (lub inaczej $\mathbf{A}$ jest modelem formuły $\alpha$); jeżeli w każdej strukturze $\mathbf{A} \models \alpha$, to $\alpha$ nazywamy tautologią.

PRZYKŁAD 3.15

Posługując się definicją semantyki formuł algorytmicznych zbadajmy wartość następującej formuły:

$$(q := \gamma)\alpha \equiv (\mathbf{if}\ \gamma\ \mathbf{do}\ q := \mathbf{true}\ \mathbf{else}\ q := \mathbf{false}\ \mathbf{fi})\alpha$$

Zauważmy, że programy występujące w tej równoważności, mają jednocześnie określone lub jednocześnie nieokreślone wyniki w dowolnej strukturze i przy dowolnym wartościowaniu. Mamy

$\mathbf{A}, v \models (q := \gamma)\alpha$ wtw

$\mathbf{A}, v \models ok(\gamma)$ i $\mathbf{A}, (q := \gamma)_{\mathbf{A}}(v) \models \alpha$ wtw

$\mathbf{A}, v \models ok(\gamma)$ i istnieje $v'$ takie, że $\mathbf{A}, v' \models \alpha$ oraz $v'(q) = \gamma_{\mathbf{A}}(v)$ i $v'(z) = v(z)$ dla $z \neq q$ wtw

Istnieje $v'$ takie, że

$\mathbf{A}, v' \models \alpha$, $v'(q) = \mathbf{1}$, $v'(z) = v(z)$ dla $z \neq q$ i $\mathbf{A}, v \models (ok(\gamma) \wedge \gamma)$ lub

$\mathbf{A}, v' \models \alpha$, $v'(q) = \mathbf{0}$, $v'(z) = v(z)$ dla $z \neq q$ i $\mathbf{A}, v \models (ok(\gamma) \wedge \neg\gamma)$ wtw

Istnieje $v'$ takie, że $\mathbf{A}, v' \models \alpha$, $v' = (q := \mathbf{true})_{\mathbf{A}}(v)$ oraz $\mathbf{A}, v \models (ok(\gamma) \wedge \gamma)$ lub

$\mathbf{A}, v' \models \alpha$, $v' = (q := \mathbf{false})_{\mathbf{A}}(v)$ oraz $\mathbf{A}, v \models (ok(\gamma) \wedge \neg\gamma)$ wtw

$\mathbf{A}, v \models ok(\gamma)$ i istnieje $v'$ takie, że

$\mathbf{A}, v' \models \alpha$ oraz $v' = (\mathbf{if}\ \gamma\ \mathbf{do}\ q := \mathbf{true}\ \mathbf{else}\ q := \mathbf{false}\ \mathbf{fi})_{\mathbf{A}}(v)$ wtw

$\mathbf{A}, v \models (\mathbf{if}\ \gamma\ \mathbf{do}\ q := \mathbf{true}\ \mathbf{else}\ q := \mathbf{false}\ \mathbf{fi})\alpha$

Rozumowanie to dowodzi, że formuła

$$(q := \gamma)\alpha \equiv (\mathbf{if}\ \gamma\ \mathbf{do}\ q := \mathbf{true}\ \mathbf{else}\ q := \mathbf{false}\ \mathbf{fi})\alpha$$

jest spełniona przez dowolne wartościowanie w dowolnej strukturze danych, tzn. wartością tej formuły jest **1** (prawda). □

UWAGA

Zauważmy, na marginesie podanej definicji znaczenia formuł algorytmicznych, że jeżeli obliczenie programu M jest nieskończone albo nieudane w pewnej strukturze A, to niezależnie od $\alpha$, wartością formuły algorytmicznej $(\mathrm{M}\alpha)$ jest **0** (fałsz). ■

LEMAT 3.2

Formuła

$$\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{K}\ \mathbf{else}\ \mathrm{M}\ \mathbf{fi}\ \alpha \equiv (ok(\gamma) \wedge ((\gamma \wedge \mathrm{K}\alpha) \vee (\neg\gamma \wedge \mathrm{M}\alpha))) \qquad (3.2)$$

jest prawdziwa w dowolnej strukturze danych dla języka algorytmicznego $L(\pi)$, w którym $\alpha$, $\gamma$ są formułami, a K, M programami.

DOWÓD

Niech **A** będzie dowolnie ustaloną strukturą danych dla $L(\pi)$, a $v$ dowolnym wartościowaniem w **A** takim, że

$$\mathbf{A}, v \models \textbf{if } \gamma \textbf{ then } K \textbf{ else } M \textbf{ fi } \alpha$$

Na mocy definicji znaczenia formuł, istnieje obliczenie skończone, udane programu **if** $\gamma$ **then** K **else** M **fi**, którego wynik $v'$ spełnia $\alpha$. Z lematu 3.1 mamy

$$\begin{aligned}
&v' \in \textbf{if } \gamma \textbf{ then } K \textbf{ else } M \textbf{ fi}_{\mathbf{A}}(v) \quad \text{wtw} \\
&\mathbf{A}, v \models (ok(\gamma) \wedge \gamma) \quad \text{i} \quad v' = \mathbf{K}_{\mathbf{A}}(v) \quad \text{albo} \\
&\mathbf{A}, v \models (ok(\gamma) \wedge \neg\gamma) \quad \text{i} \quad v' = M_{\mathbf{A}}(v)
\end{aligned}$$

Łącząc te fakty otrzymamy

$$\mathbf{A}, v \models (ok(\gamma) \wedge ((\gamma \wedge K\alpha) \vee (\neg\gamma \wedge M\alpha)))$$

Ponieważ wszystkie kroki w przeprowadzonym rozumowaniu prowadzą do zdań równoważnych, zatem

$$\mathbf{A}, v \models \textbf{if } \gamma \textbf{ then } K \textbf{ else } M \textbf{ fi } \alpha \equiv (ok(\gamma) \wedge ((\gamma \wedge K\alpha) \vee (\neg\gamma \wedge M\alpha)))$$

W ten sposób pokazaliśmy, że formuła (3.2) jest spełniona przez dowolne wartościowanie w strukturze **A**, a więc jest prawdziwa w **A**. □

LEMAT 3.3

Formuła

$$\textbf{while } \gamma \textbf{ do } M \textbf{ od } \alpha \equiv \bigcup \textbf{if } \gamma \textbf{ then } M \textbf{ fi } (\neg\gamma \wedge ok(\gamma) \wedge \alpha) \tag{3.3}$$

jest prawdziwa w dowolnej strukturze danych dla języka algorytmicznego $L(\pi)$, w którym $\alpha$ i $\gamma$ są formułami, a M programem.

DOWÓD

Niech **A** będzie dowolnie ustaloną strukturą danych dla $L(\pi)$, a $v$ dowolnym wartościowaniem w **A** takim, że

$$\mathbf{A}, v \models \textbf{while } \gamma \textbf{ do } M \textbf{ od } \alpha$$

Istnieje więc skończone, udane obliczenie programu **while** $\gamma$ **do** M **od**, którego wynik spełnia $\alpha$. Niech $v' = (\textbf{while } \gamma \textbf{ do } M \textbf{ od})_{\mathbf{A}}(v)$. Na mocy lematu 3.1 istnieje takie $i \in N$, że $\mathbf{A}, v \models M^j(ok(\gamma) \wedge \gamma)$ dla $j < i$ oraz $v' = M^i_{\mathbf{A}}(v)$

i $\mathbf{A}, v \models \mathrm{M}^i(\neg\gamma \wedge ok(\gamma))$. W konsekwencji obliczenie programu $(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^i$ jest skończone i udane oraz istnieje $v'$ takie, że

$$v' = (\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^i_{\mathbf{A}}(v), \quad \mathbf{A}, v' \models (\neg\gamma \wedge ok(\gamma) \wedge \alpha)$$

Z definicji znaczenia kwantyfikatora iteracji (por. definicję 3.20) mamy więc

$$\mathbf{A}, v \models \bigcup \mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi}(\neg\gamma \wedge ok(\gamma) \wedge \alpha) \tag{3.4}$$

Odwrotnie, niech warunek (3.4) będzie spełniony dla pewnego wartościowania $v$ w $\mathbf{A}$. Istnieje wtedy takie $i$, że

$$\mathbf{A}, v \models (\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^i(\neg\gamma \wedge ok(\gamma) \wedge \alpha)$$

Niech $n$ będzie najmniejszą liczbą naturalną $i$ spełniającą powyższą własność. Skoro obliczenie $(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^n$ jest skończone i udane, zatem były udane obliczenia programów $(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^j$ dla wszystkich $j \leqslant n$. Co więcej, jeżeli dla jakiegoś $j$, wynik programu $(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^j$ spełnia $(\neg\gamma \wedge ok(\gamma))$, to wynik programu $(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^{j+1}$ też spełnia tę formułę (wyniki tych programów są identyczne). Zatem $n$ jest najmniejszą liczbą naturalną taką, że

$$\mathbf{A}, v \models (\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^n(\neg\gamma \wedge ok(\gamma) \wedge \alpha),$$
$$\mathbf{A}, v \models (\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^j\gamma \quad \text{dla} \quad j < n \quad \text{oraz}$$
$$(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^j_{\mathbf{A}}(v) = \mathrm{M}^j_{\mathbf{A}}(v) \quad \text{dla} \quad j \leqslant n$$

Stąd, $n$ jest najmniejszą liczbą naturalną taką, że $\mathbf{A}, v \models \mathrm{M}^j(\gamma \wedge ok(\gamma))$ dla $j < n$ oraz dla $v' = \mathrm{M}^n_{\mathbf{A}}(v)$, $\mathbf{A}, v' \models (\neg\gamma \wedge ok(\gamma) \wedge \alpha)$. Na mocy lematu 3.1 program **while** $\gamma$ **do** M **od** ma obliczenie skończone, a jego wynik spełnia formułę $\alpha$, czyli

$$\mathbf{A}, v \models \mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha$$

Wykazaliśmy zatem, że przy dowolnym wartościowaniu w strukturze $\mathbf{A}$ własność (3.3) jest spełniona, tzn. jest prawdziwa w strukturze danych $\mathbf{A}$. □

LEMAT 3.4

Następująca formuła jest prawdziwa w dowolnej strukturze danych dla dowolnej formuły $\alpha$ i dowolnego programu K

$$(\alpha \wedge \bigcap \mathrm{K}(\alpha \Rightarrow \mathrm{K}\alpha)) \Rightarrow \bigcap \mathrm{K}\alpha$$

DOWÓD

Załóżmy, że dla pewnego wartościowania w strukturze danych

$$\mathbf{A}, v \models \alpha \wedge \bigcap \mathrm{K}(\alpha \Rightarrow \mathrm{K}\alpha)$$

Na mocy definicji kwantyfikatora iteracji $\bigcap$, mamy $\mathbf{A}, v \models \alpha$ oraz dla

każdego $i$, $\mathbf{A}, v \models K^i(\alpha \Rightarrow K\alpha)$. Zatem dla każdego $i \in N$, jeśli $\mathbf{A}, v \models K^i\alpha$, to $\mathbf{A}, v \models K^{i+1}\alpha$. Na mocy zasady indukcji matematycznej dla każdego $i \geqslant 0$, $\mathbf{A}, v \models K^i\alpha$, czyli $\mathbf{A}, v \models \bigcap K\alpha$. Ponieważ zarówno struktura, jak i wartościowanie były dowolnie ustalone, więc formuła $((\alpha \wedge \bigcap K(\alpha \Rightarrow K\alpha)) \Rightarrow \bigcap K\alpha)$ jest tautologią. Formułę, której prawdziwość właśnie udowodniliśmy, możemy traktować jako *algorytmiczną postać zasady indukcji.* □

PRZYKŁAD 3.16

Niech $\mathbf{S}$ będzie strukturą stosów (por. def. 2.14).
Rozważmy formułę postaci

$$M(empty(x) \wedge empty(y) \wedge bool)$$

gdzie *bool* jest zmienną zdaniową, a M jest następującym programem:

```
begin
  bool := true;
  while (¬empty(x) ∧ ¬empty(y) ∧ bool)
    do
      bool := bool ∧ top(x) = top(y);
      x := pop(x);
      y := pop(y)
    od
end.
```

(1) Dla dowolnego wartościowania $v$ w strukturze $\mathbf{S}$ mamy

$$\mathbf{S}, v \models M(empty(x) \wedge empty(y) \wedge bool)$$

wtedy i tylko wtedy, gdy stos $x$ ma taką samą zawartość jak stos $y$.

(2) Niech K oznacza program zawarty między **do** i **od** w podanym przykładzie, a $\gamma = (\neg empty(x) \wedge \neg empty(y) \wedge bool)$, wówczas

$$\mathbf{S} \models \bigcup \textbf{if } \gamma \textbf{ then } K \textbf{ fi } \neg\gamma$$

dla każdego wartościowania $v$ w $\mathbf{S}$.
Rzeczywiście, jeżeli $v$ jest dowolnym wartościowaniem, a $i$ równe jest minimum z długości stosów $v(x)$ i $v(y)$, to po $i$-krotnym usuwaniu elementów, co najmniej jeden ze stosów będzie pusty, a więc

$$\mathbf{S}, v \models (\textbf{if } \gamma \textbf{ then } K \textbf{ fi})^i \neg\gamma$$

(3) Jeżeli $v(x)$ jest stosem o $n$ elementach, to

$$\mathbf{S}, v \models \bigcap \textbf{if } \gamma \textbf{ then } K \textbf{ fi true} \equiv (\textbf{if } \gamma \textbf{ then } K \textbf{ fi})^n \neg\gamma$$ □

Formuły algorytmiczne postaci $M\alpha$ są wygodnym i pożytecznym narzędziem umożliwiającym formułowanie własności algorytmów. W pewnych

zastosowaniach (por. rozdz. 5) jest wygodne stosowanie takiej odmiany języka algorytmicznego, która dopuszcza oprócz formuł również termy algorytmiczne.

DEFINICJA 3.21

Zbiór termów algorytmicznych jest to najmniejszy zbiór zawierający zmienne indywiduowe i taki, że

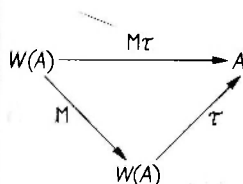
(1) jeżeli $\tau_1, ..., \tau_n$ są termami algorytmicznymi typu $t_1, ..., t_n$ i $\varphi$ jest $n$-argumentowym funktorem typu $t_1 \times ... \times t_n \to t$, to wyrażenie postaci $\varphi(t_1, ..., t_n)$ jest termem algorytmicznym typu $t$,

(2) jeżeli $\tau$ jest termem algorytmicznym typu $t$ oraz M jest dowolnym programem, to wyrażenie postaci $\mathrm{M}\tau$ jest termem algorytmicznym typu $t$. ■

UWAGA

W myśl definicji 3.21 zbiór termów klasycznych jest właściwym podzbiorem zbioru termów algorytmicznych. ■

Semantyka wyrażeń postaci $\mathrm{M}\tau$ jest oparta na tej samej idei co semantyka formuł $\mathrm{M}\alpha$ i jest przedstawiona na rys. 3.9.




RYS. 3.9

Dla dowolnie ustalonej struktury **A** i wartościowania $v$, wynik $(\mathrm{M}\tau)_\mathbf{A}(v)$ jest określony wtedy i tylko wtedy, gdy program M ma udane, skończone obliczenie przy wartościowaniu $v$ i dla wyniku $v'$ tego obliczenia wartość $\tau_\mathbf{A}(v')$ jest określona. Ponadto, jeżeli wartości $(\mathrm{M}\tau)_\mathbf{A}(v)$ oraz $\tau_\mathbf{A}(v')$ są określone i $v' = \mathrm{M}_\mathbf{A}(v)$, to

$$(\mathrm{M}\tau)_\mathbf{A}(v) = \tau_\mathbf{A}(\mathrm{M}_\mathbf{A}(v)) \tag{3.5}$$

PRZYKŁAD 3.17

Niech M będzie programem postaci

**while** $x \geqslant y$ **do** $x := x - y$ **od**

Wówczas term algorytmiczny $(\mathrm{M}x)$ definiuje w strukturze liczb naturalnych funkcję dwu zmiennych $x$, $y$, której wartością jest reszta z dzielenia $x$ przez $y$. □

Łatwo zauważyć następującą własność przyjętej semantyki termów algorytmicznych: w dowolnej strukturze danych i dla dowolnego wartościowania $v$, jeżeli wartości wszystkich termów występujących w następujących wyrażeniach są określone przy wartościowaniu $v$ oraz program M ma wyniki określone dla danych $v$, to

$$\mathbf{A}, v \models \mathrm{M}\varphi(\tau_1, \ldots, \tau_n) = \varphi((\mathrm{M}\tau_1), \ldots, (\mathrm{M}\tau_n))$$
$$A, v \models \mathrm{M}'(\mathrm{M}\tau) = (\mathbf{begin}\ \mathrm{M}';\ \mathrm{M}\ \mathbf{end})\tau$$

Uwaga ta pozwala udowodnić następujący lemat:

LEMAT 3.5

Dla dowolnego termu algorytmicznego $\tau$ istnieją: program M i zmienna $x$ takie, że dla dowolnej struktury **A** dowolnego wartościowania, program M ma określony wynik dla wartościowania początkowego $v$ wtedy i tylko wtedy, gdy wartość termu $\tau$ jest określona dla $v$ oraz gdy obie wartości $(\mathrm{M}x)_\mathbf{A}(v)$, $\tau_\mathbf{A}(v)$ są określone to

$$(\mathrm{M}x)_\mathbf{A}(v) = \tau_\mathbf{A}(v)$$ ■

Wynika stąd, że termy algorytmiczne można wyeliminować z języka. Wprowadzenie termów algorytmicznych nie rozszerza istotnie możliwości definiowania w języku algorytmicznym.

# Wyrażalność w języku algorytmicznym 3.6

W tym rozdziale pokażemy, że wszystkie wymienione do tej pory podstawowe własności programów i struktur danych są wyrażalne w języku algorytmicznym $L(\pi)$.

We wszystkich przytoczonych dalej faktach **A** jest dowolną strukturą danych, M, K są dowolnymi programami a $\alpha$, $\beta$ dowolnymi formułami w języku $L(\pi)$.

LEMAT 3.6

$\mathbf{A} \models \mathrm{M}\mathbf{true}$ wtedy i tylko wtedy, gdy wszystkie obliczenia programu M w strukturze **A** są skończone i udane.

DOWÓD

Jeżeli obliczenie programu M przy pewnych danych początkowych jest skończone, a wynik programu jest określony, to spełnia on formułę **true**. Odwrotnie, jeżeli przy pewnym wartościowaniu $v$, $\mathbf{A}, v \models \mathrm{M}\mathbf{true}$, to zgodnie

z przyjętym znaczeniem formuł algorytmicznych, istnieje udane, skończone obliczenie programu M. □

Oznaczmy przez *loop*(M) formułę zdefiniowaną rekurencyjnie w następujący sposób:

$$loop(x := w) \stackrel{\mathrm{df}}{=} \mathbf{false}$$

$$loop(\mathbf{begin}\ M_1;\ M_2\ \mathbf{end}) \stackrel{\mathrm{df}}{=} loop(M_1) \vee (M_1\, loop(M_2))$$

$$loop(\mathbf{if}\ \gamma\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2\ \mathbf{fi}) \stackrel{\mathrm{df}}{=} ok(\gamma) \wedge ((\gamma \wedge loop(M_1)) \vee (\neg\gamma \wedge loop(M_2)))$$

$$loop(\mathbf{while}\ \gamma\ \mathbf{do}\ M\ \mathbf{od}) \stackrel{\mathrm{df}}{=} \bigcap M\,(ok(\gamma) \wedge \gamma) \vee \bigcup \mathbf{if}\ \gamma\ \mathbf{then}\ M\ \mathbf{fi}\ (\gamma \wedge ok(\gamma) \wedge loop(M))$$

LEMAT 3.7

Dla dowolnego wartościowania $v$ w strukturze **A**, $\mathbf{A}, v \models loop(M)$ wtedy i tylko wtedy, gdy program M ma dla danych $v$ obliczenie nieskończone.

DOWÓD

Dowód przeprowadzimy przez indukcję ze względu na stopień komplikacji programu M. Jeżeli program M jest instrukcją przypisania, to M nie ma obliczenia nieskończonego bez względu na strukturę danych i wartościowanie. Załóżmy, że twierdzenie jest prawdziwe dla programu K oraz że M jest postaci **while** $\gamma$ **do** K **od**. Jeżeli $\mathbf{A}, v \models loop(M)$, to na mocy definicji

$$\mathbf{A}, v \models \bigcup \mathbf{if}\ \gamma\ \mathbf{then}\ K\ \mathbf{fi}\ (ok(\gamma) \wedge \gamma \wedge loop(K))$$

lub

$$\mathbf{A}, v \models \bigcap K\,(ok(\gamma) \wedge \gamma)$$

Rozważmy pierwszy przypadek. Zatem istnieje liczba naturalna $n$ (rozważmy najmniejszą taką liczbę $n$) taka, że

$$\mathbf{A}, v \models \mathbf{if}\ \gamma\ \mathbf{then}\ K\ \mathbf{fi}^n (ok(\gamma) \wedge \gamma \wedge loop(K))$$

Po $i$-krotnej iteracji programu K, $i < n$, otrzymane wartościowanie pozwala poprawnie obliczyć wartość formuły $\gamma$ i co więcej spełnia ją. Istnieją więc wartościowania $v_0 = v, v_1, \dots, v_{n-1}, v_n$ takie, że $v_i = K_{\mathbf{A}}(v_{i-1})$ dla $i < n$ oraz $\mathbf{A}, v_n \models (ok(\gamma) \wedge \gamma \wedge loop(K))$. Niech $\theta_i$ będzie obliczeniem programu K przy wartościowaniu początkowym $v_i$, $i \leqslant n$. Na mocy założenia indukcyjnego $\theta_n$ jest obliczeniem nieskończonym. Wówczas ciąg $\theta_1 \theta_2 \dots \theta_n$ jest obliczeniem nieskończonym programu **while** $\gamma$ **do** K **od**.

W drugim przypadku, jeżeli $\mathbf{A}, v \models \bigcap K\,(ok(\gamma) \wedge \gamma)$, to każda iteracja programu K ma obliczenie skończone, a jej wynik spełnia $\gamma$ i obliczenie

wartości formuły jest poprawne. Oznacza to, że pętla **while** $\gamma$ **do** K **od** będzie wykonywana nieskończenie wiele razy.

Dla dowodu implikacji odwrotnej zauważmy, że warunkiem koniecznym na to, by program M miał obliczenie nieskończone przy wartościowaniu początkowym $v$ jest **A**, $v \models (ok(\gamma) \wedge \gamma)$. Co więcej, nieskończone obliczenie programu M może być spowodowane albo tym, że program K ma stale obliczenie skończone udane, a otrzymany wynik kolejnych iteracji spełnia formułę $(ok(\gamma) \wedge \gamma)$, albo po pewnej skończonej ilości iteracji, program K ma obliczenie nieskończone. Te dwa przypadki odpowiadają spełnieniu formuły $\bigcap$ K $(ok(\gamma) \wedge \gamma)$ lub $\bigcup$ **if** $\gamma$ **then** K **fi** $(ok(\gamma) \wedge \gamma \wedge loop(\mathrm{K}))$.
W rezultacie mamy **A**, $v \models loop(\mathrm{M})$.

Dowód lematu w przypadku innych postaci programu M przebiega analogicznie. □

Na zakończenie analizy różnych rodzajów obliczeń rozważmy obliczenia skończone nieudane. Z obliczeniem nieudanym mamy do czynienia, gdy w trakcie obliczeń napotkamy operację o nieokreślonej, przy aktualnym wartościowaniu, wartości. Podobnie jak w przypadku zapętlania się programu, własność tę zdefiniujemy rekurencyjnie.

$fail(z := w) \overset{df}{=} \neg\, ok(w)$

$fail(\textbf{if}\ \gamma\ \textbf{then}\ \mathrm{M}_1\ \textbf{else}\ \mathrm{M}_2\ \textbf{fi}) \overset{df}{=}$

$\overset{df}{=} \neg\, ok(\gamma) \vee (\gamma \wedge fail(M_1)) \vee (\neg\, \gamma \wedge fail(M_2))$

$fail(\textbf{begin}\ \mathrm{M}_1;\ \mathrm{M}_2\ \textbf{end}) \overset{df}{=} fail(\mathrm{M}_1) \vee \mathrm{M}_1\, fail(\mathrm{M}_2)$

$fail(\textbf{while}\ \gamma\ \textbf{do}\ \mathrm{M}\ \textbf{od}) \overset{df}{=}$

$\overset{df}{=} \neg\, ok(\gamma) \vee \bigcup\ \textbf{if}\ \gamma\ \textbf{then}\ \mathrm{M}\ \textbf{fi}\ (\neg\, ok(\gamma) \vee (\gamma \wedge fail(\mathrm{M})))$

Uważny czytelnik dostrzeże, że wszystkie wystąpienia formuły $ok(w)$, gdzie $w$ jest termem lub formułą, można zastąpić w tej definicji przez $(z := w)$ **true**. Pozwala to zdefiniować własność *fail* także w języku, w którym nie występuje predykat $=$.

LEMAT 3.8

Dla dowolnego wartościowania $v$ w **A**, **A**, $v \models fail$ (M) wtedy i tylko wtedy, gdy program M ma przy wartościowaniu początkowym $v$ obliczenie skończone nieudane. ■

Dowód przebiega przez indukcję ze względu na stopień komplikacji programu i jest analogiczny do dowodu przeprowadzonego w przypadku formuły *loop*.

Badanie własności *fail* jest niesłusznie lekceważone przez wielu programistów. Tymczasem, jeśli konstruowany program ma być odporny na błędy użytkownika, musimy sobie zdawać sprawę z istnienia lub nie sytuacji

wyjątkowych w naszym programie. Twórca programu powinien uwzględnić, że w trakcie obliczenia może pojawić się stan, w którym argumenty pewnej operacji nie należą do jej dziedziny i odpowiednio na to zareagować. Dzięki wprowadzeniu odpowiednich testów lub innych mechanizmów (por. mechanizm exception handling w językach programowania np. Ada, Loglan [9]), tak zmodyfikować program, by reagował on poprawnie na wszystkie potknięcia użytkownika.

LEMAT 3.9

$\mathbf{A} \models (\alpha \Rightarrow \mathbf{M}\beta)$ wtedy i tylko wtedy, gdy program **M** jest poprawny w **A** ze względu na warunek wstępny $\alpha$ i warunek końcowy $\beta$.

DOWÓD

Niech dla pewnego wartościowania $\mathbf{A}, v \models (\alpha \Rightarrow \mathbf{M}\beta)$. Jeżeli $\mathbf{A}, v \models \alpha$, tzn. warunek wstępny jest spełniony, to na mocy założenia $\mathbf{A}, v \models \mathbf{M}\beta$. Zgodnie z przyjętą semantyką, istnieje skończone i udane obliczenie programu **M** przy wartościowaniu początkowym $v$ oraz jest określony wynik $v' = \mathbf{M}_{\mathbf{A}}(v)$ tego obliczenia taki, że $\mathbf{A}, v' \models \beta$. A zatem jest spełniony warunek końcowy $\beta$.

Odwrotnie, spełnienie warunku początkowego $\alpha$ gwarantuje istnienie skończonego udanego obliczenia (por. definicję 3.10) i wyniku spełniającego $\beta$. Na mocy definicji spełniania formuł, jeśli $\mathbf{A}, v \models \alpha$, dla pewnego $v$, to również $\mathbf{A}, v \models \mathbf{M}\beta$. Zatem $A, v \models (\alpha \Rightarrow \mathbf{M}\beta)$. □

Częściową poprawność programu **M** ze względu na warunek wstępny $\alpha$ i warunek końcowy $\beta$ wyraża formuła

$$((\alpha \wedge \mathbf{M}\,\mathbf{true}) \Rightarrow \mathbf{M}\beta)$$

Prawdziwość tej formuły w strukturze danych **A** oznacza, że jeżeli tylko dane początkowe spełniają formułę $\alpha$ oraz program **M** ma udane obliczenie, to wyniki programu **M** spełniają formułę $\beta$. Jeżeli $\alpha$ i $\beta$ są identycznymi formułami, to mówimy, że $\alpha$ jest niezmiennikiem programu **M**. Metoda badania częściowej poprawności i własności niezmienniczych programu jest jedną z najczęściej stosowanych metod analizy zachowania się programów (por. p. 8.1, 8.2).

LEMAT 3.10

Formuła $(\mathbf{M}\alpha)$ jest najsłabszym warunkiem wstępnym formuły $\alpha$ względem programu **M**.

DOWÓD

Niech dla pewnego wartościowania $v$ w **A**, **A**, $v \models (M\alpha)$. Zgodnie z przyjętą semantyką formuł algorytmicznych wynik $v'$ programu M jest określony, $v' = M_A(v)$ oraz **A**, $v' \models \alpha$. Zatem $M\alpha$ jest warunkiem wstępnym formuły $\alpha$ względem programu M.

Rozważmy dowolną formułę $\delta$ taką, że **A**, $v \models \delta$ implikuje istnienie wartościowania $v'$ spełniającego warunki $v' = M_A(v)$ *i* **A**, $v' \models \alpha$. Stąd **A**, $v \models \delta$ pociąga ze sobą **A**, $v \models (M\alpha)$, a więc dla dowolnego wartościowania $v$, **A**, $v \models (\delta \Rightarrow (M\alpha))$. Dowodzi to, że $(M\alpha)$ jest najsłabszym warunkiem wstępnym formuły $\alpha$ ze względu na program M. □

W dalszym ciągu będziemy posługiwać się następującymi oznaczeniami i definicjami. Powiemy, że dwa wektory **x**, **u** zmiennych są odpowiadającymi sobie wektorami, jeżeli są tej samej długości i ponadto, jeżeli $\mathbf{x} = x_1 \dots x_n$ i $\mathbf{y} = y_1 \dots y_n$, to $x_i$ ma ten sam typ co $y_i$ dla $i = 1, \dots, n$. Element wektora **x** będziemy oznaczać przez $x$. Jeżeli **x**, **y** są odpowiadającymi sobie wektorami a $x \in \mathbf{x}$, to odpowiadający mu element wektora **y** będziemy oznaczać (dla prostoty) przez $y$. Niech **x** będzie ciągiem wszystkich zmiennych występujących w formule $M\alpha$ i niech **y** będzie ciągiem różnych zmiennych, odpowiadającym ciągowi **x** takim, że $\mathbf{x} \cap \mathbf{y} = \emptyset$. Niech $\alpha(\mathbf{x}/\mathbf{y})$ oznacza formułę a $K(\mathbf{x}/\mathbf{y})$ program otrzymane z formuły $\alpha$ i programu K przez równoczesne zastąpienie wszystkich wystąpień zmiennych ze zbioru **x** przez odpowiadające im zmienne ze zbioru **y**. Przez $(\exists \mathbf{y})$ będziemy oznaczać ciąg kwantyfikatorów $(\exists y_1) \dots (\exists y_n)$.

LEMAT 3.11

Formuła $\beta = (\exists \mathbf{y})(\alpha(\mathbf{x}/\mathbf{y}) \wedge M(\mathbf{x}/\mathbf{y})(\mathbf{x} = \mathbf{y}))$ jest najmocniejszym następnikiem formuły $\alpha$ ze względu na program M.

DOWÓD

Rozważmy wartościowanie $v$ w **A** takie, że

$$\mathbf{A}, v \models (\alpha \wedge M\mathbf{true}) \tag{3.6}$$

Istnieje wtedy wartościowanie $v'$, wynik programu M, $v' = M_A(v)$.

Oznaczmy przez $v''$ wartościowanie powstające z wartościowania $v'$ przez zmianę wartości zmiennych **y**

$$v''(y) = v(x) \quad \text{dla} \quad y \in \mathbf{y}$$
$$v''(x) = v'(x) \quad \text{dla} \quad x \in \mathbf{x}$$

Z założenia mamy więc

$$\mathbf{A}, v'' \models \alpha(\mathbf{x}/\mathbf{y}) \quad \text{oraz} \quad \mathbf{A}, M(\mathbf{x}/\mathbf{y})_A(v'') \models \mathbf{x} = \mathbf{y}$$

Stąd $\mathbf{A}, v'' \models \alpha(\mathbf{x}/\mathbf{y}) \wedge \mathrm{M}(\mathbf{x}/\mathbf{y})(\mathbf{x} = \mathbf{y})$. Zgodnie z definicją znaczenia kwantyfikatora egzystencjalnego

$$\mathbf{A}, v' \models (\exists \mathbf{y})(\alpha(\mathbf{x}/\mathbf{y}) \wedge \mathrm{M}(\mathbf{x}/\mathbf{y})(\mathbf{x} = \mathbf{y}))$$

Ponieważ $v' = \mathrm{M}_{\mathbf{A}}(v)$, to korzystając z założenia (3.6) otrzymujemy ostatecznie $\mathbf{A}, v \models (\alpha \wedge \mathrm{M}\mathbf{true}) \Rightarrow \mathrm{M}\beta)$. Ponieważ przeprowadzone rozumowanie można powtórzyć dla dowolnego wartościowania $v$, więc tym samym wykazaliśmy, że $\beta$ jest następnikiem formuły $\alpha$,

$$\mathbf{A} \models ((\alpha \wedge \mathrm{M}\mathbf{true}) \Rightarrow \mathrm{M}\beta)$$

Załóżmy teraz, że $\delta$ jest innym niż $\beta$ następnikiem formuły $\alpha$ ze względu na program M, tzn.

$$\mathbf{A} \models ((\alpha \wedge \mathrm{M}\mathbf{true}) \Rightarrow \mathrm{M}\delta)$$

Pokażemy, że musi wtedy być $\mathbf{A} \models (\beta \Rightarrow \delta)$. Przypuśćmy przeciwnie, że dla pewnego wartościowania $v$

$$\mathbf{A}, v \models \beta \quad \text{oraz} \quad non\, \mathbf{A}, v \models \delta \tag{3.7}$$

Formuła $\beta$ gwarantuje nam istnienie danych spełniających warunek $\alpha$, przy których program M ma obliczenie skończone udane

$$\mathbf{A}, v \models \beta \quad \text{wtw} \quad (\exists v')\, \mathbf{A}, v' \models \alpha \quad \text{i} \quad v = \mathrm{M}_{\mathbf{A}}(v')$$

Stąd i z założenia (3.7) *non* $\mathbf{A}, v' \models \mathrm{M}\delta$ i równocześnie $\mathbf{A}, v' \models \alpha$ oraz $\mathbf{A}, v' \models \mathrm{M}\mathbf{true}$. Zatem *non* $\mathbf{A}, v' \models ((\alpha \wedge \mathrm{M}\mathbf{true}) \Rightarrow \mathrm{M}\delta)$, co przeczy założeniu, że $\delta$ jest następnikiem formuły $\alpha$. Tym samym wykazaliśmy, że $\beta$ jest najmocniejszym następnikiem formuły $\alpha$ ze względu na program M. □

Niezmiennikiem obliczeń programu nazwaliśmy w poprzednim punkcie (por. definicję 3.14) dowolną formułę, która, jeśli jest prawdziwa w wartościowaniu początkowym, jest również prawdziwa w każdym innym stanie obliczenia. Obecnie podejmiemy próbę charakteryzacji własności „być niezmiennikiem obliczeń programu" za pomocą formuł algorytmicznych.

Niech $inv_\alpha(\mathrm{M})$ będzie formułą zdefiniowaną rekurencyjnie ze względu na strukturę programu M następująco:

$$inv_\alpha(z := w) \stackrel{\mathrm{df}}{=} ((ok(w) \wedge \alpha) \Rightarrow (z := w)\alpha)$$

$$inv_\alpha(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}_1\ \mathbf{else}\ \mathrm{M}_2\ \mathbf{fi}) \stackrel{\mathrm{df}}{=} ((ok(\gamma) \wedge \alpha) \Rightarrow (\gamma \wedge inv_\alpha(\mathrm{M}_1) \vee \neg\gamma \wedge inv_\alpha(\mathrm{M}_2)))$$

$$inv_\alpha(\mathbf{begin}\ \mathrm{M}_1;\ \mathrm{M}_2\ \mathbf{end}) \stackrel{\mathrm{df}}{=} (\alpha \Rightarrow (inv_\alpha(\mathrm{M}_1) \wedge \mathrm{M}_1\, inv_\alpha(\mathrm{M}_2)))$$

$$inv_\alpha(\mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}) \stackrel{\mathrm{df}}{=} ((ok(\gamma) \wedge \alpha) \Rightarrow \bigcap\ \mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi}\ ((ok(\gamma) \wedge \gamma) \Rightarrow inv_\alpha(\mathrm{M})))$$

LEMAT 3.12

Dla dowolnej formuły $\alpha$, $\mathbf{A} \models inv_\alpha(\mathrm{M})$ wtedy i tylko wtedy, gdy $\alpha$ jest niezmiennikiem obliczeń programu M w strukturze **A**.

DOWÓD

Niech **A** będzie ustaloną strukturą, a $v$ dowolnym wartościowaniem w **A**. Udowodnimy, przez indukcję ze względu na stopień komplikacji programu, że $\mathbf{A}, v \models inv_\alpha(\mathrm{M})$ wtedy i tylko wtedy, gdy wszystkie stany obliczenia programu M spełniają $\alpha$.

Jeżeli M jest instrukcją przypisania postaci ($z := w$), to twierdzenie jest oczywiste, gdyż dowolne obliczenie M ma co najwyżej dwa stany.

Załóżmy (założenie indukcyjne), że dla dowolnego wartościowania $v$ mamy $\mathbf{A}, v \models inv_\alpha(\mathrm{K})$ wtedy i tylko wtedy, gdy wszyskie stany obliczenia programu K przy wartościowaniu $v$, spełniają formułę $\alpha$.

Rozważmy program M postaci **while** $\gamma$ **do** K **od** i niech $\mathrm{A}, v \models (\alpha \wedge ok(\gamma))$ oraz $\mathbf{A}, v \models \bigcap$ **if** $\gamma$ **then** K **fi** $((\gamma \wedge ok(\gamma)) \Rightarrow inv_\alpha(\mathrm{K}))$.
Wówczas dla każdego $i$

$$\mathbf{A}, v \models (\textbf{if } \gamma \textbf{ then } \mathrm{K} \textbf{ fi})^i ((\gamma \wedge ok(\gamma)) \Rightarrow inv_\alpha(\mathrm{K})) \tag{3.8}$$

W dalszym ciągu będziemy się starali wykazać, że wszystkie stany obliczenia programu **while** $\gamma$ **do** K **od** przy wartościowaniu początkowym $v$ spełniają formułę $\alpha$. Przypuśćmy, że ciąg wszystkich wartościowań występujących w tym obliczeniu ma następującą postać:

$$\theta = v_1, \theta_1, v_2, \theta_2, \ldots, v_n, \theta_n, v_{n+1}, \ldots, \tag{3.9}$$

gdzie $v_1 = v$ i $v_i, \theta_i, v_{i+1}$ oznacza ciąg wartościowań występujących w obliczeniu programu K, prowadzącym od $v_i$ do $v_{i+1}$ dla $i \leqslant n$. Załóżmy, że dla pewnego $n, \mathbf{A}, \mathrm{v}_n \models (\gamma \wedge ok(\gamma))$. Ponieważ $v_1, \theta_1, v_2, \theta_2, \ldots, \theta_{n-1}, v_n$ jest ciągiem wszystkich wartościowań występujących w obliczeniu programu (**if** $\gamma$ **then** K **fi**)$^{n-1}$, to na mocy założenia (3.8) mamy

$$\mathbf{A}, v_n \models inv_\alpha(\mathrm{K})$$

Na mocy założenia indukcyjnego dla programu K, jeżeli $v_n$ spełnia $\alpha$, to wszystkie stany obliczenia $v_n, \theta_n, v_{n+1}$ spełniają formułę $\alpha$. Stąd w szczególności $\mathbf{A}, v_{n+1} \models \alpha$.

Jeżeli $\mathbf{A}, v_n \models (\neg\gamma \vee \neg ok(\gamma))$, to obliczenie programu M jest skończone i albo nieudane, albo $v_n$ jest jego wynikiem. Standardowe rozumowanie indukcyjne doprowadza do wniosku, że wszystkie wartościowania występujące w obliczeniu $v_i$ programu M spełniają formułę $\alpha$.

Odwrotnie, załóżmy, że jeżeli wartościowanie początkowe $v$ spełnia formułę $\alpha$, to wszystkie stany obliczenia programu **while** $\gamma$ **do** M **od** spełniają

tę formułę. Niech ciąg (3.9) będzie tym obliczeniem. Jeżeli jest ono udane i wynikiem jego jest $v_{n+1}$, to dla wszystkich $i \leqslant n$ musi być $\mathbf{A}, v_i \models (\gamma \wedge ok(\gamma))$ oraz $\mathbf{A}, v_{n+1} \models \neg \gamma$. Wynika stąd, że zachodzi następujący warunek dla $i \leqslant n$

$$\mathbf{A}, v \models (\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{K}\ \mathbf{fi})^i((\gamma \wedge ok(\gamma)) \Rightarrow inv_\alpha(\mathrm{K}))$$

Ponieważ $(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{K}\ \mathbf{fi})^m(v) = v_{n+1}$ dla wszystkich $m > n+1$, zatem

$$\mathbf{A}, v \models (\alpha \wedge ok(\gamma)) \wedge \bigcap \mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{K}\ \mathbf{fi}((\gamma \wedge ok(\gamma)) \Rightarrow inv_\alpha(\mathrm{K})).$$

Rozumowanie przebiega podobnie, gdy obliczenie programu M jest nieskończone lub nieudane.

Przeprowadzenie analogicznych rozumowań w pozostałych przypadkach pozostawiamy czytelnikowi. □

Przyjmijmy następującą definicję:

$\perp\!\!\perp_\alpha(z := w) \stackrel{\mathrm{df}}{=} (\alpha \vee (z := w)\alpha)$
$\perp\!\!\perp_\alpha(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}_1\ \mathbf{else}\ \mathrm{M}_2\ \mathbf{fi}) \stackrel{\mathrm{df}}{=} (\alpha \vee ok(\gamma) \wedge (\gamma \wedge \perp\!\!\perp_\alpha(\mathrm{M}_1) \vee \neg \gamma \wedge \perp\!\!\perp_\alpha(\mathrm{M}_2)))$
$\perp\!\!\perp_\alpha(\mathbf{begin}\ \mathrm{M}_1;\ \mathrm{M}_2\ \mathbf{end}) \stackrel{\mathrm{df}}{=} \perp\!\!\perp_\alpha(\mathrm{M}_1) \vee \mathrm{M}_1(\perp\!\!\perp_\alpha(\mathrm{M}_2))$
$\perp\!\!\perp_\alpha(\mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}) \stackrel{\mathrm{df}}{=} (\alpha \vee (ok(\gamma) \wedge \bigcup \mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi}\ (\gamma \wedge \perp\!\!\perp_\alpha(\mathrm{M})))$

LEMAT 3.13

Dla dowolnej formuły $\alpha$ i dowolnego programu M, $\mathbf{A} \models \perp\!\!\perp_\alpha \mathrm{M}$ wtedy i tylko wtedy, gdy w każdym obliczeniu programu M istnieje stan spełniający formułę $\alpha$.

DOWÓD

Dla dowodu pokażemy, że dla dowolnego wartościowania $v$ w strukturze danych **A** zachodzi własność:

$$\mathbf{A}, v \models \perp\!\!\perp_\alpha \mathrm{M} \tag{3.10}$$

wtedy i tylko wtedy, gdy w obliczeniu programu M przy wartościowaniu początkowym $v$, istnieje stan spełniający formułę $\alpha$.

Załóżmy, że w wartościowaniu początkowym formuła $\alpha$ nie jest spełniona, gdyż w przeciwnym razie twierdzenie jest oczywiste. Dowód przeprowadzimy przez indukcję ze względu na stopień komplikacji programu M.

Jeżeli program M jest instrukcją przypisania $(x := \gamma)$, to wartość wyrażenia $\gamma_A(v)$ jest określona i uzyskane wartościowanie spełnia formułę $\alpha$ wtedy i tylko wtedy, gdy $\mathbf{A}, v \models (x := \gamma)\alpha$. W tym przypadku twierdzenie jest więc prawdziwe.

Załóżmy prawdziwość własności (3.10) dla programów prostszych niż M. Jeżeli M jest postaci **if** $\gamma$ **then** $\mathrm{M}_1$ **else** $\mathrm{M}_2$ **fi** lub **begin** $\mathrm{M}_1$; $\mathrm{M}_2$ **end**, to obliczenie programu M jest, w pierwszym przypadku, identyczne z oblicze-

niem programu $M_1$ lub $M_2$ w zależności od formuły $\gamma$, lub jest nieudane. Na mocy założenia indukcyjnego własność (3.10) jest więc prawdziwa. W drugim przypadku (tzn. w przypadku instrukcji złożonej) obliczenie programu M składa się z obliczenia programu $M_1$, po którym następuje obliczenie programu $M_2$. Wartościowanie, w którym jest spełniona formuła $\alpha$ musi więc wystąpić albo w obliczeniu $M_{1\mathbf{A}}(v)$, albo w obliczeniu $M_{2\mathbf{A}}(v')$, gdy $M_{1\mathbf{A}}(v)$ jest określone i $v' = M_{1\mathbf{A}}(v)$. Korzystając z założenia indukcyjnego otrzymujemy własność (3.10).

Rozważmy teraz przypadek, gdy M = **while** $\gamma$ **do** K **od**. Niezależnie od tego, czy obliczenie programu M jest skończone czy nie, kolejne wartościowania w obliczeniu programu M przy wartościowaniu początkowym $v$ są otrzymywane jako elementy obliczenia programu **if** $\gamma$ **then** K **fi**$^n$ dla pewnego $n$. W szczególności to wartościowanie, które spełnia formułę $\alpha$ w obliczeniu programu M jest elementem obliczenia programu **if** $\gamma$ **then** K **fi**$^n$ dla pewnego $n$. Na mocy poprzednich rozważań i założenia indukcyjnego mamy

w obliczeniu programu **while** $\gamma$ **do** K **od** przy wartościowaniu początkowym $v$ w A istnieje stan spełniający formułę $\alpha$

wtw istnieje takie $n \geqslant 1$, że w obliczeniu programu K przy wartościowaniu początkowym $v' =$ **if** $\gamma$ **then** K **fi**$^{n-1}{}_{\mathbf{A}}(v)$ istnieje stan spełniający formułę $\alpha$ oraz $\mathbf{A}, v' \models \gamma$

wtw $\mathbf{A}, v \models$ **if** $\gamma$ **then** K **fi**$^{n-1}(\gamma \wedge \amalg_\alpha K)$ dla pewnego $n \geqslant 1$

wtw $\mathbf{A}, v \models \bigcup$ **if** $\gamma$ **then** K **fi** $(\gamma \wedge \amalg_\alpha K)$.

W ten sposób udowodniliśmy własność (3.10), z której natychmiast wynika teza lematu. □

Formuły algorytmiczne umożliwiają także wyrażenie własności obliczeń nieco innego typu. Przykładem niech będą formuły

(**if** $\gamma$ **then** M **fi**)$^n \neg\gamma$ i (**if** $\gamma$ **then** M **fi**)$^n \gamma$

Prawdziwość pierwszej z nich w strukturze **A** oznacza, że długości wszystkich obliczeń programu **while** $\gamma$ **do** M **od** w **A** można oszacować z góry przez ($n \cdot const$). Prawdziwość drugiej formuły w strukturze **A** oznacza, że wszystkie obliczenia programu **while** $\gamma$ **do** M **od** w **A** można oszacować z dołu przez ($n \cdot const$), jeżeli długość obliczenia programu M daje się oszacować przez pewną stałą.

W poprzednim punkcie przedstawiliśmy propozycje pewnych definicji równoważności programów. W lematach 3.14 i 3.15 pokażemy, jak można wyrazić problem równoważności za pomocą formuł algorytmicznych. Proste dowody tych lematów pomijamy.

Niech **x** będzie zbiorem wszystkich zmiennych występujących w programach K i M, a **y** odpowiadającym mu ciągiem różnych zmiennych takim, że $\mathbf{x} \cap \mathbf{y} = \emptyset$.

LEMAT 3.14

Programy K i M są równoważne ze względu na zbiór zmiennych **z**, **z** ⊆ V(K) ∪ V(M) (por. definicję 3.16) w strukturze **A** wtw

$$\mathbf{A} \models (\mathbf{y} := \mathbf{x})\,\mathrm{K}(\mathrm{M}(\mathbf{x}/\mathbf{y})) \bigwedge_{x \in \mathbf{z} \cap V_i} (x = y) \wedge \bigwedge_{q \in \mathbf{z} \cap V_0} (\mathrm{K}q = \mathrm{M}q) \wedge (\mathrm{K}\,\mathbf{true} = \mathrm{M}\mathbf{true}).$$

■

LEMAT 3.15

Programy K i M są równoważne w strukturze **A** ze względu na zbiór formuł **Z** (por. definicję 3.17) wtedy i tylko wtedy gdy dla dowolnej formuły $\alpha \in Z$, $\mathbf{A} \models (\mathrm{M}\alpha = \mathrm{K}\alpha)$. ■

Formuły algorytmiczne pozwalają również wyrazić pewne własności struktur danych, w których są wykonywane obliczenia. Dla przykładu wymienimy tu dwie własności: być liczbą naturalną i być stosem skończonym.

Formuła $(x := 0)$ (**while** $\neg x = y$ **do** $x := x + 1$ **od true**) jest spełniona w strukturze liczb rzeczywistych przez wartościowanie $v$ wtedy i tylko wtedy, gdy $v(y)$ jest liczbą naturalną. Formuła

**while** $\neg$ *empty*$(x)$ **do** $x :=$ *pop*$(x)$ **od true**

jest spełniona w strukturze stosów przez wartościowanie $v$ wtedy i tylko wtedy, gdy $v(x)$ jest stosem skończonym. O innych własnościach struktur danych będzie mowa w rozdz. 5.

# Logika algorytmiczna 4

## Wprowadzenie 4.1

Rozdział ten jest poświęcony przedstawieniu formalnego systemu dedukcyjnego zwanego logiką algorytmiczną. Przedstawiona tu logika ma za zadanie umożliwić formalne dowodzenie semantycznych własności programów i własności struktur danych. W skrócie system ten oznaczać będziemy przez AL($\pi$), ponieważ będzie odnosić się do klasy $\pi$ deterministycznych programów iteracyjnych. W dalszych rozdziałach książki będziemy rozważać systemy algorytmiczne dla innych klas programów.

## Aksjomatyzacja 4.2

Zadanie aksjomatyzacji polega na podaniu takiego zbioru formuł, zwanych aksjomatami, i takiego zbioru reguł wnioskowania, które pozwolą wyprowadzić wszystkie formuły prawdziwe w przyjętej semantyce. Z tego względu aksjomaty nie mogą być zupełnie dowolnymi formułami. Same muszą być formułami ogólnie prawdziwymi. Co więcej, reguły wnioskowania muszą zachowywać prawdziwość formuł, a więc muszą prowadzić od formuł prawdziwych do formuł prawdziwych. Proces wywodzenia formuły ze zbioru aksjomatów, za pomocą przyjętych reguł dedukcji, nazywa się dowodem formalnym.

Badania prawdziwości formuły metodą semantyczną, tzn. przez wyliczanie jej wartości, jest niejednokrotnie skomplikowane i na ogół żmudne. Metoda aksjomatyczna, wyprowadzania prawdziwości formuły z przyjętych założeń (tzn. z aksjomatów), pozwala czasami znakomicie uprościć ten proces.

DEFINICJA 4.1

Regułą wnioskowania nazywamy parę postaci $(X, \beta)$, gdzie $X$ jest zbiorem

formuł zwanych przesłankami, a $\beta$ formułą zwaną wnioskiem. Regułę wnioskowania $(X, \beta)$ tradycyjnie zapisujemy w postaci

$$\frac{X}{\beta}$$ ■

Jeżeli udowodnimy wszystkie przesłanki pewnej reguły, to zastosowanie reguły polega na uznaniu wniosku w tej regule za udowodniony.

Reguły wnioskowania będą przedstawiane zwykle w postaci schematów. Na przykład następujący schemat jest regułą wnioskowania dla dowolnych formuł $\alpha$, $\gamma$ i dowolnych programów K i M:

$$\frac{\mathrm{K}\alpha, \mathrm{M}\alpha, \mathrm{ok}(\gamma)}{\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{K}\ \mathbf{else}\ \mathrm{M}\ \mathbf{fi}\ \alpha}$$

Nieformalnie, sens tej reguły jest następujący. Jeżeli w pewnej strukturze formuła $\alpha$ jest zawsze spełniona zarówno po wykonaniu programu K, jak i po wykonaniu programu M, i jeżeli obliczenie wartości testu $\gamma$ jest poprawne, to po wykonaniu programu **if** $\gamma$ **then** K **else** M **fi** jest spełniona formuła $\alpha$.

Powiemy, że został określony formalny *system dedukcyjny*, jeżeli jest zdefiniowany język systemu, zbiór aksjomatów i zbiór reguł wnioskowania. Zbiory aksjomatów i reguł wnioskowania wyznaczają pojęcie dowodu formalnego w rozważanym systemie dedukcyjnym.

DEFINICJA 4.2

Logiką algorytmiczną deterministycznych programów iteracyjnych będziemy nazywać system AL($\pi$) zdeterminowany przez język algorytmiczny L($\pi$) (por. p. 3.2) oraz zbiór aksjomatów Ax i reguł wnioskowania Rw, których schematy przedstawiamy

AKSJOMATY

Ax1 $((\alpha \Rightarrow \beta) \Rightarrow ((\beta \Rightarrow \delta) \Rightarrow (\alpha \Rightarrow \delta)))$

Ax2 $(\alpha \Rightarrow (\alpha \vee \beta))$

Ax3 $(\beta \Rightarrow (\alpha \vee \beta))$

Ax4 $((\alpha \Rightarrow \delta) \Rightarrow ((\beta \Rightarrow \delta) \Rightarrow ((\alpha \vee \beta) \Rightarrow \delta)))$

Ax5 $((\alpha \wedge \beta) \Rightarrow \alpha)$

Ax6 $((\alpha \wedge \beta) \Rightarrow \beta)$

Ax7 $((\delta \Rightarrow \alpha) \Rightarrow ((\delta \Rightarrow \beta) \Rightarrow (\delta \Rightarrow (\alpha \wedge \beta))))$

Ax8 $((\alpha \Rightarrow (\beta \Rightarrow \delta)) \equiv ((\alpha \wedge \beta) \Rightarrow \delta))$

Ax9 $((\alpha \wedge \neg \alpha) \Rightarrow \beta)$

Ax10 $((\alpha \Rightarrow (\alpha \wedge \neg \alpha)) \Rightarrow \neg \alpha)$

Ax11 $(\alpha \vee \neg \alpha)$

Ax12 $ok(\tau) \Rightarrow ((\forall x)\alpha(x) \Rightarrow (x := \tau)\alpha(x))$

dla dowolnego termu $\tau$ tego samego typu co zmienna indywiduowa $x$

Ax13 $(\forall x)\alpha(x) \equiv \neg(\exists x)\neg\alpha(x)$

Ax14 $\mathrm{K}((\exists x)\alpha(x)) \equiv (\exists y)(\mathrm{K}\,\alpha(x/y))$ dla $y \in V - V(K)$

Ax15 $\mathrm{K}(\alpha \vee \beta) \equiv ((\mathrm{K}\alpha) \vee (\mathrm{K}\beta))$

Ax16 $\mathrm{K}(\alpha \wedge \beta) \equiv ((\mathrm{K}\alpha) \wedge (\mathrm{K}\beta))$

Ax17 $\mathrm{K}(\neg \alpha) \Rightarrow \neg(\mathrm{K}\alpha)$

Ax18 $(z := w)\gamma \equiv (\gamma(z/w) \wedge ok(w))$

Ax19 $\mathbf{begin}\ \mathrm{K};\ \mathrm{M}\ \mathbf{end}\ \alpha \equiv \mathrm{K}(\mathrm{M}\alpha)$

Ax20 $\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{K}\ \mathbf{else}\ \mathrm{M}\ \mathbf{fi}\ \alpha \equiv (ok(\gamma) \wedge ((\neg\gamma \wedge \mathrm{K}\alpha) \vee (\gamma \wedge \mathrm{M}\alpha)))$

Ax21 $\mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{K}\ \mathbf{od}\ \alpha \equiv$

$\equiv (ok(\gamma) \wedge ((\neg\gamma \wedge \alpha) \vee (\gamma \wedge \mathrm{K}(\mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{K}\ \mathbf{od}\ \alpha))))$

Ax22 $\bigcup \mathrm{K}\alpha \equiv (\alpha \vee \mathrm{K}(\bigcup \mathrm{K}\alpha))$

Ax23 $\bigcap \mathrm{K}\alpha \equiv (\alpha \wedge \mathrm{K}(\bigcap \mathrm{K}\alpha))$

REGUŁY WNIOSKOWANIA

R1 $\dfrac{\alpha, (\alpha \Rightarrow \beta)}{\beta}$

R2 $\dfrac{(\alpha \Rightarrow \beta)}{(\mathrm{K}\alpha \Rightarrow \mathrm{K}\beta)}$

R3 $\dfrac{\{\mathrm{M}(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{K}\ \mathbf{fi})^i(\neg\gamma \wedge ok(\gamma) \wedge \alpha) \Rightarrow \beta\}_{i \in \mathrm{N}}}{(\mathrm{M}(\mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{K}\ \mathbf{od}\ \alpha) \Rightarrow \beta)}$

R4 $\dfrac{\{\mathrm{M}(\mathrm{K}^i\alpha) \Rightarrow \beta\}_{i \in \mathrm{N}}}{(\mathrm{M}(\bigcup \mathrm{K}\alpha) \Rightarrow \beta)}$

R5 $\dfrac{\{\alpha \Rightarrow \mathrm{M}(\mathrm{K}^i\beta)\}_{i \in \mathrm{N}}}{(\alpha \Rightarrow \mathrm{M}(\bigcap \mathrm{K}\beta))}$

R6 $\dfrac{(\alpha(x) \Rightarrow \beta)}{((\exists x)\alpha(x) \Rightarrow \beta)}$

R7 $\dfrac{(\beta \Rightarrow \alpha(x))}{(\beta \Rightarrow (\forall x)\alpha(x))}$

W regułach R6 i R7, zwanych regułami wprowadzania kwantyfikatora szczegółowego i ogólnego, odpowiednio do poprzednika i do następnika

implikacji, należy założyć, że $x$ nie jest zmienną wolną w $\beta$. Reguły R4 i R5 są algorytmicznymi odpowiednikami reguł R6 i R7; umożliwiają wprowadzenie kwantyfikatorów iteracji do implikacji. Jednak charakter tych reguł jest całkowicie inny, ponieważ zbiory ich przesłanek są nieskończone. Podobny charakter ma reguła R3 wprowadzania instrukcji **while** w poprzedniku implikacji. Reguły takie będziemy nazywać $\omega$-regułami. Reguła R1 jest zwana regułą odrywania lub modus ponens (m.p.).

We wszystkich przedstawionych schematach aksjomatów i reguł wnioskowania $\alpha$, $\beta$, $\delta$ są dowolnymi formułami, $\gamma$ i $\gamma'$ są dowolnymi formułami otwartymi, $\tau$ jest termem, a K i M są dowolnymi programami. ■

System dedukcyjny wyznaczony przez

(1) język L ograniczony do zmiennych zdaniowych i spójników logicznych $\wedge$, $\vee$, $\neg$, $\Rightarrow$,

(2) aksjomaty Ax1–Ax11 oraz regułę R1

nazywamy *klasycznym rachunkiem zdań* [44, 33].

System dedukcyjny wyznaczony przez

(1) język pierwszego rzędu L oraz

(2) zbiór aksjomatów Ax1–Ax13 i reguły wnioskowania R1, R6, R7 (rozważane tylko dla formuł języka L)

nazywamy *klasycznym rachunkiem predykatów* lub logiką klasyczną [33, 41, 44].

Logika algorytmiczna jest rozszerzeniem klasycznego rachunku predykatów o pewne aksjomaty i reguły charakteryzujące nowe typy operatorów. Zauważmy, że reguły wnioskowania R1, R6, R7, a więc wszystkie reguły logiki klasycznej, mają skończoną liczbę przesłanek. W logice algorytmicznej występują natomiast reguły o nieskończonej liczbie przesłanek. Zwróćmy uwagę na konsekwencje przyjęcia $\omega$-reguł w procesie dowodzenia.

DEFINICJA 4.3

Dowodem formuły $\alpha$ ze zbioru formuł $Z$ będziemy nazywać parę $\langle D, d \rangle$, w której $D$ jest zbiorem skończonych ciągów liczb naturalnych, uporządkowanym przez relację „być segmentem początkowym", a $d$ funkcją ze zbioru $D$ w zbiór $F(\pi)$ taką, że

(1) każdy liniowo uporządkowany podzbiór zbioru $D$ jest skończony,

(2) jeżeli $c = (i_1, \ldots, i_n) \in D$, to $d(c) \in AX \cup Z$ wtedy i tylko wtedy, gdy nie istnieje takie $j \in N$, że $(i_1, \ldots, i_n, j) \in D$,

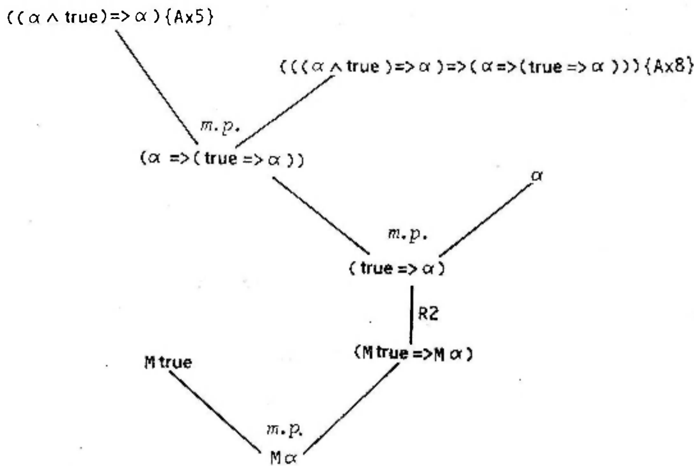
(3) jeżeli $(i_1, \ldots, i_n, j) \in D$ oraz $d((i_1 \ldots, i_n, j)) = \alpha_j$, $j \in J \subset N$, to $c = (i_1, \ldots, i_n) \in D$ oraz $d(c)$ jest wnioskiem w tej regule wnioskowania, w której przesłankami są formuły $\alpha_j$, $j \in J$,

(4) ciąg pusty $\emptyset$ należy do $D$ i $d(\emptyset) = \alpha$. ■

Zauważmy, że zbiór $D$ w definicji 4.3 tworzy pewne drzewo. Wierzchołkami drzewa są elementy zbioru $D$. Wszystkie wierzchołki w drzewie, które są ciągami $n$-elementowymi, tworzą $n$-ty poziom drzewa. Wierzchołki $c$ i $c'$ są połączone krawędzią wtedy i tylko wtedy, gdy $c$ jest wierzchołkiem na poziomie $n$-tym postaci $(i_1, ..., i_n)$, a $c'$ jest wierzchołkiem na poziomie $n+1$-szym postaci $(i_1, ..., i_n, j)$ dla pewnych $i_1, ..., i_n$, $j \in N$. Wierzchołek $c'$ nazywamy *następnikiem* lub synem wierzchołka $c$. Wierzchołki, które nie mają następników, nazywamy *liśćmi drzewa D*. *Korzeniem drzewa* nazywamy wierzchołek drzewa $D$, który nie jest synem żadnego innego wierzchołka z $D$. Drzewo $D$ etykietowane formułami zgodnie z definicją 4.3 będziemy nazywać *drzewem dowodu* formuły.

PRZYKŁAD 4.1

Niech $Z = \{\alpha, \mathbf{true}\}$, gdzie $\alpha$ jest dowolnie ustaloną formułą a M programem. Rysunek 4.1 przedstawia drzewo dowodu formuły $\alpha$ ze zbioru $Z$. □




RYS. 4.1

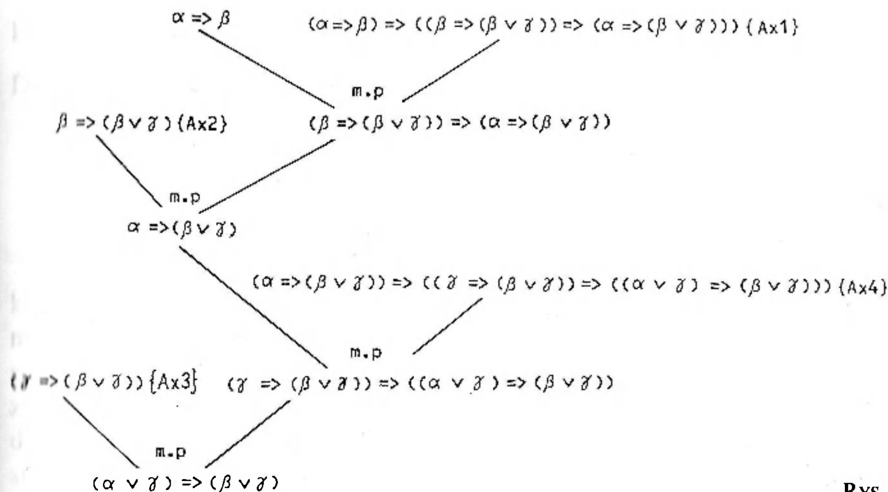
PRZYKŁAD 4.2

Niech $\alpha$, $\beta$, $\gamma$ będą dowolnymi formułami. Na rysunku 4.2 przedstawiono formalny dowód formuły

$$((\alpha \vee \gamma) \Rightarrow (\beta \vee \gamma))$$

przy założeniu, że formuła $(\alpha \Rightarrow \beta)$ ma dowód. □

Jeżeli reguły, które stosujemy do dowodu konkretnej formuły, mają skończoną ilość przesłanek, to drzewo dowodu jest skończone — zawiera tylko skończoną ilość wierzchołków. W logice klasycznej dowód jest więc




Rys. 4 2

pojęciem skończonym. Jako natychmiastowy wniosek otrzymujemy własność finitystyczności procesu dedukcyjnego w logice klasycznej. Jeżeli formuła $\alpha$ ma dowód ze zbioru założeń $Z$, to $\alpha$ ma też dowód z pewnego skończonego podzbioru $Z_0$ zbioru $Z$. Mianowicie, $Z_0$ składa się z tych tylko formuł zbioru $Z$, które występują w dowodzie formalnym formuły $\alpha$. Każde twierdzenie logiki klasycznej jest konsekwencją skończonego zbioru założeń.

Niestety, w logice algorytmicznej sprawa jest o wiele bardziej skomplikowana. Dowód formalny nie zawsze jest skończony. Jeżeli chociaż raz użyliśmy w dowodzie reguły $\omega$, to szerokość drzewa dowodu jest nieskończona. Co więcej, chociaż wszystkie gałęzie (drogi) w drzewie dowodu są skończone, to może się zdarzyć, że nie ma wspólnego ograniczenia ich długości. Oznacza to, że poziomów w drzewie dowodu formuły też może być nieskończenie wiele. Sytuację tę ilustruje następujący przykład.

PRZYKŁAD 4.3

Niech $Z$ będzie nieskończonym zbiorem formuł

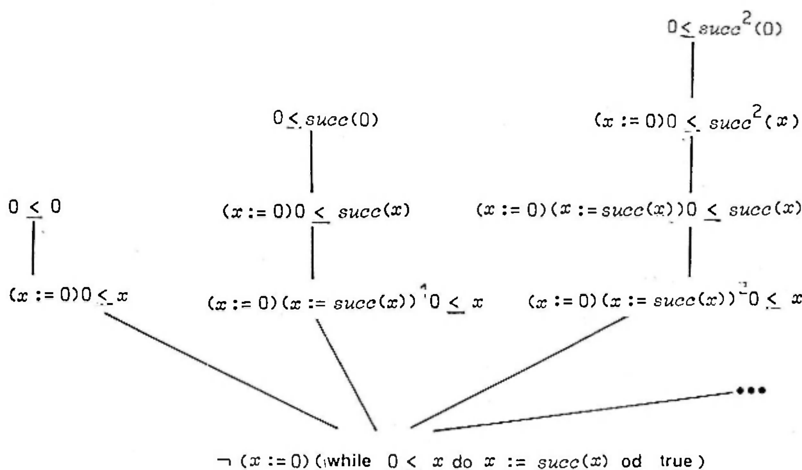
$$Z = \{0 \leqslant 0,\ 0 \leqslant succ(0),\ 0 \leqslant succ(succ(0)), \ldots, 0 \leqslant succ^i(0), \ldots\}$$

i niech $\alpha$ będzie formułą postaci

$$\neg\ \mathbf{begin}\ x := 0;\ \mathbf{while}\ 0 \leqslant x\ \mathbf{do}\ x := succ(x)\ \mathbf{od}\ \mathbf{end}\ \mathbf{true}$$

Formalny dowód formuły $\alpha$ ze zbioru $Z$ przedstawiono na rys. 4.3. Zauważmy, że wszystkie gałęzie drzewa dowodu są skończone, ale dla każdej liczby naturalnej $n$, istnieje w tym drzewie gałąź o długości $n$. □

Sprawdzenie, czy dane skończone drzewo etykietowane formułami jest dowodem formalnym pewnej formuły, jest zadaniem dość prostym. Zarówno




Rys. 4.3

zbiór aksjomatów, jak i reguł wnioskowania zawierają skończoną liczbę schematów. Postępując zgodnie z definicją dowodu (por. definicję 4.3) możemy wykazać, że dany zbiór jest lub nie jest dowodem pewnej formuły. Zupełnie innym jednak zagadnieniem jest zbudowanie dowodu pewnej formuły, nawet przy założeniu, że ten dowód istnieje. Z samej postaci formuły trudno wywnioskować (lub jest to zupełnie niemożliwe) jaka jest struktura drzewa dowodu, jakie aksjomaty są w tym przypadku istotne, czy nawet, jaka reguła posłużyła do wywnioskowania danej formuły. Ponadto, jedna formuła może mieć kilka całkowicie różnych dowodów. Proces tworzenia dowodu jest procesem twórczym, wymagającym od autora wiele pomysłowości.

System formalny, który przedstawiliśmy w tym punkcie tzw. system Hilberta, nie nadaje się do automatyzacji procesu dowodzenia. Istnieją jednak inne systemy formalne logiki algorytmicznej, które pozwalają na mechaniczne, przynajmniej w pewnym stopniu, tworzenie dowodu. Takim systemem jest aksjomatyzacja typu gentzenowskiego, którą przedstawimy w dalszej części rozdziału.

### Definicja 4.4

Jeżeli formuła $\alpha$ ma dowód ze zbioru założeń $Z$, to piszemy $Z \vdash \alpha$. Jeżeli $\alpha$ ma dowód wykorzystujący jedynie aksjomaty systemu, to piszemy $\vdash \alpha$ i formułę $\alpha$ nazywamy twierdzeniem logiki algorytmicznej. Funkcję $C: 2^F \to 2^F$ taką, że dla dowolnego zbioru formuł $Z$, $C(Z)$ jest najmniejszym zbiorem zawierającym $Ax \cup Z$ i zamkniętym ze względu na reguły wnioskowania ze zbioru Rw, nazywamy operacją syntaktycznej konsekwencji. Jeżeli $\alpha \in C(Z)$, to powiemy, że $\alpha$ jest syntaktyczną konsekwencją zbioru $Z$. ■

Z przyjętych definicji wynika wprost następująca charakteryzacja operacji syntaktycznej konsekwencji.

LEMAT 4.1

Dla dowolnych zbiorów formuł $Z$, $Z_1$, $Z_2$ zachodzą następujące własności:

(1) jeżeli $Z_1 \subset Z_2$, to $C(Z_1) \subset C(Z_2)$;

(2) $Z \subset C(Z)$;

(3) $C(C(Z)) = C(Z)$;

(4) dla dowolnej formuły $\alpha$, $Z \vdash \alpha$ wtw $\alpha \in C(Z)$.

Inaczej mówiąc, jeżeli formuła $\alpha$ jest konsekwencją zbioru $Z$, to $\alpha$ jest konsekwencją każdego zbioru większego niż $Z$ (własność tę nazywamy monotonicznością operacji konsekwencji). Każda formuła należąca do zbioru $Z$ należy również do konsekwencji tego zbioru. Ponadto, konsekwencje zbioru konsekwencji $C(Z)$ są już zawarte w zbiorze $C(Z)$ (tzn. zbiór $C(Z)$ jest domknięty ze względu na swoje konsekwencje). Ostatnia własność (4) stwierdza równoważność pojęć „być konsekwencją zbioru $Z$"" i mieć dowód formalny oparty na $Z$".

Ważnym i wielokrotnie w dalszym ciągu używanym, będzie następujący, natychmiastowy wniosek z lematu 4.1. ■

LEMAT 4.2

(1) Jeżeli $\alpha$ jest twierdzeniem rachunku zdań, to jest twierdzeniem logiki algorytmicznej.

(2) Jeżeli $\alpha$ jest twierdzeniem logiki klasycznej, to jest twierdzeniem logiki algorytmicznej. ■

DEFINICJA 4.5

Jeżeli dla dowolnego zbioru formuł $Z$ z tego, że dla wszystkich $i \in I$, $I \subset N$, $\alpha_i \in C(Z)$ wynika, że $\beta \in C(Z)$, to schemat

$$\frac{\alpha_i, i \in I, I \subset N}{\beta}$$

będziemy nazywać wtórną regułą wnioskowania. ■

Przykładem wtórnej reguły wnioskowania jest więc schemat

$$\frac{\alpha \Rightarrow \beta}{(\alpha \vee \delta) \Rightarrow (\beta \vee \delta)}$$

(por. przykład 4.2). Dalej przedstawimy inne przykłady reguł wtórnych. Reguły te pozwolą w dużym stopniu uprościć formalne dowody twierdzeń.

PRZYKŁAD 4.4

Dla dowolnego $i \in N$ oraz dowolnych: formuły otwartej $\gamma$, formuły $\alpha$ i pro-

gramu M, następująca formuła jest twierdzeniem logiki algorytmicznej AL$(\pi)$:

$$(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^i(\neg\gamma \wedge ok(\gamma) \wedge \alpha) \Rightarrow \mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha \qquad (4.1)$$

Dowód przebiega przez indukcję ze względu na $i$. Dla $i = 0$ mamy

$$\vdash (\neg\gamma \wedge ok(\gamma) \wedge \alpha) \Rightarrow \mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha$$

na mocy aksjomatów Ax21 i Ax3. Przyjmijmy $\beta = (\neg\gamma \wedge ok(\gamma) \wedge \alpha)$ i załóżmy (założenie indukcyjne), że

$$\vdash (\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^k\beta \Rightarrow \mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha$$

Na mocy aksjomatu Ax20

$$\vdash (\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^{k+1}\beta \Rightarrow ok(\gamma) \wedge \big((\gamma \wedge \mathrm{M}(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^k\beta) \vee (\neg\gamma \wedge (\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^k\beta)\big)$$

Stosując do założenia indukcyjnego regułę R2 oraz korzystając z aksjomatów Ax1–Ax4 (por. przykład 4.14) otrzymamy

$$\vdash (\gamma \wedge ok(\gamma) \wedge \mathrm{M}(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^k\beta) \Rightarrow (\gamma \wedge ok(\gamma) \wedge \mathrm{M}(\mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha))$$

Ponadto, na mocy Ax20

$$\vdash (\neg\gamma \wedge ok(\gamma) \wedge (\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^k\beta) \Rightarrow (\neg\gamma \wedge ok(\gamma) \wedge \alpha)$$

Zatem na mocy aksjomatów Ax1, Ax5–Ax9 (por. przykład 4.2)

$$\vdash (\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^{k+1}\beta \Rightarrow (ok(\gamma) \wedge (\gamma \wedge \mathrm{M}(\mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha) \vee \neg\gamma \wedge \alpha))$$

Czyli ostatecznie na mocy aksjomatu Ax21

$$\vdash (\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^{k+1}\beta \Rightarrow \mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha$$

Na mocy zasady indukcji matematycznej udowodniliśmy, że formuła postaci (4.1) ma dowód dla dowolnej liczby naturalnej $i$. □

PRZYKŁAD 4.5

Dla dowolnego zbioru formuł $Z$, zbiór $C(Z)$ jest zamknięty ze względu na następującą regułę wtórną zwaną regułą niezmiennika:

$$\frac{(\alpha \Rightarrow \mathrm{M}\alpha)}{((\alpha \wedge \mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \mathbf{true}) \Rightarrow \mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha)}$$

gdzie $\alpha$ jest dowolną formułą, M dowolnym programem, a $\gamma$ dowolną formułą otwartą.

Pokażemy, że jeżeli przesłanka tej reguły jest dowodliwa w AL, to wniosek też ma dowód. Reguła ta jest wygodnym narzędziem do dowodzenia własności programów iteracyjnych.

Niech

$$Z \vdash (\alpha \Rightarrow M\alpha) \tag{4.2}$$

Pokażemy przez indukcję ze względu na $i$, że formuła

$$\big((\alpha \wedge \textbf{if } \gamma \textbf{ then } M \textbf{ fi}^i(\neg\gamma \wedge ok(\gamma))) \Rightarrow \textbf{while } \gamma \textbf{ do } M \textbf{ od } \alpha\big) \tag{4.3}$$

ma dowód formalny ze zbioru $Z$ w logice algorytmicznej. Oznaczmy przez IF program **if** $\gamma$ **then** M **fi** i przez WH program **while** $\gamma$ **do** M **od**. Na mocy aksjomatów Ax2, Ax21 i reguły modus ponens

$$\vdash \big((\alpha \wedge ok(\gamma) \wedge \neg\gamma) \Rightarrow \mathrm{WH}\alpha\big)$$

Zatem dla $i = 0$ formuła (4.3) jest udowodniona.

Załóżmy (założenie indukcyjne), że dla pewnego $j$

$$Z \vdash \big((\alpha \wedge \mathrm{IF}^j(\neg\gamma \wedge ok(\gamma))) \Rightarrow \mathrm{WH}\alpha\big)$$

Na mocy aksjomatu Ax20

$$\vdash \mathrm{IF}^{j+1}(\neg\gamma \wedge ok(\gamma)) \Rightarrow \big(ok(\gamma) \wedge \big(\gamma \wedge M\big(\mathrm{IF}^j(\neg\gamma \wedge ok(\gamma))\big) \vee \\ \vee \neg\gamma \wedge \big(\mathrm{IF}^j(\neg\gamma \wedge ok(\gamma))\big)\big)\big)$$

a na mocy Ax5 i założenia (4.2)

$$Z \vdash \big(\alpha \wedge \mathrm{IF}^{j+1}(\neg\gamma \wedge ok(\gamma))\big) \Rightarrow \\ \Rightarrow \big(ok(\gamma) \wedge \big((\gamma \wedge M\alpha \wedge M\big(\mathrm{IF}^j(\neg\gamma \wedge ok(\gamma))\big)\big) \vee (-\gamma \wedge \alpha)\big)\big)$$

Stąd i z aksjomatu Ax16, założenia indukcyjnego oraz udowodnionej w poprzednim przykładzie formuły (4.1) otrzymujemy

$$Z \vdash \big(\alpha \wedge \mathrm{IF}^{j+1}(\neg\gamma \wedge ok(\gamma))\big) \Rightarrow \\ \Rightarrow \big((\gamma \wedge ok(\gamma) \wedge M(\mathrm{WH}\alpha)) \vee (\alpha \wedge ok(\gamma) \wedge \neg\gamma)\big)$$

Na mocy aksjomatu Ax21 mamy

$$Z \vdash \big(\alpha \wedge \mathrm{IF}^{j+1}(\neg\gamma \wedge ok(\gamma))\big) \Rightarrow \mathrm{WH}\alpha$$

a zatem formuła (4.3) została udowodniona dla dowolnej liczby naturalnej $i$.

Zastosowanie aksjomatu Ax8 pozwoli nam przekształcić formułę (4.3) do postaci wymaganej w $\omega$-regule R3, po zastosowaniu której otrzymujemy

$$Z \vdash (\mathrm{WH}\ \textbf{true} \Rightarrow (\alpha \Rightarrow \mathrm{WH}\,\alpha))$$

Stosując jeszcze raz aksjomat Ax8 otrzymamy

$$Z \vdash ((\alpha \wedge \textbf{while } \gamma \textbf{ do } M \textbf{ od true}) \Rightarrow \textbf{while } \gamma \textbf{ do } M \textbf{ od } \alpha)$$

co należało udowodnić. □

PRZYKŁAD 4.6

Niech $\alpha$, $\beta$ będą formułami otwartymi, wtedy dla dowolnego programu K następujący schemat jest wtórną regułą logiki algorytmicznej:

$$\frac{(\alpha \Rightarrow \beta),\ (ok(\beta) \Rightarrow ok(\alpha))}{\textbf{while}\ \beta\ \textbf{do}\ K\ \textbf{od true} \Rightarrow \textbf{while}\ \alpha\ \textbf{do}\ K\ \textbf{od true}}$$

DOWÓD

Pokażemy, że dla dowolnego zbioru formuł $Z$ i dowolnych $\alpha$, $\beta$, K, jeżeli $Z \vdash (\alpha \Rightarrow \beta)$ oraz $Z \vdash (ok(\beta) \Rightarrow ok(\alpha))$, to wniosek reguły ma dowód ze zbioru $Z$.

$Z \vdash ((ok(\beta) \Rightarrow ok(\alpha))$ {założenie}

$Z \vdash (\neg \beta \Rightarrow \neg \alpha)$ {rachunek zdań, założenie}

$Z \vdash ((\neg \beta \wedge ok(\beta)) \Rightarrow \textbf{while}\ \alpha\ \textbf{do}\ K\ \textbf{od true})$ {Ax2 i Ax21}

Załóżmy (założenie indukcyjne), że dla pewnego $i$

$Z \vdash \textbf{if}\ \beta\ \textbf{then}\ K\ \textbf{fi}^i(\neg \beta \wedge ok(\beta)) \Rightarrow \textbf{while}\ \alpha\ \textbf{do}\ K\ \textbf{od true}$

Niech IF $\stackrel{\text{df}}{=}$ **if** $\beta$ **then** K **fi** oraz WH $\stackrel{\text{df}}{=}$ **while** $\alpha$ **do** K **od** wtedy

$\vdash \textbf{if}\ \beta\ \textbf{then}\ K\ \textbf{fi}^{i+1}(\neg \beta \wedge ok(\beta)) \Rightarrow$

$\Rightarrow (ok(\beta) \wedge (\neg \beta \wedge IF^i(\neg \beta \wedge ok(\beta)) \vee \beta \wedge K(IF^i(\neg \beta \wedge ok(\beta)))))$ {Ax20}

$\vdash (\neg \beta \wedge ok(\beta) \wedge IF^i(\neg \beta \wedge ok(\beta)) \vee \beta \wedge ok(\beta) \wedge K(IF^i(\neg \beta \wedge ok(\beta)))) \Rightarrow$

$\Rightarrow (\neg \beta \wedge ok(\beta) \vee \beta \wedge ok(\beta) \wedge K(IF^i(\neg \beta \wedge ok(\beta))))$ {Ax5 i reguła wtórna por. przykład 4.2}

$Z \vdash (\neg \beta \wedge ok(\beta) \vee \beta \wedge ok(\beta) \wedge K(IF^i(\neg \beta \wedge ok(\beta)))) \Rightarrow$

$\Rightarrow (\neg \alpha \wedge ok(\alpha) \vee \beta \wedge ok(\beta) \wedge K(WH\ \textbf{true}))$ {z założenia indukcyjnego, reguły R2 i reguły wtórnej przedstawionej w p. 4.7 na rys. 4.5}

Ale

$\vdash (\neg \alpha \vee \beta \wedge K(WH\ \textbf{true})) \equiv (\neg \alpha \vee \neg \alpha \wedge \beta \wedge K(WH\ \textbf{true}) \vee \alpha \wedge \beta \wedge$
$\wedge K(WH\ \textbf{true}))$

skąd na mocy praw rachunku zdań mamy

$Z \vdash (ok(\beta) \wedge (\neg \beta \vee \beta \wedge K(IF^i(\neg \beta \wedge ok(\beta))))) \Rightarrow$
$\Rightarrow (ok(\alpha) \wedge (\neg \alpha \vee \alpha \wedge K(WH\ \textbf{true})))$

$Z \vdash (ok(\alpha) \wedge (\neg \alpha \vee \alpha \wedge K(WH\ \textbf{true}))) \Rightarrow WH\ \textbf{true}$ {Ax21}

Z przedstawionego ciągu twierdzeń na mocy aksjomatu Ax1 i reguły modus ponens otrzymujemy

$Z \vdash$ **if** $\beta$ **then** K **fi**$^{i+1}(\neg\beta \wedge ok(\beta)) \Rightarrow$ **while** $\alpha$ **do** K **od true**

Zasada indukcji pozwala więc wywnioskować, że dla dowolnego $i$

$Z \vdash$ **if** $\beta$ **then** K **fi**$^{i}(\neg\beta \wedge ok(\beta)) \Rightarrow$ **while** $\alpha$ **do** K **od true**

Stąd na mocy reguły R3 mamy ostatecznie

$Z \vdash$ **while** $\beta$ **do** K **od true** $\Rightarrow$ **while** $\alpha$ **do** K **od true** □

# Twierdzenie o pełności logiki algorytmicznej 4.3

W tym punkcie uzasadnimy wybór aksjomatów i reguł wnioskowania. Pokażemy, że system formalny logiki algorytmicznej jest zgodny z przyjętą wcześniej semantyką. Nie pozwala on udowodnić formuły fałszywej, czyli wszystkie formuły, które mają dowód, są prawdziwe. Co więcej, wszystkie formuły prawdziwe mają w tym systemie dowód formalny.

LEMAT 4.3

Wszystkie aksjomaty systemu AL(π) są tautologiami.

DOWÓD

(1) Rozważmy aksjomat Ax18 w przypadku, gdy $\tau$ jest termem, $\gamma$ jest formułą otwartą, a $x$ jest zmienną indywiduową

$$(x := \tau)\gamma(x) \equiv (\gamma(x/\tau) \wedge ok(\tau))$$

Niech **A** będzie ustaloną strukturą danych, a $v$ wartościowaniem. Przypomnijmy najpierw, że formuła $ok(\tau)$ jest spełniona przez wartościowanie, jeżeli tylko należy ono do dziedziny funkcji $\tau_{\mathbf{A}}$ (por. p. 2.4). Mamy zatem

$$\mathbf{A}, v \models (\gamma(x/\tau) \wedge ok(\tau)) \quad \text{wtw} \quad v \in Dom(\tau) \quad \text{oraz} \quad \mathbf{A}, v \models \gamma(x/\tau)$$

Ponieważ, $v \in Dom(\tau)$ jest warunkiem koniecznym i dostatecznym na to by istniało obliczenie skończone, udane programu $(x := \tau)$, a ponadto, dla $v'$ takiego, że $v'(x) = \tau_{\mathbf{A}}(v)$

$$\mathbf{A}, v \models \gamma(x/\tau) \quad \text{wtw} \quad \mathbf{A}, v' \models \gamma(x)$$

więc ostatecznie

$$\mathbf{A}, v \models (x := \tau)\gamma(x) \quad \text{wtw} \quad \mathbf{A}, v \models (\gamma(x/\tau) \wedge ok(\tau))$$

(2) Rozważmy aksjomat Ax21 postaci

**while** $\gamma$ **do** M **od** $\alpha \equiv ok(\gamma) \wedge ((\neg\gamma \wedge \alpha) \vee (\gamma \wedge$ M (**while** $\gamma$ **do** M **od**)))

Pokażemy, że niezależnie od formuł $\gamma$, $\alpha$ i programu M przedstawiona formuła jest spełniona przez dowolne wartościowanie $v$ w dowolnej strukturze danych **A**.

$\mathbf{A}, v \models \mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha$ wtw

istnieje udane, skończone obliczenie programu **while** $\gamma$ **do** M **od**, którego wynik spełnia $\alpha$ wtw

(na mocy lematu 3.1) istnieje $i \in N$ takie, że dla $j < i$ $v \in Dom(\mathrm{M}^j_{\mathbf{A}})$ i $\mathbf{A}, v \models \mathrm{M}^j(\gamma \wedge ok(\gamma))$ oraz $\mathbf{A}, v \models \mathrm{M}^i(\neg\gamma \wedge ok(\gamma) \wedge \alpha)$ wtw

$\mathbf{A}, v \models (\neg\gamma \wedge ok(\gamma) \wedge \alpha)$, albo istnieje $i \neq 0$ takie, że dla $j < i-1$ $\mathbf{A}, v \models \mathrm{M}(\mathrm{M}^j\gamma)$ oraz $\mathbf{A}, v \models \mathrm{M}(\mathrm{M}^{i-1}(\neg\gamma \wedge ok(\gamma) \wedge \alpha))$, $v \in Dom(\mathrm{M}_{\mathbf{A}})$ i $\mathrm{M}_{\mathbf{A}}(v) \in Dom(\mathrm{M}^{i-1}_{\mathbf{A}})$ wtw

$\mathbf{A}, v \models (\neg\gamma \wedge ok(\gamma) \wedge \alpha)$ albo $v \in Dom(\mathrm{M}_{\mathbf{A}})$ i istnieje $i \neq 0$ takie, że dla $j < i$, $\mathrm{M}_{\mathbf{A}}(v) \in Dom(\mathrm{M}^j_{\mathbf{A}})$ oraz $\mathbf{A}, \mathrm{M}_{\mathbf{A}}(v) \models \mathrm{M}^j(\gamma \wedge ok(\gamma))$ dla $j < i$ i $\mathbf{A}, \mathrm{M}_{\mathbf{A}}(v) \models \mathrm{M}^i(\neg\gamma \wedge ok(\gamma) \wedge \alpha)$ wtw

$\mathbf{A}, v \models (\neg\gamma \wedge ok(\gamma) \wedge \alpha)$ lub $\mathbf{A}, v \models (\gamma \wedge ok(\gamma))$ oraz $v \in Dom(\mathrm{M}_{\mathbf{A}})$ i $\mathbf{A}, \mathrm{M}_{\mathbf{A}}(v) \models \mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha$ wtw

$\mathbf{A}, v \models \big(ok(\gamma) \wedge ((\neg\gamma \wedge \alpha) \vee (\gamma \wedge \mathrm{M}(\mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od})))\big)$

Dowód lematu w pozostałych przypadkach przebiega podobnie. □

Udowodniony lemat stwierdza, że każdy aksjomat systemu AL$(\pi)$ jest schematem formuł prawdziwych w każdej strukturze danych. Następnym krokiem będzie pokazanie, że każda reguła systemu AL$(\pi)$ prowadzi od formuł prawdziwych do formuł prawdziwych.

### Lemat 4.4

Dla dowolnej reguły wnioskowania systemu AL$(\pi)$ postaci $(Z, \alpha)$ i dla dowolnej struktury danych **A**, jeżeli $\mathbf{A} \models Z$, to $\mathbf{A} \models \alpha$.

### Dowód

Dowód polega na sprawdzeniu, że dla każdej reguły prawdziwość przesłanek pociąga prawdziwość wniosku.
Rozważmy regułę R2

$$\frac{(\alpha \Rightarrow \beta)}{(\mathrm{M}\alpha \Rightarrow \mathrm{M}\beta)}$$

Niech $(\alpha \Rightarrow \beta)$ będzie formułą prawdziwą w **A**. Przypuśćmy, że dla pewnego wartościowania $v$

$$A, v \models M\alpha$$

Wynika stąd, że $v \in Dom(M_A)$ oraz $A, M_A(v) \models \alpha$. Na mocy założenia o prawdziwości przesłanki rozważanej reguły, mamy $A, M_A(v) \models \beta$, a zatem

$$A, v \models M\beta$$

Ponieważ $v$ było dowolnym wartościowaniem, więc $A \models (M\alpha \Rightarrow M\beta)$, co należało pokazać.
Rozważmy regułę R3

$$\frac{\{K\,(\mathbf{if}\ \gamma\ \mathbf{then}\ M\ \mathbf{fi})^i(\neg\gamma \wedge ok(\gamma) \wedge \alpha) \Rightarrow \beta\}_{i \in N}}{(K\,(\mathbf{while}\ \gamma\ \mathbf{do}\ M\ \mathbf{od}\ \alpha) \Rightarrow \beta)}$$

gdzie $\gamma$ jest formułą otwartą, $\alpha$, $\beta$ są dowolnymi formułami, K, M są dowolnymi programami. Niech A będzie strukturą danych, a $v$ wartościowaniem takim, że

$$A, v \models K\,(\mathbf{while}\ \gamma\ \mathbf{do}\ M\ \mathbf{od}\ \alpha) \quad \text{i} \quad non\, A, v \models \beta$$

Stąd na mocy lematu 3.3, istnieje takie $i \in N$, że

$$A, K_A(v) \models (\mathbf{if}\ \gamma\ \mathbf{then}\ M\ \mathbf{fi})^i(\neg\gamma \wedge ok(\gamma) \wedge \alpha)$$

i w konsekwencji

$$A, v \models K\,(\mathbf{if}\ \gamma\ \mathbf{then}\ M\ \mathbf{fi})^i(\neg\gamma \wedge ok(\gamma) \wedge \alpha)$$

Ponieważ, na mocy założenia $non\, A, v \models \beta$, więc istnieje takie $i \in N$, że

$$non\, A, v \models (K\,(\mathbf{if}\ \gamma\ \mathbf{then}\ M\ \mathbf{fi})^i(\neg\gamma \wedge ok(\gamma) \wedge \alpha) \Rightarrow \beta)$$

Zatem $i$-ta przesłanka rozważanej reguły nie jest prawdziwa. Dowiedliśmy w ten sposób, że jeżeli wszystkie przesłanki reguły R3 są formułami prawdziwymi w strukturze A, to i wniosek jest formułą prawdziwą w A.
Rozważmy regułę R5

$$\frac{\{(\beta \Rightarrow M\,(K^i\alpha))\}_{i \in N}}{(\beta \Rightarrow M\,(\bigcap K\alpha))}$$

Załóżmy, że dla każdego $i \in N$,

$$A \models (\beta \Rightarrow M\,(K^i\alpha))$$

i przypuśćmy, że dla pewnego wartościowania $v$

$$A, v \models \beta \quad \text{i} \quad non\, A, v \models M\,(\bigcap K\alpha)$$

Stąd albo wartościowanie $M_A(v)$ jest nieokreślone albo jest określone i na mocy definicji kwantyfikatora iteracji (por. p. 3.5) istnieje takie $n \geqslant 0$, że $non\, A, M_A(v) \models (K^n\alpha)$. W obu przypadkach

$$non\, A, v \models (\beta \Rightarrow M\,(K^n\alpha))$$

co przeczy przyjętemu założeniu.
Rozważmy regułę R7

$$\frac{(\beta \Rightarrow \alpha(x))}{(\beta \Rightarrow (\forall x)\, \alpha(x))}$$

Niech $\mathbf{A} \models (\beta \Rightarrow \alpha(x))$ i przypuśćmy, że dla pewnego wartościowania $v$

$$\mathbf{A}, v \models \beta \quad \text{oraz} \quad non\, \mathbf{A}, v \models (\forall x)\, \alpha(x)$$

Na mocy definicji kwantyfikatora $\forall$ (por. p. 2.4) istnieje taka wartość $a$ zmiennej $x$ w $\mathbf{A}$, przy której

$$\mathbf{A}, v_a^x \models \neg\, \alpha(x)$$

Ponieważ zmienna $x$ nie występuje w sposób wolny w formule $\beta$, zatem $\mathbf{A}, v_a^x \models \beta$. Wynika stąd, że

$$non\, \mathbf{A}, v_a^x \models (\beta \Rightarrow \alpha(x))$$

co przeczy założeniu, że przesłanka rozważanej reguły jest prawdziwa przy dowolnym wartościowaniu.
Analogiczny dowód w pozostałych przypadkach pomijamy. □

Podsumowaniem dotychczasowych rozważań jest twierdzenie 4.1. Nosi ono nazwę *twierdzenia o słuszności aksjomatyzacji.*

TWIERDZENIE 4.1

Dla dowolnej formuły $\alpha$ i dowolnego zbioru $Z$, jeżeli $\alpha$ ma dowód ze zbioru $Z$, to $\alpha$ jest prawdziwa w każdym modelu zbioru $Z$, krótko

$$Z \vdash \alpha \quad \text{implikuje} \quad Z \models \alpha$$

DOWÓD

Z założenia $Z \vdash \alpha$ wynika, że istnieje dowód formalny $\langle D, d \rangle$ formuły $\alpha$ ze zbioru $Z$. Niech $\mathbf{A}$ będzie dowolnie wybranym modelem zbioru $Z$. Zauważmy, że jeżeli na poziomie $n$ drzewa dowodu $\langle D, d \rangle$ występuje formuła, która nie jest prawdziwa w $\mathbf{A}$, to na mocy lematu 4.4 w zbiorze formuł-przesłanek reguły użytej do jej wywnioskowania istnieje formuła, która również nie jest prawdziwa w $\mathbf{A}$. Zatem, jeżeli na poziomie $n$ drzewa dowodu istnieje formuła $\beta_n$ taka, że $non\, \mathbf{A} \models \beta_n$, to na poziomie $n+1$-szym drzewa $D$ istnieje formuła $\beta_{n+1}$ i $non\, \mathbf{A} \models \beta_{n+1}$. Ponadto wierzchołki, którym są przypisane te formuły, są połączone krawędzią. Stąd, gdyby $non\, \mathbf{A} \models \alpha$, to w drzewie dowodu $D$ istniałaby gałąź taka, że wszystkie formuły przyporządkowane jej wierzchołkom nie byłyby prawdziwe w $\mathbf{A}$. W szczególności, formuła przypisana liściowi na tej gałęzi nie byłaby prawdziwa w $\mathbf{A}$. Sprzeczność, ponieważ

formuła przypisana liściowi musi być albo aksjomatem, albo musi należeć do zbioru $Z$, a więc musi być prawdziwa. Struktura **A** była dowolnie wybranym modelem zbioru $Z$, więc mamy $Z \models \alpha$, co należało udowodnić. □

Jedną z konsekwencji udowodnionego twierdzenia jest *własność niesprzeczności systemu* formalnego AL(π). Nie istnieje bowiem formuła algorytmiczna taka, że zarówno ona, jak i jej negacja mają dowód formalny w tym systemie. Gdyby formuła taka istniała, i gdyby np. $\alpha$ była taką formułą, to na mocy twierdzenia o słuszności aksjomatyzacji byłoby

$$\mathbf{A}, v \models \alpha \quad \text{oraz} \quad non\, \mathbf{A}, v \models \alpha$$

dla dowolnego wartościowania $v$ w dowolnej strukturze **A**. Czyli formuła $\alpha$ miałaby równocześnie wartość **0** i **1** przy tym samym wartościowaniu, co nie jest możliwe, bo każda formuła (por. p. 2.4 i p. 3.3) wyznacza w zbiorze wartościowań pewną funkcję.

Twierdzenie o słuszności aksjomatyzacji możemy wypowiedzieć w następującej formie:

jeżeli formuła $\alpha$ wynika syntaktycznie ze zbioru $Z$, to wynika ze zbioru $Z$ semantycznie.

System AL(π) umożliwia więc dowodzenie tylko formuł prawdziwych. Powstaje pytanie, czy wszystkie formuły prawdziwe mają dowód w tym systemie. Następujące twierdzenie, zwane *twierdzeniem o pełności*, daje odpowiedź na to pytanie.

TWIERDZENIE 4.2

Dla dowolnego zbioru formuł $Z$ i dla dowolnego formuły algorytmicznej $\alpha$, $Z \models \alpha$ wtedy i tylko wtedy, gdy $Z \vdash \alpha$. ■

Ze względu na objętość tej książki, jak i na jej przeznaczenie, trudny i długi dowód tego twierdzenia pomijamy. Zainteresowanych odsyłamy do pracy [40]. Zwracamy uwagę czytelnika na p. 4.7, w którym znajduje się dowód twierdzenia o pełności logiki algorytmicznej dla nieco innej aksjomatyzacji.

W następującym przykładzie przedstawiamy zastosowanie twierdzenia o pełności. Wykażemy, że istnieje dowód formalny pewnej formuły pokazując jej prawdziwość w dowolnej strukturze danych.

PRZYKŁAD 4.7

Niech $V_{out}(\mathbf{K}) \cap V.(\alpha) = \emptyset$ (por. p. 3.3) dla pewnej formuły $\alpha$ i pewnego programu **K**. Wówczas formuła

$$K\alpha \equiv (\alpha \wedge K\mathbf{true})$$

jest twierdzeniem logiki algorytmicznej.
Rzeczywiście, dla dowolnej struktury **A** i dla dowolnego wartościowania $v$

$$\mathbf{A}, v \models K\alpha \quad \text{wtw} \quad \mathbf{A}, v \models K\mathbf{true} \quad \text{i} \quad \mathbf{A}, K_{\mathbf{A}}(v) \models \alpha$$

Ponieważ na mocy założenia zmienne, które mogą ulec zmianie w wyniku obliczenia programu K, nie występują w formule $\alpha$, zatem $\alpha_{\mathbf{A}}(K_{\mathbf{A}}(v)) = \alpha_{\mathbf{A}}(v)$. W konsekwencji, dla dowolnej struktury **A** mamy

$$\mathbf{A} \models K\alpha \equiv (\alpha \wedge K\mathbf{true})$$

Stąd na mocy twierdzenia o pełności otrzymujemy

$$\vdash K\alpha \equiv (\alpha \wedge K\mathbf{true})$$

Wynik ten, przedstawiony w postaci reguły wnioskowania

$$\frac{K\mathbf{true}}{K\alpha \equiv \alpha} \quad \text{gdzie} \quad V_{out}(K) \cap V(\alpha) = \emptyset \tag{4.4}$$

daje wygodne narzędzie do dowodzenia własności programów. □

PRZYKŁAD 4.8

Wykażemy, że dla dowolnych K, M takich, że $V(M) \cap V(K) = \emptyset$

$$\vdash K(M\alpha) \equiv M(K\alpha)$$ □

DOWÓD

Niech $x$ będzie dowolną zmienną, $v$ dowolnym wartościowaniem, a **A** strukturą danych. Jeżeli $x \in V(K)$, to $x \notin V(M)$, a zatem

$$K_{\mathbf{A}}(M_{\mathbf{A}}(v))(x) = K_{\mathbf{A}}(v)(x) = M_{\mathbf{A}}(K_{\mathbf{A}}(v))(x)$$

Jeżeli $x \notin V(K)$, to

$$K_{\mathbf{A}}(M_{\mathbf{A}}(v))(x) = M_{\mathbf{A}}(v)(x) = M_{\mathbf{A}}(K_{\mathbf{A}}(v))(x)$$

Stąd, dla dowolnej zmiennej $x$, wartościowania $K_{\mathbf{A}}(M_{\mathbf{A}}(v))$ i $M_{\mathbf{A}}(K_{\mathbf{A}}(v))$ pokrywają się. W konsekwencji

$$\mathbf{A}, v \models K(M\alpha) \equiv \mathbf{A}, v \models M(K\alpha),$$

a więc na mocy twierdzenia o pełności $\vdash K(M\alpha) \equiv M(K\alpha)$. □

Podsumowując rozważania tego punktu powtórzmy jeszcze raz, że formuła jest twierdzeniem logiki algorytmicznej wtedy i tylko wtedy, gdy jest tautologią i ponadto, wynikanie semantyczne pokrywa się z wynikaniem

syntaktycznym. System formalny AL$(\pi)$ umożliwia więc zamienne stosowanie metody semantycznej sprawdzania, czy formuła jest twierdzeniem, z metodą syntaktyczną.

Niestety, jak już zauważyliśmy, konstruowanie dowodu formalnego nie jest czynnością mechaniczną i nie jest wcale czynnością łatwą. Tym bardziej że w zestawie reguł wnioskowania istnieją $\omega$-reguły. Postawmy więc pytanie, czy $\omega$-reguła jest rzeczywiście potrzebna? Czy nie można osiągnąć tego samego celu, tzn. pełnej charakteryzacji zbioru formuł prawdziwych, za pomocą systemu aksjomatycznego, w którym nie ma reguł z nieskończoną ilością przesłanek?

Przypuśćmy, że system taki istnieje i oznaczmy go przez $X$. Pamiętajmy, że zasadnicze założenia, jakie narzuciliśmy na ten system, są następujące:

(Z1) wszystkie reguły systemu są skończone;

(Z2) zachodzi twierdzenie o pełności $(\forall Z)(\forall\alpha)\, Z \vdash \alpha \equiv Z \models \alpha$.

Niech $Z$ będzie zbiorem formuł postaci

$$\{(x := 0)\, 0 \leqslant x, \ldots, (x := 0)\,((x := succ(x))^i\, 0 \leqslant x), \ldots\}$$

a $\alpha$ formułą

$$\neg\,(x := 0)\,(\mathbf{while}\; 0 \leqslant x\; \mathbf{do}\; x := succ(x)\; \mathbf{od}\; \mathbf{true})$$

Zauważmy, że $Z \models \alpha$. Zatem na mocy założenia (Z2) mamy $Z \vdash_X \alpha$, tzn. $\alpha$ ma dowód ze zbioru $Z$ w systemie $X$. Zbiór $Z$ jest wprawdzie nieskończony, jednak dowód formalny formuły $\alpha$ ze zbioru $Z$ w systemie $X$ jest skończony, ponieważ wykorzystuje tylko reguły o skończonej ilości przesłanek (Z1). Zatem istnieje podzbiór $Z_0 \subset Z$ złożony tylko z formuł występujących w dowodzie formuły $\alpha$ taki, że $Z_0 \vdash \alpha$. Wykażemy teraz, że jest to niemożliwe.

Rozważmy dowolny skończony podzbiór zbioru $Z$,

$$Z_I = \{((x := 0)(x := succ(x))^i\, 0 \leqslant x\}_{i \in I}$$

dla pewnego skończonego podzbioru $I$ zbioru liczb naturalnych.

Niech **A** będzie strukturą danych taką, że uniwersum systemu **A** jest zbiór liczb naturalnych $N$, interpretacją $succ$ jest operacja następnika w $N$, interpretacją 0 jest stała zero, a interpretacją relacji jednoczłonowej $0 \leqslant$ jest relacja $(0 \leqslant)_{\mathbf{A}}$ określona następująco:

$$(0 \leqslant)_{\mathbf{A}}(n) = \mathbf{true} \quad \text{wtw} \quad n \in I$$

Dla dowolnego wartościowania mamy

$$((x := 0)(x := succ(x))^i (0 \leqslant x)_{\mathbf{A}}(v) = (0 \leqslant)_{\mathbf{A}}\, succ^i_{\mathbf{A}}(0) = (0 \leqslant)_{\mathbf{A}}(i)$$

czyli

$$\mathbf{A}, v \models (x := 0)(x := succ(x))^i\, 0 \leqslant x \quad \text{wtw} \quad i \in I$$

Tak więc struktura **A** jest modelem zbioru $Z_I$. Ponieważ $I$ jest skończonym

podzbiorem zbioru $N$, to istnieje liczba naturalna $j$ taka, że *non* $j \in I$. Dla takiego $j$ mamy

$$non\ \mathbf{A}, v \models (x := 0)(x := succ(x))^j\, 0 \leqslant x$$

Zatem program **begin** $x := 0$; **while** $0 \leqslant x$ **do** $x := succ(x)$ **od end** ma skończone obliczenie i co za tym idzie *non* $\mathbf{A} \models \alpha$.

Wykazaliśmy, że jeśli wybierzemy jakikolwiek podzbiór skończony $Z_0$ zbioru $Z$, to znajdziemy model zbioru $Z_0$, który nie będzie modelem formuły $\alpha$. W konsekwencji mamy *non* $Z_0 \models \alpha$. Sprzeczność, którą otrzymaliśmy, dowodzi, że nie jest możliwe znalezienie systemu formalnego $X$ tak, by założenia (Z1) i (Z2) były spełnione równocześnie.

Reguła $\omega$, chociaż niewygodna w użyciu, pozwala udowodnić wiele innych reguł wnioskowania, tzw. reguł wtórnych (por. definicję 4.5), które umożliwiają dowodzenie własności programów już bez użycia $\omega$-reguły. Przedstawiamy tutaj przykład takiej pożytecznej reguły wtórnej.

Przykład 4.9

Następujący schemat jest poprawną regułą wnioskowania w logice algorytmicznej:

$$\frac{\mathrm{K}_1\,\mathbf{true},\ \mathrm{K}_2\,\mathbf{true}}{\mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha \equiv \mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{K}_1; \mathrm{M}; \mathrm{K}_2\ \mathbf{od}\ \alpha}$$

gdzie $\gamma$ jest dowolną formułą otwartą, $\alpha$ dowolną formułą, a $\mathrm{K}_1$, $\mathrm{K}_2$, M programami takimi, że

$$V_{out}(\mathrm{K}_1) \cap V(\mathrm{M}\gamma \vee \alpha) = \emptyset \quad \text{oraz} \quad V_{out}(\mathrm{K}_2) \cap V(\mathrm{M}\gamma \vee \alpha) = \emptyset$$

Dowód

Niech $Z$ będzie dowolnym zbiorem formuł. Jeżeli $Z \vdash \mathrm{K}_2\,\mathbf{true}$, to na mocy reguły (4.4) mamy

$$Z \vdash (\gamma \equiv \mathrm{K}_2\gamma)$$

Stosując regułę R2 do tego twierdzenia otrzymamy

$$Z \vdash (\mathrm{K}_1(\mathrm{M}\gamma) \equiv \mathrm{K}_1(\mathrm{M}(\mathrm{K}_2\gamma)))$$

Podobnie, jeżeli $Z \vdash \mathrm{K}_1\,\mathbf{true}$, to na mocy reguły (4.4) mamy

$$Z \vdash (\mathrm{M}\gamma \equiv \mathrm{K}_1(\mathrm{M}\gamma))$$

Łącząc te dwa twierdzenia otrzymamy na mocy Ax1

$$Z \vdash (\mathrm{M}\gamma \equiv \mathbf{begin}\ \mathrm{K}_1; \mathrm{M}; \mathrm{K}_2\ \mathbf{end}\ \gamma)$$

Analogicznie otrzymujemy

$$Z \vdash (\mathrm{M}(\neg\gamma \wedge ok(\gamma) \wedge \alpha) \equiv \textbf{begin}\ \mathrm{K}_1; \mathrm{M}; \mathrm{K}_2\ \textbf{end}(\neg\gamma \wedge ok(\gamma) \wedge \alpha))$$

Przyjmijmy jako $\beta$ formułę $(\neg\gamma \wedge ok(\gamma) \wedge \alpha)$. Na mocy aksjomatów Ax1–Ax11 możemy wywnioskować, że

$$Z \vdash (\gamma \wedge ok(\gamma) \wedge \mathrm{M}\beta) \vee (\neg\gamma \wedge ok(\gamma) \wedge \beta) \equiv$$
$$\equiv (\gamma \wedge ok(\gamma) \wedge \mathrm{K}_1(\mathrm{M}\mathrm{K}_2\beta)) \vee (\neg\gamma \wedge ok(\gamma) \wedge \beta)$$

a na mocy aksjomatu Ax20

$$Z \vdash \textbf{if}\ \gamma\ \textbf{then}\ \mathrm{M}\ \textbf{fi}\ \beta \equiv \textbf{if}\ \gamma\ \textbf{then}\ \mathrm{K}_1; \mathrm{M}; \mathrm{K}_2\ \textbf{fi}\ \beta$$

Proste rozumowanie indukcyjne pozwala udowodnić, że dla każdej liczby naturalnej $i$

$$Z \vdash (\textbf{if}\ \gamma\ \textbf{then}\ \mathrm{M}\ \textbf{fi})^i \beta \equiv (\textbf{if}\ \gamma\ \textbf{then}\ \mathrm{K}_1; \mathrm{M}; \mathrm{K}_2\ \textbf{fi})^i \beta$$

(zauważmy, że wobec przyjętych założeń $V_{out}(\mathrm{K}_1) \cap V(\mathrm{M}^i\beta) = \emptyset$ oraz $V_{out}(\mathrm{K}_2) \cap V(\mathrm{M}^i\beta) = \emptyset$). Stąd na mocy reguły R3 oraz własności (4.3) otrzymujemy

$$Z \vdash \textbf{while}\ \gamma\ \textbf{do}\ \mathrm{M}\ \textbf{od}\ \alpha \equiv \textbf{while}\ \gamma\ \textbf{do}\ \mathrm{K}_1; \mathrm{M}; \mathrm{K}_2\ \textbf{od}\ \alpha$$

Wiele różnych przykładów zastosowania udowodnionej reguły znajdzie czytelnik w następnym rozdziale. □

PRZYKŁAD 4.10

Niech $Z$ będzie dowolnym zbiorem formuł i niech $\gamma$, $\delta$ będą formułami otwartymi takimi, że

$$Z \vdash (\gamma \Rightarrow \delta)$$

Niech K, $\mathrm{M}_1$, $\mathrm{M}_2$ będą dowolnymi programami, a $\alpha$ dowolną formułą oraz $V_{out}(\mathrm{M}_1) \cap V(\delta) = \emptyset$. Wtedy następująca formuła jest konsekwencją zbioru formuł $Z$:

$$\textbf{while}\ \gamma\ \textbf{do}\ \mathrm{M}_1; \textbf{if}\ \delta\ \textbf{then}\ \mathrm{K}\ \textbf{fi};\ \mathrm{M}_2\ \textbf{od}\ \alpha \equiv \textbf{while}\ \gamma\ \textbf{do}\ \mathrm{M}_1;\ \mathrm{K};\ \mathrm{M}_2\ \textbf{od}\ \alpha$$

DOWÓD

Na mocy własności udowodnionej w przykładzie 4.7 mamy

$$Z \vdash \mathrm{M}_1\delta \equiv (\delta \wedge \mathrm{M}_1\,\textbf{true}) \qquad Z \models \mathrm{M}_1\neg\delta \equiv (\neg\delta \wedge \mathrm{M}_1\,\textbf{true})$$

a na mocy założenia

$$Z \vdash (\gamma \wedge \delta) \equiv \gamma \qquad\qquad Z \vdash (\gamma \wedge \neg\delta) \equiv \textbf{false}$$

Oznaczmy przez M program **begin** $M_1$; **if** $\delta$ **then** K **fi**; $M_2$ **end** oraz przez M′ program **begin** $M_1$; K; $M_2$ **end**.

Niech ponadto $\beta$ oznacza dowolną formułę. Wtedy formuła

$$\textbf{if}\ \gamma\ \textbf{then}\ \mathrm{M}\ \textbf{fi}\ \beta$$

jest równoważna, kolejno, następującym formułom:

$$ok(\gamma) \wedge ((\gamma \wedge \mathrm{M}\beta) \vee (\neg\gamma \wedge \beta))$$
$$ok(\gamma) \wedge \big((\gamma \wedge \mathrm{M}_1(\delta \wedge \mathrm{K}(\mathrm{M}_2\beta) \vee \neg\delta \wedge \mathrm{M}_2\beta)\big) \vee (\neg\gamma \wedge \beta)\big)$$
$$ok(\gamma) \wedge \big((\gamma \wedge \delta \wedge \mathrm{M}_1(\mathrm{K}(\mathrm{M}_2\beta))) \vee (\gamma \wedge \neg\delta \wedge \mathrm{M}_1(\mathrm{M}_2\beta)) \vee (\neg\gamma \wedge \beta)\big)$$
$$ok(\gamma) \wedge ((\gamma \wedge \mathrm{M}'\beta) \vee (\neg\gamma \wedge \beta))$$
$$\textbf{if}\ \gamma\ \textbf{then}\ \mathrm{M}'\ \textbf{fi}\ \beta$$

Przeprowadzając oczywiste rozumowanie indukcyjne, otrzymamy dla każdego $i$,

$$Z \vdash (\textbf{if}\ \gamma\ \textbf{then}\ \mathrm{M}\ \textbf{fi})^i(\neg\gamma \wedge ok(\gamma) \wedge \alpha) = (\textbf{if}\ \gamma\ \textbf{then}\ \mathrm{M}'\ \textbf{fi})^i(\neg\gamma \wedge ok(\gamma) \wedge \alpha)$$

Na mocy własności (4.3) oraz reguły R3 otrzymujemy ostatecznie

$$Z \vdash \textbf{while}\ \gamma\ \textbf{do}\ \mathrm{M}\ \textbf{od}\ \alpha \equiv \textbf{while}\ \gamma\ \textbf{do}\ \mathrm{M}'\ \textbf{od}\ \alpha$$

Następujący schemat jest zatem poprawną regułą wnioskowania

$$\frac{(\gamma \Rightarrow \delta)}{\textbf{while}\ \gamma\ \textbf{do}\ \mathrm{M}_1;\ \textbf{if}\ \delta\ \textbf{then}\ \mathrm{K}\ \textbf{fi};\ \mathrm{M}_2\ \textbf{od}\ \alpha \equiv \textbf{while}\ \gamma\ \textbf{do}\ \mathrm{M}_1;\ \mathrm{K};\ \mathrm{M}_2\ \textbf{od}\ \alpha}$$

przy założeniu, że $V_{out}(\mathrm{M}_1) \cap V(\delta) = \emptyset$. □

PRZYKŁAD 4.11

Następujące schematy są przykładami prostych reguł wtórnych, których dowód nie wymaga zastosowania $\omega$-reguły:

$$\frac{(\alpha \Rightarrow (ok(\gamma) \wedge \neg\gamma))}{(\alpha \wedge \textbf{if}\ \gamma\ \textbf{then}\ \mathrm{M}\ \textbf{fi}\ \beta) \equiv (\alpha \wedge \beta)}$$

$$\frac{(\alpha \Rightarrow (ok(\gamma) \wedge \neg\gamma))}{(\alpha \wedge \textbf{while}\ \gamma\ \textbf{do}\ \mathrm{M}\ \textbf{od}\ \beta) \equiv (\alpha \wedge \beta)}$$

gdzie $\gamma$ jest formułą otwartą, $\alpha$, $\beta$ są dowolnymi formułami, a M jest dowolnym programem.

DOWÓD

Jeżeli formuła $(\alpha \Rightarrow (ok(\gamma) \wedge \neg\gamma))$ ma dowód z pewnego zbioru formuł $Z$, to na mocy aksjomatów Ax2 i Ax7 formuły

$$((\alpha \wedge \gamma) \equiv (\neg \gamma \wedge ok(\gamma) \wedge \gamma))$$

oraz

$$((\alpha \wedge \beta) \equiv (\neg \gamma \wedge ok(\gamma) \wedge \alpha \wedge \beta))$$

mają dowód. Wynika stąd

$$Z \vdash (\alpha \wedge ok(\gamma) \wedge (\gamma \wedge \mathrm{M}\delta \vee \neg \gamma \wedge \beta) \equiv (\alpha \wedge \beta))$$

gdzie $\delta$ jest dowolną formułą. Przyjmując we wzorze jako $\delta$ raz formułę $\beta$, a raz formułę (**while** $\gamma$ **do** M **od** $\beta$) oraz korzystając odpowiednio z aksjomatu Ax20 lub Ax21 otrzymamy

$$Z \vdash (\alpha \wedge \textbf{if}\, \gamma\, \textbf{then}\, \mathrm{M}\, \textbf{fi}\, \beta) \equiv (\alpha \wedge \beta)$$
$$Z \vdash (\alpha \wedge \textbf{while}\, \gamma\, \textbf{do}\, \mathrm{M}\, \textbf{od}\, \beta) \equiv (\alpha \wedge \beta)$$ □

Twierdzenie, które zamierzamy udowodnić na zakończenie tego punktu, ma duże znaczenie dla lepszego zrozumienia czym jest proces tworzenia dowodu.

TWIERDZENIE 4.3

Dla dowolnych formuł $\alpha$, $\beta$ i dla dowolnego zbioru formuł $Z$

$$Z \cup \{\alpha\} \vdash \beta \quad \text{wtw} \quad Z \vdash ((\forall \mathbf{x})\, \alpha(\mathbf{x}) \Rightarrow \beta)$$

gdzie **x** jest zbiorem wszystkich zmiennych wolnych występujących w formule $\alpha$.

DOWÓD

Udowodnimy jedynie implikację „⇒"
Dowód implikację odwrotnej, polegający na zastosowaniu reguły modus ponens, pomijamy.
Niech **A** będzie modelem zbioru $Z$ i niech dla pewnego dowolnie ustalonego wartościowania $v_0$

$$\mathbf{A}, v_0 \models (\forall \mathbf{x})\, \alpha(\mathbf{x})$$

Ponieważ wartość formuły $(\forall \mathbf{x})\, \alpha(\mathbf{x})$ nie zależy od wartości zmiennych **x**, więc dla dowolnego $v$ mamy

$$\mathbf{A}, v \models (\forall \mathbf{x})\, \alpha(\mathbf{x})$$

Wynika stąd, że **A** jest modelem zbioru $Z \cup \{\alpha\}$. Jeżeli założymy, że $Z \cup \{\alpha\} \vdash \beta$, to z twierdzenia o pełności (4.2) otrzymamy

$$\mathbf{A} \models \beta$$

W szczególności $\mathbf{A}, v_0 \models \beta$.

W konsekwencji, każdy model zbioru $Z$ jest też modelem formuły $(\forall \mathbf{x})\alpha(\mathbf{x}) \Rightarrow \beta$, tzn. $Z \models (\forall \mathbf{x})\alpha(\mathbf{x}) \Rightarrow \beta$. Na mocy twierdzenia o pełności mamy

$$Z \vdash (\forall \mathbf{x})\alpha(\mathbf{x}) \Rightarrow \beta \qquad \square$$

Jako natychmiastowy wniosek otrzymujemy następujące twierdzenie zwane *twierdzeniem o dedukcji.*

TWIERDZENIE 4.4

Dla dowolnej formuły $\beta$, dowolnego zbioru formuł $Z$ i dowolnej zamkniętej formuły $\alpha$

$$Z \cup \{\alpha\} \vdash \beta \quad \text{wtw} \quad Z \vdash (\alpha \Rightarrow \beta) \qquad \blacksquare$$

## Teorie algorytmiczne 4.4

Jeżeli formuła jest twierdzeniem logiki algorytmicznej, to jest prawdziwa w każdej strukturze danych. Prawdziwość takiej formuły wynika więc z jej budowy syntaktycznej, nie zależy natomiast od przyjętych wartości zmiennych czy wyboru interpretacji symboli funkcyjnych i relacyjnych języka. Twierdzenia logiki wyrażają zatem własności niezależne od struktury danych. Zdarzają się jednak sytuacje, w których chcemy lub musimy ograniczyć nasze rozważania. Często interesują nas nie zdania ogólnie prawdziwe, ale te prawdziwe w wybranej przez nas klasie struktur lub wręcz w wybranej strukturze danych.

Wyboru klasy struktur możemy dokonać przez przyjęcie pewnych dodatkowych założeń, które będą tę klasę charakteryzowały. Jeżeli chcemy poznać własności struktur uporządkowanych, wystarczy przyjąć jako dodatkowe założenia następujący zbiór $Z$ zdań

$$(\forall x)\, x \leqslant x$$
$$(\forall x, y)(x \leqslant y \wedge y \leqslant x \Rightarrow x = y)$$
$$(\forall x, y, z)(x \leqslant y \wedge y \leqslant z \Rightarrow x \leqslant z)$$

Zgodnie z przyjętą wcześniej umową i twierdzeniem o pełności logiki algorytmicznej $Z \vdash \alpha$ oznacza, że $\alpha$ wyraża pewną własność przysługującą każdej uporządkowanej strukturze danych.

DEFINICJA 4.6

Niech Aks będzie niepustym zbiorem formuł języka algorytmicznego $L(\pi)$ i niech $\vdash$ będzie operacją syntaktycznej konsekwencji wyznaczoną przez aksjomaty i reguły wnioskowania systemu $AL(\pi)$. System $T = \langle L(\pi), \vdash, \mathrm{Aks}\rangle$

będziemy nazywać *teorią algorytmiczną*, zbiór Aks zaś zbiorem aksjomatów specyficznych teorii T. ■

PRZYKŁAD 4.12

Niech Ar będzie zbiorem złożonym z następujących formuł:

Ar1 $\neg succ(x) = 0$

Ar2 $((succ(x) = succ(y) \Rightarrow x = y)$

Ar3 $\textbf{begin}\ x := 0;\ \textbf{while}\ \neg x = y\ \textbf{do}\ x := succ(x)\ \textbf{do end true}$

gdzie $x, y$ są zmiennymi indywiduowymi, *succ* jest funktorem jednoargumentowym, 0 jest stałą, a = równością.

System $\langle L(\pi), \vdash, Ar\rangle$ jest przykładem teorii algorytmicznej. Zbiór Ar, wraz ze zbiorem aksjomatów równości, tworzy zbiór jej aksjomatów specyficznych. Teorię tę będziemy nazywać arytmetyką liczb naturalnych. □

DEFINICJA 4.7

Powiemy, że struktura danych **A** dla języka $L(\pi)$ jest modelem teorii $T = \langle L(\pi), \vdash, Aks\rangle$ wtedy i tylko wtedy, gdy **A** jest modelem zbioru Aks, tzn. $(\forall\alpha \in Aks)\ \mathbf{A} \models \alpha$. ■

PRZYKŁAD 4.13

Modelem teorii $\langle L(\pi), \vdash, Ar\rangle$ jest standardowa struktura liczb naturalnych $\mathbf{N} = \langle N, s, 0; =\rangle$. Operacja następnika $s$ jest interpretacją funktora *succ*, zero 0 jest interpretacją stałej 0, a identyczność jest interpretacją predykatu = (por. przykład 3.13 oraz 2.21).

Rzeczywiście, niech $v$ będzie dowolnym wartościowaniem takim, że $v(y) = n$. Mamy wtedy

$$\mathbf{N}, v \models succ^n(0) = y \quad \text{i} \quad non\,\mathbf{N}, v \models succ^j(0) = y \quad \text{dla} \quad j \neq n$$

Zatem

$$\mathbf{N}, v \models (x := 0)(x := succ(x))^n(x = y)$$

oraz

$$\mathbf{N}, v \models (x := 0)(x := succ(x))^j(x = y) \quad \text{dla} \quad j \neq n$$

Stąd i z definicji semantyki programów mamy

$$\mathbf{N}, v \models (x := 0)(\textbf{while}\ x \neq y\ \textbf{do}\ x := succ(x)\ \textbf{od true})$$

Ponieważ $v$ jest dowolnym wartościowaniem, więc formuła Ar3 jest praw-

dziwa w N. Oczywiście $\mathbf{N} \models \mathrm{Ar1}$, bo nie istnieje liczba naturalna, której następnikiem byłoby 0. Ponadto $\mathbf{N} \models \mathrm{Ar2}$, bo następnik *succ* w zbiorze liczb naturalnych jest funkcją różnowartościową. Ostatecznie **N** jest modelem zbioru formuł Ar. □

Jeżeli teoria T jest oparta na pewnym języku z równością L, to będziemy zakładać, że zbiór aksjomatów tej teorii zawiera następujące aksjomaty równości (por. przykład 2.19):

$$(\forall x)\, x = x$$
$$(\forall x, y)(x = y \Rightarrow y = x)$$
$$(\forall x, y, z)(x = y \wedge y = z \Rightarrow x = z)$$
$$(\forall \mathbf{x}, \mathbf{y})\big(\mathbf{x} = \mathbf{y} \wedge ok(\varphi(\mathbf{x})) \wedge ok(\varphi(\mathbf{y})) \Rightarrow \varphi(\mathbf{x}) = \varphi(\mathbf{y})\big) \quad \text{dla} \quad \varphi \in \Phi$$
$$(\forall \mathbf{x}, \mathbf{y})(\mathbf{x} = \mathbf{y} \Rightarrow \varrho(\mathbf{x}) = \varrho(\mathbf{y})) \quad \text{dla} \quad \varrho \in \mathrm{P}$$

i pamiętając o tym nie będziemy ich za każdym razem wymieniać.
Zbiór przedstawionych tu aksjomatów może mieć różne modele. Jednym z nich, ale nie jedynym, jest struktura, w której predykat = jest interpretowany jako identyczność. Modele teorii z równością, w których równość jest interpretowana jako identyczność, nazywamy *modelami właściwymi dla pojęcia równości.*

TWIERDZENIE 4.5

Niech teoria T zawiera wśród aksjomatów zbiór aksjomatów równości Axeq (użyjemy tu symbolu eq zamiast =), charakteryzujący pewien predykat dwuargumentowy *eq* (por. p. 2.4).
Jeżeli teoria z równością ma model, to ma model właściwy dla pojęcia równości.

DOWÓD

Zgodnie z rozważaniami przeprowadzonymi w rozdz. 2 predykat *eq* wyznacza w **A** relację kongruencji (por. definicję 2.11).
Możemy zatem mówić o strukturze ilorazowej. Niech $[a]$ oznacza klasę abstrakcji relacji $\approx$ wyznaczoną przez element $a$.
Niech dla dowolnego wartościowania $v$ w strukturze **A**, $[v]$ oznacza wartościowanie w strukturze $\mathbf{A}/\approx$ takie, że

$$[v](x) \stackrel{\mathrm{df}}{=} [v(x)] \quad \text{dla dowolnej zmiennej indywiduowej } x$$
$$[v](q) \stackrel{\mathrm{df}}{=} v(q) \quad \text{dla dowolnej zmiennej zdaniowej } q$$

Łatwo zauważyć, że dla dowolnego termu $\tau$ i dowolnej formuły otwartej $\gamma$ zachodzą następujące równości:

$$[\tau_{\mathbf{A}}(v)] = \tau_{\mathbf{A}/\approx}([v])$$
$$\gamma_{\mathbf{A}}(v) = \gamma_{\mathbf{A}/\approx}([v])$$

Przez indukcję ze względu na stopień komplikacji programu M i formuły $\alpha$ można pokazać (dokładny dowód pomijamy), że $M_{\mathbf{A}}(v)$ jest określone wtedy i tylko wtedy, gdy $M_{\mathbf{A}/\approx}([v])$ jest określone oraz dla dowolnego wartościowania $v$ takiego, że $M_{\mathbf{A}}(v)$ jest określone

$$[M_{\mathbf{A}}(v)] = M_{\mathbf{A}/\approx}([v])$$
$$\mathbf{A}, v \models \alpha \quad \text{wtw} \quad \mathbf{A}/\approx, [v] \models \alpha$$

Rozważmy dowolne wartościowanie $v^*$ w strukturze danych $\mathbf{A}/\approx$ oraz dowolną formułę $\beta$ będącą aksjomatem specyficznym teorii T. Wtedy, na mocy powyższej równoważności, dla wartościowania $v'$ takiego, że $v'(x) \in v^*(x)$ dla dowolnej zmiennej indywiduowej $x$ oraz $v'(q) = v^*(q)$ dla dowolnej zmiennej zdaniowej $q$ mamy

$$\mathbf{A}, v' \models \beta \quad \text{wtw} \quad \mathbf{A}/\approx, v^* \models \beta$$

Ponieważ **A** jest modelem formuły $\beta$, zatem dla dowolnego wartościowania $v^*$, $\mathbf{A}/\approx, v^* \models \beta$. Wynika stąd, że $\mathbf{A}/\approx$ jest modelem teorii T. Predykat *eq* jest w tym modelu interpretowany jako identyczność, ponieważ zachodzą następujące równoważności:

$$eq_{\mathbf{A}/\approx}([a]), [b]) \quad \text{wtw} \quad eq_{\mathbf{A}}(a, b) \quad \text{wtw} \quad a \approx b \quad \text{wtw} \quad [a] = [b]$$

Wykazaliśmy więc, że struktura $\mathbf{A}/\approx$ jest modelem teorii T właściwym dla pojęcia równości. □

DEFINICJA 4.8

Niech $T = \langle L(\pi), \vdash, Aks \rangle$ będzie teorią algorytmiczną. Każdą formułę $\alpha$, która ma dowód ze zbioru Aks aksjomatów specyficznych, symbolicznie $Aks \vdash \alpha$, nazywamy twierdzeniem teorii T. ■

Twierdzenie o pełności (por. twierdzenie 4.2) możemy teraz wypowiedzieć nieco inaczej, stosując przyjęte definicje.

TWIERDZENIE 4.6

Następujące warunki są równoważne:

(1) formuła $\alpha$ jest twierdzeniem teorii T;

(2) formuła $\alpha$ jest prawdziwa we wszystkich modelach teorii T. ■

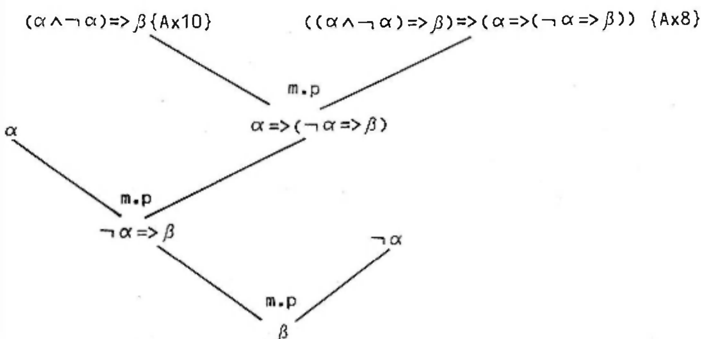
Każda teoria charakteryzuje klasę struktur wyznaczoną przez aksjomaty specyficzne, mianowicie klasę wszystkich modeli zbioru aksjomatów

specyficznych. Jest jednak jeden przypadek, gdy teoria nie wyznacza żadnej specjalnej klasy obiektów. Teorię taką nazywamy sprzeczną.

DEFINICJA 4.9

Powiemy, że teoria T jest niesprzeczna wtedy i tylko wtedy, gdy nie istnieje formuła $\alpha$ w języku tej teorii taka, że równocześnie $\alpha$ i $\neg\alpha$ są jej twierdzeniami. W przeciwnym razie teorię nazywamy sprzeczną. ■

Teoria, w której można udowodnić jednocześnie formułę i jej negację nie jest interesującym przedmiotem badań, gdyż można w niej udowodnić wszystko (rys. 4.4).




RYS. 4.4

Z przyjętej definicji wynika, że jeżeli teoria jest niesprzeczna, to istnieje formuła, która nie jest jej twierdzeniem. Odwrotnie, jeżeli chociaż jedna formuła w języku teorii nie jest twierdzeniem teorii, to teoria ta jest niesprzeczna. Podsumowując te rozważania dochodzimy do prostego spostrzeżenia: teoria jest niesprzeczna wtedy i tylko wtedy, gdy istnieje formuła, która nie jest jej twierdzeniem.

Wprost z definicji 4.9 wynika również, że jeżeli teoria jest sprzeczna, to nie ma modelu. Czy jest też odwrotnie?

Jeżeli teoria T ze zbiorem aksjomatów specyficznych Aks nie ma modelu, to dla dowolnej formuły $\alpha$, $Aks \models \alpha$. Zatem na mocy twierdzenia o pełności $Aks \vdash \alpha$ dla dowolnej formuły $\alpha$, czyli teoria T jest sprzeczna. W ten sposób otrzymaliśmy bardzo ważną charakteryzację teorii niesprzecznych — zwaną *twierdzeniem o istnieniu modelu*.

TWIERDZENIE 4.7

Teoria algorytmiczna jest niesprzeczna wtedy i tylko wtedy, gdy ma model. ■

UWAGA

Twierdzenie o istnieniu modelu uzyskaliśmy w sposób elementarny z twierdzenia o pełności, nie mówiąc w istocie jak taki model dla niesprzecznej teorii zbudować. Najczęściej jednak sytuacja wygląda inaczej. Najpierw dowodzi się twierdzenia o istnieniu modelu, wskazując jak można otrzymać model (często dowód nie jest konstruktywny), a potem za jego pomocą dowodzi się twierdzenia o pełności dla niesprzecznej teorii. Głębsze rozważania na ten temat przekraczają ramy tej książki. Zainteresowanych czytelników odsyłamy do pracy [45]. ■

TWIERDZENIE 4.8

Dla dowolnych struktur danych **A**, **B** i dla dowolnego zbioru formuł $Z$, jeżeli **A** jest izomorficzne z **B**, to $\mathbf{A} \models Z$ wtedy i tylko wtedy, gdy $\mathbf{B} \models Z$.

DOWÓD

Niech $h$ będzie izomorfizmem odwzorowującym strukturę **A** na **B**. Dowód twierdzenia przebiega w dwóch etapach, w których dowiedziemy przez indukcję, że dla dowolnego programu M, dowolnej formuły $\alpha$ i dowolnego wartościowania $v$ w **A**, $v \in Dom(\mathrm{M}_{\mathbf{A}})$ wtw $(h \circ v) \in Dom(\mathrm{M}_{\mathbf{B}})$ oraz

$$h(\mathrm{M}_{\mathbf{A}}(v)) = \mathrm{M}_{\mathbf{B}}(h \circ v) \quad \text{dla} \quad v \in Dom(\mathrm{M}_{\mathbf{A}}) \tag{4.5}$$

$$\mathbf{A}, v \models \alpha \quad \text{wtw} \quad \mathbf{B}, h \circ v \models \alpha \tag{4.6}$$

Niech $v$ będzie ustalonym wartościowaniem w strukturze **A**. Rozważmy program postaci $(x := \tau)$. Na mocy twierdzenia 2.1

$$v \in Dom(\tau_{\mathbf{A}}) \quad \text{wtw} \quad h \circ v \in Dom(\tau_{\mathbf{B}}) \quad \text{oraz} \quad h(\tau_{\mathbf{A}}(v)) = \tau_{\mathbf{B}}(h \circ v)$$

Zatem dla $v \in Dom(\tau_{\mathbf{A}})$ i $v_1 = (x := \tau)_{\mathbf{A}}(v)$, $v_2 = (x := \tau)_{\mathbf{B}}(h \circ v)$ mamy $h \circ v_1 = v_2$, co dowodzi własności (4.5) w przypadku instrukcji przypisania.

Niech M będzie instrukcją warunkową postaci

**if** $\gamma$ **then** $\mathrm{M}_1$ **else** $\mathrm{M}_2$ **fi**

i załóżmy, że własność (4.5) jest udowodniona dla programów $\mathrm{M}_1$ i $\mathrm{M}_2$. Na mocy twierdzenia 2.1

$$\mathbf{A}, v \models (\gamma \wedge ok(\gamma)) \quad \text{wtw} \quad \mathbf{B}, h \circ v \models (\gamma \wedge ok(\gamma))$$

a na mocy założenia indukcyjnego dla $i = 1, 2$

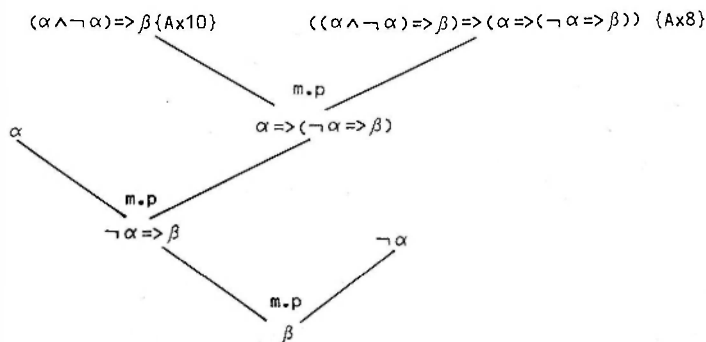
$$v \in Dom(\mathrm{M}_{i\mathbf{A}}) \quad \text{wtw} \quad (h \circ v) \in Dom(\mathrm{M}_{i\mathbf{B}}) \quad \text{i}$$
$$h(\mathrm{M}_{i\mathbf{A}}(v)) = \mathrm{M}_{i\mathbf{B}}(h \circ v) \quad \text{dla} \quad v \in Dom(\mathrm{M}_{i\mathbf{A}})$$

specyficznych. Jest jednak jeden przypadek, gdy teoria nie wyznacza żadnej specjalnej klasy obiektów. Teorię taką nazywamy sprzeczną.

DEFINICJA 4.9

Powiemy, że teoria T jest niesprzeczna wtedy i tylko wtedy, gdy nie istnieje formuła $\alpha$ w języku tej teorii taka, że równocześnie $\alpha$ i $\neg\alpha$ są jej twierdzeniami. W przeciwnym razie teorię nazywamy sprzeczną. ■

Teoria, w której można udowodnić jednocześnie formułę i jej negację nie jest interesującym przedmiotem badań, gdyż można w niej udowodnić wszystko (rys. 4.4).




RYS. 4.4

Z przyjętej definicji wynika, że jeżeli teoria jest niesprzeczna, to istnieje formuła, która nie jest jej twierdzeniem. Odwrotnie, jeżeli chociaż jedna formuła w języku teorii nie jest twierdzeniem teorii, to teoria ta jest niesprzeczna. Podsumowując te rozważania dochodzimy do prostego spostrzeżenia: teoria jest niesprzeczna wtedy i tylko wtedy, gdy istnieje formuła, która nie jest jej twierdzeniem.

Wprost z definicji 4.9 wynika również, że jeżeli teoria jest sprzeczna, to nie ma modelu. Czy jest też odwrotnie?

Jeżeli teoria T ze zbiorem aksjomatów specyficznych Aks nie ma modelu, to dla dowolnej formuły $\alpha$, Aks $\models \alpha$. Zatem na mocy twierdzenia o pełności Aks $\vdash \alpha$ dla dowolnej formuły $\alpha$, czyli teoria T jest sprzeczna. W ten sposób otrzymaliśmy bardzo ważną charakteryzację teorii niesprzecznych — zwaną *twierdzeniem o istnieniu modelu.*

TWIERDZENIE 4.7

Teoria algorytmiczna jest niesprzeczna wtedy i tylko wtedy, gdy ma model. ■

UWAGA

Twierdzenie o istnieniu modelu uzyskaliśmy w sposób elementarny z twierdzenia o pełności, nie mówiąc w istocie jak taki model dla niesprzecznej teorii zbudować. Najczęściej jednak sytuacja wygląda inaczej. Najpierw dowodzi się twierdzenia o istnieniu modelu, wskazując jak można otrzymać model (często dowód nie jest konstruktywny), a potem za jego pomocą dowodzi się twierdzenia o pełności dla niesprzecznej teorii. Głębsze rozważania na ten temat przekraczają ramy tej książki. Zainteresowanych czytelników odsyłamy do pracy [45]. ■

TWIERDZENIE 4.8

Dla dowolnych struktur danych **A**, **B** i dla dowolnego zbioru formuł $Z$, jeżeli **A** jest izomorficzne z **B**, to $\mathbf{A} \models Z$ wtedy i tylko wtedy, gdy $\mathbf{B} \models Z$.

DOWÓD

Niech $h$ będzie izomorfizmem odwzorowującym strukturę **A** na **B**. Dowód twierdzenia przebiega w dwóch etapach, w których dowiedziemy przez indukcję, że dla dowolnego programu M, dowolnej formuły $\alpha$ i dowolnego wartościowania $v$ w **A**, $v \in Dom(\mathrm{M}_{\mathbf{A}})$ wtw $(h \circ v) \in Dom(\mathrm{M}_{\mathbf{B}})$ oraz

$$h(\mathrm{M}_{\mathbf{A}}(v)) = \mathrm{M}_{\mathbf{B}}(h \circ v) \quad \text{dla} \quad v \in Dom(\mathrm{M}_{\mathbf{A}}) \tag{4.5}$$

$$\mathbf{A}, v \models \alpha \quad \text{wtw} \quad \mathbf{B}, h \circ v \models \alpha \tag{4.6}$$

Niech $v$ będzie ustalonym wartościowaniem w strukturze **A**. Rozważmy program postaci $(x := \tau)$. Na mocy twierdzenia 2.1

$$v \in Dom(\tau_{\mathbf{A}}) \quad \text{wtw} \quad h \circ v \in Dom(\tau_{\mathbf{B}}) \quad \text{oraz} \quad h(\tau_{\mathbf{A}}(v)) = \tau_{\mathbf{B}}(h \circ v)$$

Zatem dla $v \in Dom(\tau_{\mathbf{A}})$ i $v_1 = (x := \tau)_{\mathbf{A}}(v)$, $v_2 = (x := \tau)_{\mathbf{B}}(h \circ v)$ mamy $h \circ v_1 = v_2$, co dowodzi własności (4.5) w przypadku instrukcji przypisania.

Niech M będzie instrukcją warunkową postaci

**if** $\gamma$ **then** $\mathrm{M}_1$ **else** $\mathrm{M}_2$ **fi**

i załóżmy, że własność (4.5) jest udowodniona dla programów $\mathrm{M}_1$ i $\mathrm{M}_2$. Na mocy twierdzenia 2.1

$$\mathbf{A}, v \models (\gamma \wedge ok(\gamma)) \quad \text{wtw} \quad \mathbf{B}, h \circ v \models (\gamma \wedge ok(\gamma))$$

a na mocy założenia indukcyjnego dla $i = 1, 2$

$$v \in Dom(\mathrm{M}_{i\mathbf{A}}) \quad \text{wtw} \quad (h \circ v) \in Dom(\mathrm{M}_{i\mathbf{B}}) \quad \text{i}$$
$$h(\mathrm{M}_{i\mathbf{A}}(v)) = \mathrm{M}_{i\mathbf{B}}(h \circ v) \quad \text{dla} \quad v \in Dom(\mathrm{M}_{i\mathbf{A}})$$

Łącząc przedstawione fakty otrzymamy własność (4.5) dla programu **if** $\gamma$ **then** $M_1$ **else** $M_2$ **fi**.

Analogicznie przebiega dowód dla programu złożonego postaci **begin** $M_1$; $M_2$ **end**. Rozważmy program iteracyjny postaci

**while** $\gamma$ **do** K **od**

i załóżmy, że własność (4.5) jest prawdziwa dla programu K. Stąd i z poprzednich rozważań, własność (4.5) zachodzi dla wszystkich programów postaci $K^i$, gdzie $i \in N$.

Jeżeli $v \in Dom(\mathbf{while}\ \gamma\ \mathbf{do}\ K\ \mathbf{od}_{\mathbf{A}})$, to istnieje takie $j$, że **while** $\gamma$ **do** K $\mathbf{od}_{\mathbf{A}}(v) = v' = K^j(v)$ oraz $\mathbf{A}, v' \models (\neg\gamma \wedge ok(\gamma))$, a na mocy założenia indukcyjnego $(h \circ v) \in Dom(K^j_{\mathbf{B}})$ i $h(K^j_{\mathbf{A}}(v)) = K^j_{\mathbf{B}}(h \circ v)$. Z twierdzenia 2.1 $\mathbf{B}, h \circ v' \models$ $\models (\neg\gamma \wedge ok(\gamma))$. W konsekwencji

$$h \circ v \in Dom(\mathbf{while}\ \gamma\ \mathbf{do}\ K\ \mathbf{od}_{\mathbf{B}})$$

oraz

$$h(\mathbf{while}\ \gamma\ \mathbf{do}\ K\ \mathbf{od}_{\mathbf{A}}(v)) = \mathbf{while}\ \gamma\ \mathbf{do}\ K\ \mathbf{od}_{\mathbf{B}}(h \circ v)$$

Z tych rozważań wynika, że własność (4.5) jest prawdziwa dla dowolnych programów.

Rozważmy teraz formuły. Własność (4.6) udowodniliśmy wcześniej dla formuł klasycznych (por. twierdzenie 2.1). Rozważmy formułę postaci $M\alpha$, gdzie M jest programem, a $\alpha$ dowolną formułą, dla której zachodzi własność (4.6). Na mocy definicji semantyki i założenia indukcyjnego mamy

$$\begin{array}{lll} \mathbf{A}, v \models M\alpha & \text{wtw}\ \ \mathbf{A}, M_{\mathbf{A}}(v) \models \alpha\ \ \text{wtw}\ \ \mathbf{B}, M_{\mathbf{B}}(v) \models \alpha\ \ \text{wtw}\ \ \mathbf{B}, h \circ v \models M\alpha, \\ \mathbf{A}, v \models \bigcup M\alpha & \text{wtw}\ \ (\exists i)\, \mathbf{A}, v \models M^i\alpha\ \ \text{wtw}\ \ (\exists i)\, \mathbf{A}, M^i_{\mathbf{A}}(v) \models \alpha \\ & \text{wtw}\ \ (\exists i)\, \mathbf{B}, M^i_{\mathbf{B}}(v) \models \alpha\ \ \text{wtw}\ \ \mathbf{B}, h \circ v \models \bigcup M\alpha \end{array}$$

We wszystkich pozostałych przypadkach dowód własności (4.6) przebiega analogicznie.

Niech **A** będzie modelem zbioru formuł Z. Zatem każda formuła $\alpha$ jest prawdziwa w **A**. Weźmy dowolne wartościowanie $v$ w **B**. Ponieważ $h$ jest odwzorowaniem na zbiór **B**, zatem istnieje takie wartościowanie $v'$ w **A**, że $h \circ v' = v$. Z założenia $\mathbf{A}, v' \models \alpha$, więc na mocy własności (4.6) $\mathbf{B}, h \circ v' \models \alpha$ i w konsekwencji $\mathbf{B}, v \models \alpha$. Stąd **B** jest modelem formuły $\alpha$. □

Z twierdzenia 4.8 wynika, że jeżeli teoria ma chociaż jeden model, to ma ich nieskończenie wiele, bo każda struktura izomorficzna z modelem jest też modelem tej teorii. Jeżeli teoria nie ma żadnych innych modeli, to powiemy, że charakteryzuje pewną strukturę z dokładnością do izomorfizmu.

DEFINICJA 4.10

Teorię nazywamy *kategoryczną*, jeżeli jest niesprzeczna i jej dowolne dwa modele są izomorficzne. ■

TWIERDZENIE 4.9

Algorytmiczna arytmetyka liczb naturalnych (por. przykład 4.12) jest teorią kategoryczną.

DOWÓD

Pokażemy, że każdy model teorii T = ⟨L(π), ⊢, Ar⟩ jest izomorficzny z modelem standardowym **N** = ⟨$N$, $s$, 0; =⟩.

Niech **A** = ⟨$A$, $f$, cons, =⟩ będzie modelem zbioru aksjomatów Ar, gdzie $f$ jest interpretacją funktora $s$, a *cons* jest stałą odpowiadającą zeru. Zauważmy, że wobec prawdziwości aksjomatu Ar2 w strukturze **A**, $f$ jest funkcją całkowitą. Niech $h$ będzie funkcją taką, że

$$h(0) = cons$$
$$h(s(n)) = f(h(n))$$

Funkcja $h$ jest więc określona na zbiorze liczb naturalnych, a jej wartościami są elementy z $A$.

(1) Czy $h$ jest funkcją różnowartościową?

Niech $n$ będzie dowolną liczbą naturalną różną od zera i niech $m = 0$, wtedy $n = s(n-1)$ oraz $h(n) = f(h(n-1))$ i $h(m) = cons$. Na mocy aksjomatu Ar1 *cons* nie jest wartością funkcji $f$. Zatem $f(h(n-1)) \neq cons$ i $h(n) \neq h(0)$.

Niech $n$, $m$ będą ustalonymi liczbami naturalnymi takimi, że $n \neq m$. Załóżmy, że dla dowolnych $i$, $j$ takich, że $i < n$ i $j < m$, jeżeli $i \neq j$, to $h(i) \neq h(j)$ (założenie indukcyjne).

Rozważmy parę $(n, m)$. Mamy $n = s(n-1)$ i $m = s(m-1)$. Oczywiście $(n-1) \neq (m-1)$. Z założenia indukcyjnego, $h(m-1) \neq h(n-1)$ oraz z definicji funkcji $h$, $h(m) = f(h(m-1))$ i $h(n) = f(h(n-1))$. Na mocy założenia A ⊢ Ar2, więc $f$ jest funkcją różnowartościową. W konsekwencji $h(n) \neq h(m)$. Na mocy zasady indukcji matematycznej dla dowolnych $n \neq m$ mamy $h(n) \neq h(m)$.

(2) Czy $h$ jest odwzorowaniem na zbiór **A**?

Rozważmy dowolny element $a \in A$. Jeżeli $a = cons$, to na mocy definicji, $a$ jest wartością funkcji $h$. Jeżeli $a \neq cons$, to na mocy aksjomatu Ar3 dla wartościowania $v$ takiego, że $v(y) = a$, program

**begin** $x := 0$; **while** $x \neq y$ **do** $x := succ(x)$ **od end**

nie zapętla się przy wartościowaniu $v$ w **A**. Wynika stąd, że istnieje liczba naturalna $i > 0$ taka, że $a = v(y) = f^i(cons)$. Z definicji funkcji $h$

$$f^i(cons) = h(s^i(0)) = h(i)$$

a więc $a$ jest wartością funkcji $h$ dla argumentu $i$.

Zatem $h$ jest funkcją różnowartościową odwzorowującą $N$ na $A$ i taką, że $h(s(n)) = f(h(n))$ dla dowolnego $n \in N$, a więc $h$ ustala izomorfizm struktury **A** i standardowego modelu teorii T.

□

# Przykład dowodu 4.5

W tym punkcie przedstawimy przykład dowodu poprawności pewnego programu w arytmetyce liczb rzeczywistych **R**. W dowodzie tym wyeksponujemy fragmenty rozumowania, w których stosuje się prawa logiki algorytmicznej. Pokażemy mianowicie, że program BP (ang. binary power) [7], jest poprawnym rozwiązaniem zadania „podnieść liczbę $x$ do potęgi naturalnej $n$" w strukturze liczb rzeczywistych.

```
BP:  {obliczyć x^n}
     begin
       z := x;
       y := 1;
       m := n;              {z^m * y = x^n}
       while m ≠ 0
       do                   {z^m * y = x^n}
         if odd(m)          {tzn. gdy m nieparzyste}
         then
           y := y * z
         fi;
         m := mdiv2;
         z := z * z;        {z^m * y = x^n}
       od
     end                    {z^m * y = x^n ∧ m = 0}
```

Dowód składa się z trzech części:

(1) najpierw udowodnimy, że instrukcja iterowana

```
M ≝ begin
        if odd(m) then y := y * z fi;
        m := mdiv2; z := z * z.
    end
```

jest częściowo poprawna, niezmiennicza względem warunku

$\delta \overset{\mathrm{df}}{=} (z^m * y = x^n)$

(2) następnie wykażemy, że program BP zatrzymuje się, jeśli $n$ jest liczbą naturalną;

(3) a na koniec posłużymy się regułą

$$\frac{(\delta \Rightarrow \mathrm{M}\delta)}{(\delta \wedge \textbf{while } \alpha \textbf{ do } \mathrm{M} \textbf{ od true}) \Rightarrow \textbf{while } \alpha \textbf{ do } \mathrm{M} \textbf{ od } (\neg\alpha \wedge \delta)}$$

(por. przykład 4.5), aby wyciągnąć ostateczny wniosek, że

$$\mathbf{R} \models \mathrm{BP}(y = x^n)$$

Ad (1)
Zaczniemy od udowodnienia przesłanki powyższej reguły. Mamy udowodnić następującą implikację

$$(z^m * y = x^n) \Rightarrow \mathrm{M}(z^m * y = x^n)$$

Ta formuła jest, na mocy aksjomatu Ax19, równoważna formule

$$(z^m * y = x^n) \Rightarrow \textbf{if } odd(m) \textbf{ then } y := y * z \textbf{ fi}((m := m div 2)((z := z * z)(z^m * y = x^n)))$$

Dwukrotnie stosując aksjomat Ax18 otrzymujemy równoważne formuły, najpierw

$$(z^m * y = x^n) \Rightarrow \textbf{if } odd(m) \textbf{ then } y := y * z \textbf{ fi}((m := m div 2)((z * z)^m * y = x^n))$$

a potem

$$(z^m * y = x^n) \Rightarrow \textbf{if } odd(m) \textbf{ then } y := y * z \textbf{ fi}((z * z)^{m div 2} * y = x^n)$$

Z kolei, po zastosowaniu aksjomatu Ax20 widzimy, że ostatnia formuła jest równoważna formule

$$(z^m * y = x^n) \Rightarrow \big(odd(m) \wedge (y := y * z)((z * z)^{m div 2} * y = x^n) \vee \neg odd(m) \wedge \wedge ((z * z)^{m div 2} * y = x^n)\big)$$

(Pomijamy wyrażenie $ok(odd(m))$ ponieważ jest ono równoważne **true**). Jeszcze raz stosujemy aksjomat Ax18 i przekształcamy tę formułę do równoważnej postaci

$$(z^m * y = x^n) \Rightarrow \big(odd(m) \wedge ((z * z)^{m div 2} * y * z = x^n) \vee \neg odd(m) \wedge \wedge ((z * z)^{m div 2} * y = x^n)\big).$$

Teraz odwołujemy się do znanych nam własności działań mnożenia, potęgowania, dzielenia całkowitoliczbowego *div* oraz do znaczenia predykatu *odd* (jest on spełniony przez te i tylko te liczby całkowite, które są nieparzyste)

$$(\neg odd(m) \Rightarrow 2 * m div 2 = m)$$
$$(odd(m) \Rightarrow 2 * m div 2 + 1 = m)$$

Z faktów tych, znanych nam z algebry szkolnej wynika, że oba człony alternatywy w następniku implikacji są równoważne jej poprzednikowi. Otrzymaliśmy więc zdanie, które jest prawem rachunku zdań

$$(z^m * y = x^n) \Rightarrow (z^m * y = x^n)$$

Udowodniliśmy tym samym prawdziwość formuły $(\delta \Rightarrow \mathrm{M}\delta)$.

Ad (2)
Przez indukcję ze względu na $i \in N$ pokażemy, że dla $m$ naturalnych

$$\mathbf{if}\ mdiv2 \neq 0\ \mathbf{then}\ m := mdiv2\ \mathbf{fi}^i\ mdiv2 = 0 \Rightarrow \Rightarrow \mathbf{if}\ m \neq 0\ \mathbf{then}\ m := mdiv2\ \mathbf{fi}^{i+1}\ m = 0 \tag{4.7}$$

$$0 \leqslant m < 2^i \Rightarrow \mathbf{if}\ m \neq 0\ \mathbf{then}\ m := mdiv2\ \mathbf{fi}^{i+1}\ m = 0 \tag{4.8}$$

Ponieważ następujące zdania

$$mdiv2 \neq 0 \Rightarrow m \neq 0$$
$$mdiv2 = 0 \Rightarrow (m = 0 \vee (m \neq 0 \wedge (m := mdiv2)\, m = 0))$$
$$m < 2^{i+1} \Rightarrow mdiv2 < 2^i$$

są prawdziwe w strukturze liczb rzeczywistych, zatem na mocy praw rachunku zdań i aksjomatu Ax20 mamy dla $i = 0$

$$0 \leqslant m < 1 \Rightarrow \mathbf{if}\ m \neq 0\ \mathbf{then}\ m := mdiv2\ \mathbf{fi}\ m = 0$$

oraz

$$mdiv2 = 0 \Rightarrow \mathbf{if}\ m \neq 0\ \mathbf{then}\ m := mdiv2\ \mathbf{fi}\ m = 0$$

Oznacza to, że obie własności (4.7) i (4.8) są w tym przypadku prawdziwe.

Załóżmy prawdziwość własności (4.7), (4.8) dla $i = k$.
Wykażemy ich słuszność dla $i = k+1$. Na mocy aksjomatu Ax20

$$\mathbf{if}\ mdiv2 \neq 0\ \mathbf{then}\ m := mdiv2\ \mathbf{fi}^{k+1}\ mdiv2 = 0 \Rightarrow$$
$$\Rightarrow ((mdiv2 \neq 0 \wedge (m := mdiv2)(\mathbf{if}\ mdiv2 \neq 0\ \mathbf{then}\ m := mdiv2\ \mathbf{fi})^k mdiv2 = 0)$$
$$\vee (mdiv2 = 0 \wedge (\mathbf{if}\ mdiv2 \neq 0\ \mathbf{then}\ m := mdiv2\ \mathbf{fi})^k mdiv2 = 0))$$

Stąd na mocy założenia indukcyjnego, własności operacji *div* i reguły odrywania

$$(\mathbf{if}\ mdiv2 \neq 0\ \mathbf{then}\ m := mdiv2\ \mathbf{fi})^{k+1} mdiv2 = 0 \Rightarrow$$
$$\Rightarrow (m \neq 0 \wedge (m := mdiv2)(\mathrm{K}^{k+1} m = 0) \vee m = 0 \wedge \mathrm{K}^{k+1} m = 0)$$

gdzie K oznacza program **if** $m \neq 0$ **then** $m := mdiv\ 2$ **fi**
Stosując jeszcze raz aksjomat Ax20 otrzymujemy

$$\mathbf{if}\ mdiv2 \neq 0\ \mathbf{then}\ m := mdiv2\ \mathbf{fi}^{k+1} mdiv2 = 0 \Rightarrow (\mathrm{K}^{k+2} m = 0)$$

W ten sposób zakończyliśmy dowód indukcyjny własności (4.7). Dla dowodu własności (4.8) zauważmy, że

$$0 \leqslant m < 2^{k+1} \Rightarrow 0 \leqslant mdiv2 < 2^k$$

Na mocy założenia indukcyjnego

$$mdiv2 < 2^k \Rightarrow \mathbf{if}\ mdiv2 \neq 0\ \mathbf{then}\ m := mdiv2\ \mathbf{fi}^{k+1}\ mdiv2 = 0$$

Z własności (4.7) oraz praw rachunku zdań mamy

$0 \leqslant m < 2^{k+1} \Rightarrow \textbf{if } m \neq 0 \textbf{ then } m := m\,div 2 \textbf{ fi}^{k+2} m = 0$

Wynika stąd, na mocy zasady indukcji matematycznej, że własność (4.8) zachodzi dla dowolnej liczby naturalnej $i$.

Wobec aksjomatu Ax21 (por. wzór (4.3)) oraz faktu, że w zbiorze liczb rzeczywistych $(\forall m)(\exists i) m < 2^i$, otrzymujemy ostatecznie prawdziwość formuły

**while** $m \neq 0$ **do** $m := m\,div 2$ **od true**

Ponieważ zarówno program **if** $odd(m)$ **then** $y := y * z$ **fi** jak i instrukcja przypisania $z := z * z$ nie zmieniają wartości zmiennej $m$, zatem stosując regułę pomocniczą (4.4) (por. przykład 4.10) wykażemy również prawdziwość formuły

**while** $m \neq 0$ **do** M **od true**

Ad (3)
Z reguły niezmiennika (por. przykład 4.5) dla $\alpha \stackrel{\text{df}}{=} m \neq 0$

$(\delta \wedge \textbf{while } \alpha \textbf{ do M od true}) \Rightarrow \textbf{while } \alpha \textbf{ do M od } (\neg\alpha \wedge \delta)$

Zatem dla K $\stackrel{\text{df}}{=}$ **begin** $z := x;\ y := 1;\ m := n$ **end**

$\text{K}((z^m * y = x^n) \wedge \textbf{while } \alpha \textbf{ do M od true}) \Rightarrow \text{BP}(y = x^n)$

Stosując aksjomat Ax18 otrzymamy w poprzedniku implikacji

$(x^n = x^n) \wedge \text{K}(\textbf{while } \alpha \textbf{ do M od true})$

Na mocy udowodnionej już części (2) formuła ta jest prawdziwa. Po zastosowaniu reguły odrywania R1 udowodnimy ostatecznie prawdziwość formuły

$\text{BP}(y = x^n)$

w strukturze liczb rzeczywistych. □

# Aksjomatyczna definicja semantyki 4.6

W jednym z poprzednich punktów pokazaliśmy jak ścisły jest związek między przyjętą semantyką a formalnym systemem logiki algorytmicznej. Obecnie zbadamy ten związek nieco głębiej.

Jeśli przyjrzymy się dokładniej aksjomatom logiki algorytmicznej, to zauważymy, że zasady semantyczne obliczania wartości formuły, czy zasady wykonywania obliczeń programu, mają swoje odpowiedniki w schematach aksjomatów. Pytanie, jakie sobie postawimy w tym punkcie, jest następujące:

W jakim stopniu aksjomatyzacja wyznacza sposób rozumienia programów, sposób ich wykonywania? Czy system formalny logiki algorytmicznej może być traktowany jako ścisła definicja semantyki deterministycznych **while**-programów?

Na semantykę języka algorytmicznego składają się trzy podstawowe elementy: interpretacja symboli funkcyjnych i relacyjnych, występujących w wyrażeniach poprawnych języka, interpretacja operatorów programotwórczych i interpretacja operatorów logicznych. W związku z tym przyjmijmy następującą definicję.

DEFINICJA 4.11

Strukturą semantyczną dla języka algorytmicznego L będziemy nazywać trójkę $\langle \mathbf{A}, \mathrm{I}, \models \rangle$, gdzie **A** jest strukturą danych, I — funkcją, która każdemu programowi przyporządkowuje dwuczłonową relację w zbiorze wszystkich wartościowań w **A**, a $\models$ jest relacją spełniania. ■

W dalszym ciągu ograniczymy nasze rozważania do klasy struktur semantycznych $\langle \mathbf{A}, \mathrm{I}, \models \rangle$ dla języka $L_=$ z równością spełniających następujące warunki (założenia):

(1) struktura danych jest ekspresywna, tzn. dla każdego elementu $a$ struktury istnieje term $\tau_a$ w języku L, którego wartością jest $a$, niezależnie od przyjętego wartościowania;

(2) interpretacja termów i formuł klasycznych jest klasyczna, tzn. zgodna z definicją podaną w p. 2.4;

(3) relacja spełniana ma dodatkowo następującą własność: dla dowolnego wartościowania $v$

$$\mathbf{A}, v \models \mathrm{M}\alpha \quad \text{wtw} \quad (\exists v')(v, v') \in \mathrm{I}(\mathrm{M}) \quad \text{oraz} \quad \mathbf{A}, v' \models \alpha$$

Zauważmy, że w porównaniu z semantyką opisaną w rozdz. 3, nie zakładamy żadnego konkretnego sposobu wykonywania obliczeń programu, żadnej ustalonej interpretacji dla występujących w języku konstrukcji programotwórczych. Przyjmujemy jedynie, że każdy program będzie wyznaczał relację binarną w zbiorze stanów pamięci, tzn. w zbiorze wartościowań.

Niech $\langle \mathbf{A}, \mathrm{I}, \models \rangle$ będzie strukturą semantyczną spełniającą warunki (1), (2), (3) (w dalszej części tego punktu będziemy je zawsze milcząco przyjmować). Wówczas jest spełniona następująca własność separowalności (odróżnialności).

LEMAT 4.5

Dla dowolnych wartościowań $v$, $v'$ w strukturze **A**,

$v \neq v'$ wtw istnieje formuła $\alpha$ taka, że $\mathbf{A}, v \models \alpha$ i $\mathbf{A}, v' \models \neg\alpha$

DOWÓD

Niech $v, v'$ będą dwoma różnymi wartościowaniami w strukturze $\mathbf{A}$ i $v \neq v'$. Istnieje zatem zmienna $z$ taka, że $v(z) \neq v'(z)$. Jeżeli $z$ jest zmienną zdaniową $q$, to przyjmując $\alpha = q$, gdy $v(q) = \mathbf{1}$ lub $\alpha = \neg q$, gdy $v(q) = \mathbf{0}$ otrzymamy $\mathbf{A}, v \models \alpha$ i $\mathbf{A}, v' \models \neg\alpha$. Jeżeli $z$ jest zmienną indywiduową $x$ oraz $v(x) = a$, to jako $\alpha$ przyjmijmy formułę $(\tau_a = x)$. Wtedy

$$\mathbf{A}, v \models (\tau_a = x) \text{ i } non\, \mathbf{A}, v' \models (\tau_a = x).$$

Odwrotnie, jeżeli $v = v'$, tzn. dla dowolnej zmiennej $z, v(z) = v'(z)$. Wtedy oczywiście dla dowolnej formuły $\alpha$

$$\mathbf{A}, v \models \alpha \quad \text{wtw} \quad \mathbf{A}, v' \models \alpha$$ □

DEFINICJA 4.12

Strukturę semantyczną $\langle \mathbf{A}, \mathrm{I}, \models \rangle$ nazwiemy standardową wtedy i tylko wtedy, gdy spełnione są następujące równości:

$$\mathrm{I}(x := w) = \{(v, v') : v'(x) = w_{\mathbf{A}}(v), v'(z) = v(z) \quad \text{dla} \quad z \neq x\}$$
$$\mathrm{I}(\textbf{begin } K; M \textbf{ end}) = \mathrm{I}(K) \circ \mathrm{I}(M)$$
$$\mathrm{I}(\textbf{if } \gamma \textbf{ then } K \textbf{ else } M \textbf{ fi}) = \mathrm{I}(K) \circ id_{\mathbf{A}}(\gamma) \cup \mathrm{I}(M) \circ id_{\mathbf{A}}(\neg\gamma)$$
$$\mathrm{I}(\textbf{while } \gamma \textbf{ do } K \textbf{ od}) = \bigcup \mathrm{I}(\textbf{if } \gamma \textbf{ then } K \textbf{ fi})^i \circ id_{\mathbf{A}}(\neg\gamma)$$

dla dowolnych $K, M \in \pi$, $\gamma \in F_0$, $x \in V$, $id_{\mathbf{A}}(\gamma) = \{(v, v) : \mathbf{A}, v \models (\gamma \wedge ok(\gamma))\}$ ■

LEMAT 4.6

Jeżeli $\langle \mathbf{A}, \mathrm{I}, \models \rangle$ jest standardową strukturą semantyczną, to dla każdego programu M i dowolnych wartościowań $v, v'$ w $\mathbf{A}$

$$(v, v') \in \mathrm{I}(M) \quad \text{wtw} \quad (v, v') \in M_{\mathbf{A}}$$

Dowód wynika z przyjętej definicji oraz z rozważań przeprowadzonych w rozdz. 3 (por. p. 3.2). ■

DEFINICJA 4.13

Struktura semantyczna $\langle \mathbf{A}, \mathrm{I}, \models \rangle$ jest modelem systemu $AL(\pi)$ wtedy i tylko wtedy, gdy dla dowolnego aksjomatu $\alpha$, $\mathbf{A} \models \alpha$ oraz dla dowolnej reguły systemu $AL(\pi)$, z prawdziwości przesłanek w strukturze danych $\mathbf{A}$, wynika prawdziwość wniosku. ■

Jako wniosek z lematu 4.6 i twierdzenia 4.1 o słuszności aksjomatyzacji przyjmujemy następujący fakt.

TWIERDZENIE 4.10

Każda standardowa struktura semantyczna jest modelem dla AL(π). ■

W dalszym ciągu zajmiemy się badaniem jakie własności interpretacji I wymuszają poszczególne aksjomaty logiki algorytmicznej.

LEMAT 4.7

Jeżeli struktura semantyczna $\langle \mathbf{A}, \mathrm{I}, \models \rangle$ jest modelem aksjomatu Ax16:

$$\mathrm{M}(\alpha \wedge \beta) \equiv (\mathrm{M}\alpha \wedge \mathrm{M}\beta),$$

to dla dowolnego programu M, I(M) jest funkcją częściową.

DOWÓD

Przypuśćmy, że dla pewnych wartościowań $v$, $v'$, $v''$ mamy $v'' \neq v'$, $(v, v'') \in \mathrm{I}(\mathrm{M})$ i $(v, v') \in \mathrm{I}(\mathrm{M})$. Na mocy lematu 4.5 istnieje formuła $\alpha$ taka, że $\mathbf{A}, v'' \models \alpha$ i $\mathbf{A}, v' \models \neg\alpha$. Zatem $\mathbf{A}, v \models \mathrm{M}\alpha$ oraz $\mathbf{A}, v \models \mathrm{M}\neg\alpha$. Na mocy aksjomatu Ax16 musi być $\mathbf{A}, v \models \mathrm{M}(\alpha \wedge \neg\alpha)$, co jest niemożliwe. Pokazaliśmy więc, że dla każdego $v$ istnieje co najwyżej jedno $v'$ takie, że $(v, v') \in \mathrm{I}(\mathrm{M})$, tzn. że I(M) jest funkcją częściową dla każdego M. □

LEMAT 4.8

Jeżeli struktura semantyczna $\langle \mathbf{A}, \mathrm{I}, \models \rangle$ jest modelem dla aksjomatu Ax16 i aksjomatu Ax19:

$$\textbf{begin } \mathrm{K};\ \mathrm{M} \textbf{ end } \alpha \equiv \mathrm{K}(\mathrm{M}\alpha),$$

to dla dowolnych programów K i M

$$\mathrm{I}(\textbf{begin } \mathrm{K};\ \mathrm{M} \textbf{ end}) = \mathrm{I}(\mathrm{K}) \circ \mathrm{I}(\mathrm{M}).$$

DOWÓD

Niech $(v, v') \in \mathrm{I}(\textbf{begin } \mathrm{K};\ \mathrm{M} \textbf{ end})$ i niech $\mathbf{A}, v' \models \alpha$. Zatem

$$\mathbf{A}, v' \models \textbf{begin } \mathrm{K};\ \mathrm{M} \textbf{ end } \alpha$$

Na mocy aksjomatu Ax19 mamy $\mathbf{A}, v' \models \mathrm{K}(\mathrm{M}\alpha)$. Z definicji struktury semantycznej wynika istnienie wartościowań $v_1$, $v_2$ takich, że $(v, v_1) \in \mathrm{I}(\mathrm{K})$, $(v_1, v_2) \in \mathrm{I}(\mathrm{M})$ oraz $\mathbf{A}, v_2 \models \alpha$. Zgodnie z lematem 4.7 wartościowania $v_1$ i $v_2$ są wyznaczone jednoznacznie i niezależnie od formuły $\alpha$. Stąd rozumowanie to można powtórzyć dla dowolnej formuły $\alpha$, dowodząc tym samym, że $\mathbf{A}, v' \models \alpha$

implikuje $A, v_2 \models \alpha$. Zatem z lematu 4.5 mamy $v_2 = v'$ i tym samym $(v, v_1) \in I(K)$, $(v_1, v') \in I(M)$. W konsekwencji $(v, v') \in I(K) \circ I(M)$.

Inkluzji przeciwnej dowodzi się analogicznie. □

LEMAT 4.9

Jeżeli struktura semantyczna $\langle A, I, \models \rangle$ jest modelem dla aksjomatów Ax19, Ax16 i aksjomatu Ax20:

$$\textbf{if } \gamma \textbf{ then else } M \textbf{ fi } \alpha \equiv (ok(\gamma) \wedge ((\neg\gamma \wedge K\alpha) \vee (\gamma \wedge M\alpha))),$$

to dla dowolnych programów K, M i dowolnej formuły otwartej $\gamma$

$$I(\textbf{if } \gamma \textbf{ then } K \textbf{ else } M \textbf{ fi}) = I(K) \circ id_A(\gamma) \cup I(M) \circ id_A(\neg\gamma)$$

Dowód pomijamy, ponieważ jest analogiczny do przedstawionych w poprzednich przypadkach. ■

LEMAT 4.10

Jeżeli struktura semantyczna $\langle A, I, \models \rangle$ jest modelem dla aksjomatów Ax16, Ax19, Ax20 i aksjomatu Ax21:

$$\textbf{while } \gamma \textbf{ do } K \textbf{ od } \alpha \equiv (ok(\gamma) \wedge ((\neg\gamma \wedge \alpha) \vee (\gamma \wedge K(\textbf{while } \gamma \textbf{ do } K \textbf{ od } \alpha))))$$

to dla dowolnej formuły $\gamma$ i dowolnego programu M

$$\bigcup I(\textbf{if } \gamma \textbf{ then } M \textbf{ fi})^i \circ id_A(\neg\gamma) \subset I(\textbf{while } \gamma \textbf{ do } M \textbf{ od})$$

DOWÓD

Przypuśćmy, że $(v, v') \in \bigcup I(\textbf{if } \gamma \textbf{ then } M \textbf{ fi})^i \circ id_A(\neg\gamma)$. Zatem istnieje liczba naturalna $m$ taka, że

$$A, v' \models (\neg\gamma \wedge ok(\gamma)) \quad \text{oraz} \quad (v, v') \in I(\textit{if } \gamma \textbf{ then } M \textbf{ fi})^m$$

Załóżmy teraz, że dla pewnej formuły $\alpha$, $A, v' \models \alpha$. Mamy wtedy $A, v \models (\textbf{if } \gamma \textbf{ then } M \textbf{ fi})^m (\alpha \wedge \neg\gamma \wedge ok(\gamma))$. Na mocy aksjomatu Ax21

$$A, v \models \textbf{while } \gamma \textbf{ do } M \textbf{ od } \alpha$$

W konsekwencji istnieje wartościowanie $v''$, wyznaczone (na mocy lematu 4.7) jednoznacznie, takie, że

$$(v, v'') \in I(\textbf{while } \gamma \textbf{ do } M \textbf{ od}) \quad \text{i} \quad A, v'' \models \alpha$$

Ponieważ rozumowanie to możemy powtórzyć dla dowolnej formuły $\alpha$ (przy tym samym $v''$), to na mocy lematu 4.5 $v' = v''$. Stąd ostatecznie mamy $(v, v') \in I(\textbf{while } \gamma \textbf{ do } M \textbf{ od})$, co należało pokazać. □

LEMAT 4.11

Jeżeli struktura semantyczna $\langle \mathbf{A}, \mathrm{I}, \models \rangle$ jest modelem dla aksjomatów Ax16, Ax19, Ax20, Ax21 i reguła R3 jest poprawna w **A**, to

$$\mathrm{I}(\textbf{while } \gamma \textbf{ do } \mathrm{K} \textbf{ od}) = \bigcup \mathrm{I}(\textbf{if } \gamma \textbf{ then } \mathrm{K} \textbf{ fi})^i \circ id_{\mathbf{A}}(\neg\gamma)$$

DOWÓD

Niech $(v, v') \in \mathrm{I}(\textbf{while } \gamma \textbf{ do } \mathrm{M} \textbf{ od})$ i niech $\mathbf{A}, v' \models \alpha$. Załóżmy, że $x_1, x_2, \ldots, x_k$ są wszystkimi zmiennymi indywiduowymi, $q_1, \ldots, q_l$ wszystkimi zmiennymi zdaniowymi występującymi w programie **while** $\gamma$ **do** M **od** i w formule $\alpha$. Co więcej załóżmy, że $v(x_j) = a_j$ dla $j < k+1$. Niech $\beta$ będzie formułą opisującą wartościowanie $v$ dla zmiennych $x_1, x_2, \ldots, x_k$, $q_1, \ldots, q_l$, tzn.

$$\beta = (x_1 = \tau_{a1}) \wedge (x_2 = \tau_{a2}) \wedge \ldots \wedge (x_k = \tau_{ak}) \wedge q_1' \wedge \ldots \wedge q_l'$$

gdzie $q_i'$ oznacza $q_i$, gdy $v(q_i) = \mathbf{1}$ oraz oznacza $\neg q_i$, gdy $v(q_i) = \mathbf{0}$. Stąd $\mathbf{A}, v \models \beta$. Ponieważ $\mathbf{A}, v \models \textbf{while } \gamma \textbf{ do } \mathrm{M} \textbf{ od } \alpha$ zatem

$$non\ \mathbf{A} \models (\textbf{while } \gamma \textbf{ do } \mathrm{M} \textbf{ od } \alpha \Rightarrow \neg\beta)$$

Na mocy reguły R3 istnieje taka liczba naturalna $m$, że

$$non\ \mathbf{A} \models (\textbf{if } \gamma \textbf{ then } \mathrm{M} \textbf{ fi})^m (\alpha \wedge \neg\gamma \wedge ok(\gamma)) \Rightarrow \neg\beta$$

Istnieje zatem wartościowanie $v''$ takie, że

$$\mathbf{A}, v'' \models (\textbf{if } \gamma \textbf{ then } \mathrm{M} \textbf{ fi})^m (\alpha \wedge \neg\gamma \wedge ok(\gamma)) \text{ i } \mathbf{A}, v'' \models \beta$$

a więc na mocy definicji formuły $\beta$

$$\mathbf{A}, v \models (\textbf{if } \gamma \textbf{ then } \mathrm{M} \textbf{ fi})^m (\alpha \wedge \neg\gamma \wedge ok(\gamma))$$

Załóżmy, że $m$ jest najmniejszą liczbą naturalną o tej własności. Istnieje więc wartościowanie $v'''$ takie, że

$$(v, v''') \in \mathrm{I}(\textbf{if } \gamma \textbf{ then } \mathrm{M} \textbf{ fi})^m \quad \text{i} \quad \mathbf{A}, v''' \models (\alpha \wedge \neg\gamma \wedge ok(\gamma))$$

Rozważmy teraz dowolną formułę $\alpha'$, dla której $\mathbf{A}, v' \models \alpha'$. Pokażemy, że $\mathbf{A}, v''' \models (\alpha' \wedge \neg\gamma \wedge ok(\gamma))$. Naśladując prezentowane poprzednio postępowanie otrzymamy

$$\mathbf{A}, v \models (\textbf{if } \gamma \textbf{ then } \mathrm{M} \textbf{ fi})^j (\alpha' \wedge \neg\gamma \wedge ok(\gamma))$$

dla pewnej liczby naturalnej $j$. Na mocy założenia o $m$ i aksjomatu Ax20 musi być $m \leqslant j$. W konsekwencji $\mathbf{A}, v \models (\textbf{if } \gamma \textbf{ then } \mathrm{M} \textbf{ fi})^m (\alpha' \wedge \neg\gamma \wedge ok(\gamma))$. Na mocy lematu 4.7 i definicji wartościowania $v'''$ mamy

$$\mathbf{A}, v''' \models (\alpha' \wedge \neg\gamma \wedge ok(\gamma))$$

... liczba naturalna $m$ i wartościowanie

$v'''$ takie, że $(v, v''') \in \mathrm{I}(\textbf{if } \gamma \textbf{ then } \mathrm{M} \textbf{ fi})^m$ i dla dowolnej formuły $\alpha'$, jeśli $\mathbf{A}, v' \models \alpha'$, to $\mathbf{A}, v''' \models (\alpha' \wedge \neg\gamma \wedge ok(\gamma))$. Na mocy lematu 4.5 $v''' = v'$ z czego natychmiast wynika

$$(v, v') \in \mathrm{I}(\textbf{if } \gamma \textbf{ then } \mathrm{M} \textbf{ fi})^m \circ id_{\mathbf{A}}(\neg\gamma). \qquad \square$$

LEMAT 4.12

Jeżeli struktura $\langle \mathbf{A}, \mathrm{I}, \models \rangle$ jest modelem dla aksjomatu Ax18:

$$(z := w)\,\gamma(z) \equiv (\gamma(z/w) \wedge ok(w)),$$

to dla dowolnej zmiennej $z$ i dowolnego wyrażenia $w$ (termu lub formuły otwartej w zależności od typu zmiennej $z$)

$$\mathrm{I}(z := w) = \{(v, v') : w_{\mathbf{A}}(v) \text{ jest zdefiniowane oraz } v'(z) = w_{\mathbf{A}}(v),\ v'(u) = v(u) \text{ dla } u \neq z\}$$

DOWÓD

Rozważmy przypadek, gdy $z$ jest zmienną indywiduową $x$, a $w$ jest termem $\tau$. Niech $(v, v') \in \mathrm{I}(x := \tau)$ i niech $v'(x) = a$ oraz $v'(y) = b$ dla pewnej zmiennej indywidualnej $y \neq x$. Na mocy założenia (1) istnieją termy $\tau_a$ i $\tau_b$ bez zmiennych wolnych takie, że

$$\mathbf{A}, v' \models (x = \tau_a) \wedge (y = \tau_b)$$

Zatem

$$\mathbf{A}, v \models (x := \tau)((x = \tau_a) \wedge (y = \tau_b))$$

Korzystając z aksjomatu Ax18 otrzymujemy

$$\mathbf{A}, v \models ((\tau = \tau_a) \wedge (y = \tau_b) \wedge ok(\tau))$$

co na mocy ustaleń o strukturze semantycznej oznacza, że $\tau_{\mathbf{A}}(v)$ jest określone i $\tau_{\mathbf{A}}(v) = \tau_{a\mathbf{A}}(v) = a = v'(x)$ oraz $v(y) = \tau_{b\mathbf{A}}(v) = b = v'(y)$. Powtarzając to rozumowanie dla dowolnej zmiennej $z \neq x$ udowodnimy inkluzję $\subset$.

Dla dowodu inkluzji przeciwnej niech $\tau_{\mathbf{A}}(v)$ będzie określone oraz $v'(x) = a = \tau_{\mathbf{A}}(v)$ i $v'(y) = v(y) = b$. Na mocy założeń o strukturze semantycznej istnieją termy $\tau_a$ i $\tau_b$ definiujące w języku L stałe $a$ i $b$. Zatem

$$\mathbf{A}, v' \models ((x = \tau_a) \wedge (y = \tau_b) \wedge ok(\tau))$$

Przyjmijmy jako $\gamma$ w aksjomacie Ax18 formułę $(x = \tau_a) \wedge (y = \tau_b)$. Wtedy mamy $\mathbf{A}, v \models (x := \tau)((x = \tau_a) \wedge (y = \tau_b))$. Stąd na mocy definicji spełniania w strukturze semantycznej istnieje $v''$ takie, że

$$(v, v'') \in \mathrm{I}(x := \tau) \quad \text{oraz} \quad \mathbf{A}, v'' \models ((x = \tau_a) \wedge (y = \tau_b))$$

czyli

$$v''(a) = \tau_a \quad \text{i} \quad v''(y) = \tau_b$$

Zgodnie z przyjętymi oznaczeniami mamy więc $v''(x) = v'(x)$ i $v''(y) = v'(y)$. Powtarzając rozumowanie dla dowolnej zmiennej $z \neq x$ otrzymamy $v'' = v'$, a zatem $(v, v') \in \mathrm{I}(x := \tau)$. Analogiczny dowód dla instrukcji przypisania postaci $(q := \gamma)$ pomijamy. □

UWAGA

Zauważmy, że tylko lematy 4.11, 4.12 wykorzystywały w dowodzie założenie (1) o ekspresywności struktury semantycznej. Lematy 4.7–4.10 można udowodnić bez tego założenia. ■

Jako wniosek z udowodnionych faktów otrzymujemy zasadnicze twierdzenie tego punktu.

TWIERDZENIE 4.11

Każda struktura semantyczna będąca modelem logiki algorytmicznej jest strukturą standardową. ■

Wykazaliśmy w ten sposób, że aksjomaty logiki AL(π) wymuszają standardową interpretację konstrukcji programotwórczych. Wobec lematu 4.6, udowodniliśmy równoważność pojęcia modelu systemu AL(π) (w klasie modeli spełniających założenie (1)) i struktury standardowej. Tym samym pokazaliśmy (por. lemat 4.5), że aksjomaty logiki algorytmicznej AL(π) jednoznacznie wyznaczają semantykę programów należących do klasy π-deterministycznych programów iteracyjnych. Co więcej, jest to dokładnie ta sama semantyka, którą opisano wcześniej w rozdz. 3.

## Czy można zautomatyzować dowodzenie twierdzeń?

4.7

To pytanie od lat nurtuje logików i matematyków. Odpowiedź zależy oczywiście od tego, z jakim systemem formalnym mamy do czynienia i jakiego typu własności chcemy dowodzić. W przypadku klasycznego rachunku zdań odpowiedź jest twierdząca. Rachunek zdań jest rozstrzygalny. Istnieje algorytm, który o każdej formule zbudowanej jedynie ze zmiennych zdaniowych i spójników logicznych, stwierdza po skończonej liczbie kroków, czy formuła jest, czy nie jest, twierdzeniem rachunku zdań.

Najbardziej znaną metodą rozstrzygania dla rachunku zdań jest metoda zerojedynkowa polegająca na sprawdzeniu wszystkich możliwych danych. Nawet tak prosta metoda jest kosztowna. Co więcej, okazuje się, że koszt dowolnej metody rośnie wykładniczo wraz z długością formuły [1].

W przypadku logiki klasycznej pierwszego rzędu odpowiedź na nasze pytanie jest jeszcze bardziej złożona. Logika pierwszego rzędu nie jest rozstrzygalna. Nie istnieje jednolita metoda rozpoznawania czy dowolna formuła pierwszego rzędu jest, czy nie jest, twierdzeniem. Wprawdzie dowód jest w logice klasycznej skończonym ciągiem znaków i w procesie wypisywania wszystkich dowodów na pewno trafimy na dowód rozważanej formuły, jeżeli jest ona rzeczywiście twierdzeniem. Jednak, gdy rozważana formuła nie jest twierdzeniem, taki proces jest nieskończony. Po żadnej skończonej liczbie wypisanych dowodów nie można być pewnym, że formuła nie ma w ogóle dowodu formalnego. Ponadto, wypisywanie lub przeglądanie wszystkich dowodów twierdzeń, nawet całkowicie automatyczne, jest w praktyce niewykonalne ze względu na czas potrzebny do realizacji takiego przedsięwzięcia.

Logika algorytmiczna AL(π) jest również systemem nierozstrzygalnym [31]. Co więcej, dowód formuły nie musi być skończony; drzewo dowodu może mieć wierzchołki rzędu nieskończonego (por. p. 4.2).

Zdając sobie sprawę z tych obiektywnych trudności, czy należy zrezygnować z prac zmierzających do konstrukcji systemów automatycznego dowodzenia twierdzeń? Oczywiście nie. Badania w tej dziedzinie mogą przynieść ciekawe wyniki teoretyczne, prace eksperymentalne zaś mogą przyczynić się do powstania programów typu kalkulator dla obliczeń symbolicznych i wnioskowań logicznych. Istnieje wiele interesujących systemów, które wspomagają dowodzenie. Stworzono programy komputerowe, oparte w przeważającej większości na metodzie rezolucji [11] lub metodzie Gentzena [20,33], które znalazły zastosowanie w dydaktyce. Eksperymenty z tymi programami rokują nadzieję na włączenie ich do systemów wspomagających pracę programisty, obok edytorów tekstu, kompilatorów i programów monitorujących obliczenie (ang. debagger) [20, 11].

W tym punkcie przedstawimy system formalny GAL oparty na idei Gentzena. Proces dowodu w systemie GAL polega na rozkładzie danej formuły na podformuły zgodnie z pewnymi prawami rozkładu. Wybór reguły rozkładu jest jednoznacznie wyznaczony przez postać rozkładanej formuły. W ten sposób każda formuła wyznacza jednoznacznie i algorytmicznie pewne drzewo etykietowane formułami. Gdy wszystkie drogi w tym drzewie są skończone, daje ono albo formalny dowód formuły, albo kontrprzykład. Mamy więc do czynienia z uniwersalnym (stosowalnym do każdej formuły) algorytmem budowy dowodu formuły. Algorytm ten jednak nie zawsze kończy swoje obliczenia.

Niech L(π) będzie językiem algorytmicznym programów iteracyjnych (por. p. 3.5) z tym, że dla uproszczenia rozważań pominiemy kwantyfikatory (zarówno iteracji, jak i kwantyfikatory klasyczne). Co więcej, będziemy rozważać jedynie klasę struktur danych, w której wszystkie funktory są interpretowane jako funkcje całkowite.

W całym podrozdziale będziemy stosować konsekwentnie następujące

oznaczenia: $\mathbf{\Gamma}$, $\mathbf{\Omega}$ będą oznaczać skończone ciągi formuł języka $L(\pi)$, zwykle $\mathbf{\Gamma} = \alpha_1, \ldots, \alpha_n$, $\mathbf{\Omega} = \beta_1, \ldots, \beta_m$, dla pewnych $n, m \in N$. Wyrażenie postaci $\mathbf{\Gamma} \to \mathbf{\Omega}$ będziemy nazywać *sekwentem*. Napis $(\bigwedge \mathbf{\Gamma} \Rightarrow \bigvee \mathbf{\Omega})$ będzie skrótem dla formuły

$$(\alpha_1 \wedge (\alpha_2 \wedge \ldots \wedge \alpha_n) \ldots) \Rightarrow (\beta_1 \vee (\beta_2 \vee \ldots \vee \ldots \vee \beta_m) \ldots)$$

jeżeli oba ciągi $\mathbf{\Gamma}$ i $\mathbf{\Omega}$ są niepuste. Jeżeli $\mathbf{\Gamma}$ jest ciągiem pustym, to $\bigwedge \mathbf{\Gamma}$ oznacza stałą **true**, a jeżeli $\mathbf{\Omega}$ jest ciągiem pustym, to $\bigvee \mathbf{\Omega}$ oznacza stałą **false**. O formule $(\bigwedge \mathbf{\Gamma} \Rightarrow \bigvee \mathbf{\Omega})$ będziemy mówili, że *odpowiada sekwentowi* $\mathbf{\Gamma} \to \mathbf{\Omega}$.

DEFINICJA 4.14

*Sekwent* $\mathbf{\Gamma} \to \mathbf{\Omega}$ nazywamy *nierozkładalnym* wtedy i tylko wtedy, gdy każda formuła występująca w zbiorze $\mathbf{\Gamma} \cup \mathbf{\Omega}$ jest albo zmienną zdaniową, albo formułą elementarną. Takie formuły nazywamy nierozkładalnymi.

Sekwent $\mathbf{\Gamma} \to \mathbf{\Omega}$ nazwiemy *aksjomatem systemu* GAL wtedy i tylko wtedy, gdy zbiory $\mathbf{\Gamma}$ i $\mathbf{\Omega}$ nie są rozłączne. ■

System GAL jest wyznaczony przez zbiór aksjomatów i zbiór reguł rozkładu RP. Ogólna postać reguł rozkładu jest następująca:

$$\frac{\{\mathbf{\Gamma}_j \to \mathbf{\Omega}_j\}_{j \in J}}{\mathbf{\Gamma} \to \mathbf{\Omega}} \tag{4.9}$$

Sekwent $\mathbf{\Gamma} \to \mathbf{\Omega}$ nazywamy wnioskiem lub konkluzją, a sekwenty $\mathbf{\Gamma}_j \to \mathbf{\Omega}_j$ przesłankami reguły rozkładu.

*Reguły rozkładu RR dla systemu* GAL

GRUPA POPRZEDNIKA

$$rr1 \quad \frac{\mathbf{\Gamma}', \mathbf{\Gamma}, \mathrm{s}s\gamma \to \mathbf{\Omega}}{\mathbf{\Gamma}', \mathrm{s}(s\gamma), \mathbf{\Gamma} \to \mathbf{\Omega}}$$

$$rr2 \quad \frac{\mathbf{\Gamma}', \mathbf{\Gamma} \to \mathbf{\Omega}, \mathrm{s}\alpha}{\mathbf{\Gamma}', \mathrm{s}\ \neg\alpha, \mathbf{\Gamma} \to \mathbf{\Omega}}$$

$$rr3 \quad \frac{\mathbf{\Gamma}', \mathbf{\Gamma}, \mathrm{s}\alpha, \mathrm{s}\beta \to \mathbf{\Omega}}{\mathbf{\Gamma}', \mathrm{s}(\alpha \wedge \beta), \mathbf{\Gamma} \to \mathbf{\Omega}}$$

$$rr4 \quad \frac{\mathbf{\Gamma}', \mathbf{\Gamma}, \mathrm{s}\alpha \to \mathbf{\Omega}; \mathbf{\Gamma}', \mathbf{\Gamma}, \mathrm{s}\beta \to \mathbf{\Omega}}{\mathbf{\Gamma}', \mathrm{s}(\alpha \vee \beta), \mathbf{\Gamma} \to \mathbf{\Omega}}$$

$$rr6 \quad \frac{\mathbf{\Gamma}', \mathbf{\Gamma}, \mathrm{s}(\mathrm{K}(\mathrm{M}\alpha)) \to \mathbf{\Omega}}{\mathbf{\Gamma}', \mathrm{s}(\mathbf{begin}\ \mathrm{K}; \mathrm{M}\ \mathbf{end}\ \alpha), \mathbf{\Gamma} \to \mathbf{\Omega}}$$

$$rr7\quad \frac{\Gamma', \Gamma, \mathrm{s}(\gamma \wedge \mathrm{K}\alpha) \rightarrow \Omega;\ \Gamma', \Gamma, \mathrm{s}(\neg\gamma \wedge \mathrm{M}\alpha) \rightarrow \Omega}{\Gamma', \mathrm{s}(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{K}\ \mathbf{else}\ \mathrm{M}\ \mathbf{fi}\ \alpha), \Gamma \rightarrow \Omega}$$

$$rr8\quad \frac{\{\Gamma', \Gamma, \mathrm{s}(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{M}\ \mathbf{fi})^i(\neg\gamma \wedge \alpha) \rightarrow \Omega\}_{i \in N}}{\Gamma', \mathrm{s}(\mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha), \Gamma \rightarrow \Omega}$$

## Grupa Następnika

$$rr1\quad \frac{\Gamma \rightarrow \Omega', \Omega, \mathrm{s}s\gamma}{\Gamma \rightarrow \Omega', \mathrm{s}(s\gamma), \Omega}$$

$$rr2\quad \frac{\Gamma, \mathrm{s}\alpha \rightarrow \Omega', \Omega}{\Gamma \rightarrow \Omega', \mathrm{s}\neg\alpha, \Omega}$$

$$rr3\quad \frac{\Gamma \rightarrow \Omega', \Omega, \mathrm{s}\alpha;\ \Gamma \rightarrow \Omega', \Omega, \mathrm{s}\beta}{\Gamma \rightarrow \Omega', \mathrm{s}(\alpha \wedge \beta), \Omega}$$

$$rr4\quad \frac{\Gamma \rightarrow \Omega', \Omega, \mathrm{s}\alpha, \mathrm{s}\beta}{\Gamma \rightarrow \Omega', \mathrm{s}(\alpha \vee \beta), \Omega}$$

$$rr5\quad \frac{\Gamma, \mathrm{s}\alpha \rightarrow \Omega', \Omega, \mathrm{s}\beta}{\Gamma \rightarrow \Omega', \mathrm{s}(\alpha \Rightarrow \beta), \Omega}$$

$$rr6\quad \frac{\Gamma \rightarrow \Omega', \Omega, \mathrm{s}(\mathrm{K}(\mathrm{M}\alpha))}{\Gamma \rightarrow \Omega', \mathrm{s}(\mathbf{begin}\ \mathrm{K};\ \mathrm{M}\ \mathbf{end}\ \alpha), \Omega}$$

$$rr7\quad \frac{\Gamma \rightarrow \Omega', \Omega, \mathrm{s}(\gamma \wedge \mathrm{K}\alpha), \mathrm{s}(\neg\gamma \wedge \mathrm{M}\alpha)}{\Gamma \rightarrow \Omega', \mathrm{s}(\mathbf{if}\ \gamma\ \mathbf{then}\ \mathrm{K}\ \mathbf{else}\ \mathrm{M}\ \mathbf{fi}\ \alpha), \Omega}$$

$$rr8\quad \frac{\Gamma \rightarrow \Omega', \Omega, \mathrm{s}(\neg\gamma \wedge \alpha), \mathrm{s}(\gamma \wedge \mathrm{M}(\mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha))}{\Gamma \rightarrow \Omega', \mathrm{s}(\mathbf{while}\ \gamma\ \mathbf{do}\ \mathrm{M}\ \mathbf{od}\ \alpha), \Omega}$$

We wszystkich przedstawionych schematach $\Gamma'$, $\Omega'$ oznaczają ciągi formuł nierozkładalnych, $\Gamma$, $\Omega$ — ciągi dowolnych formuł, s dowolny ciąg instrukcji przypisania, $s$ — dowolną instrukcję przypisania, $\alpha$, $\beta$ dowolne formuły, $\gamma$ dowolną formułę otwartą, a M, M′ dowolne programy klasy $\pi$.

LEMAT 4.13

Rozważmy dowolną regułę systemu GAL postaci (4.9). Dla dowolnej struktury danych **A** języka L($\pi$) i dla dowolnego wartościowania $v$ w **A** zachodzi następująca równoważność:

$$\mathbf{A}, v \models (\bigwedge \Gamma \Rightarrow \bigvee \Omega) \quad \text{wtw} \quad (\forall j \in J)\, \mathbf{A}, v \models (\bigwedge \Gamma j \Rightarrow \bigvee \Omega j) \qquad \blacksquare$$

Dowód lematu wynika z twierdzenia o słuszności aksjomatyzacji (por. p. 4.2). Pamiętajmy jednak o ograniczeniach tego paragrafu: język nie zawiera kwantyfikatorów, a w strukturze danych wszystkie operacje są całkowite. Wynika stąd w szczególności, że we wszystkich rozważanych tu strukturach **A** i dla dowolnego termu $\tau$ mamy $\mathbf{A} \models ok(\tau)$.

DEFINICJA 4.15

Diagramem formuły $\alpha$ nazywamy parę uporządkowaną $\langle D, d \rangle$, gdzie $D$ jest drzewem, a $d$ jest odwzorowaniem, które każdemu wierzchołkowi drzewa $D$ przyporządkowuje etykietę będącą niepustym sekwentem. Drzewo $D$ i odwzorowanie $d$ są zdefiniowane przez indukcję ze względu na poziom $l$ drzewa $D$ następująco:

(1) Jedynym wierzchołkiem poziomu $l = 0$ jest $\emptyset$ (korzeń drzewa $D$) a $d(\emptyset) = \rightarrow \alpha$.

(2) Załóżmy, że drzewo $D$ i odwzorowanie $d$ zdefiniowano aż do poziomu $l \leqslant n$. Niech $w$ będzie wierzchołkiem na poziomie $n$. Jeżeli $d(w)$ jest sekwentem nierozkładalnym lub aksjomatem wtedy wierzchołek $w$ jest liściem drzewa $D$. W przeciwnym razie załóżmy, że $w = (i_1, ..., i_n)$ i $d(w) = \mathbf{\Gamma} \rightarrow \mathbf{\Omega}$ i rozważmy dwa przypadki w zależności od tego czy rozważany poziom $n$ jest liczbą parzystą, czy nieparzystą. Załóżmy, że

$$\boldsymbol{\sigma} = \begin{cases} \mathbf{\Gamma} & \text{gdy } n \text{ jest liczbą parzystą} \\ \mathbf{\Omega} & \text{gdy } n \text{ jest liczbą nieparzystą} \end{cases}$$

Jeżeli $\boldsymbol{\sigma}$ zawiera jedynie formuły nierozkładalne, to jedynym synem (następnikiem) wierzchołka $w$ jest wierzchołek

$$w0 = (i_1, ..., i_n, 0)$$

a etykietą tego wierzchołka jest $d(w0) = d(w)$.

Jeżeli $\boldsymbol{\sigma}$ zawiera chociaż jedną formułę rozkładalną, to $d(w)$ jest wnioskiem w pewnej regule rozkładu należącej do zbioru RR. Niech to będzie reguła postaci (4.9). Wtedy dla każdego $j \in J$

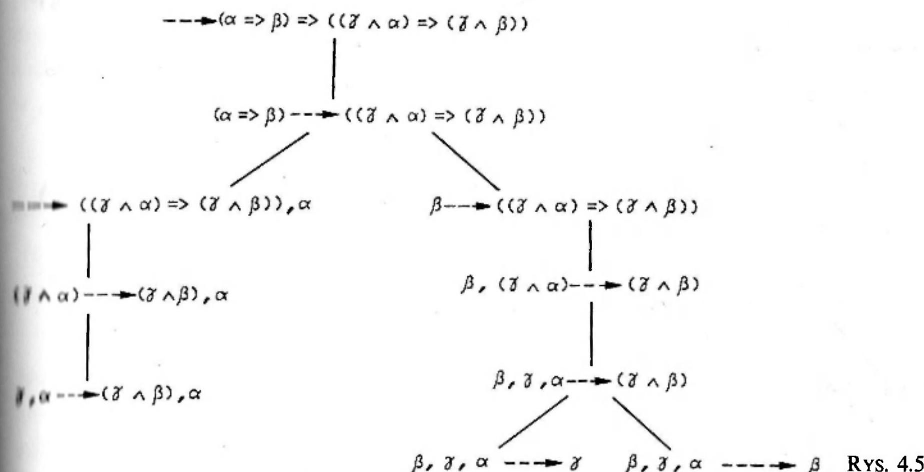
$$wj = (i_1, ..., i_n, j)$$

jest wierzchołkiem w drzewie $D$, dokładniej, jest synem wierzchołka $w$ oraz $d(wj)$ jest równe $j$-tej przesłance rozważanej reguły, tzn. $d(wj) = \mathbf{\Gamma}_j \rightarrow \mathbf{\Omega}_j$. ■

UWAGA

Diagram formuły budujemy rozkładając na przemian albo formułę z poprzednika sekwentu, jeśli jest on etykietą wierzchołka na poziomie nieparzystym, albo formułę z następnika sekwentu, jeśli jest on etykietą wierzchołka na

poziomie parzystym. Takie postępowanie zapewnia pewnego rodzaju sprawiedliwość rozkładu. Ponadto, ponieważ każda reguła wskazuje jednoznacznie sytuację, w której może być zastosowana, możemy stwierdzić, że diagram formuły jest przez nią jednoznacznie wyznaczony. Założenie to jest przydatne, jeśli chcemy tworzyć diagram automatycznie, natomiast nie jest konieczne i czasami nawet niewygodne, gdy sami budujemy dowód formalny. ■



RYS. 4.5

PRZYKŁAD 4.14

Rysunek 4.5 przedstawia diagram formuły

$$(\alpha \Rightarrow \beta) \Rightarrow ((\gamma \wedge \alpha) \Rightarrow (\gamma \wedge \beta))$$

gdzie $\alpha$, $\beta$, $\gamma$ są dowolnymi formułami.
Rysunek 4.6 przedstawia diagram formuły

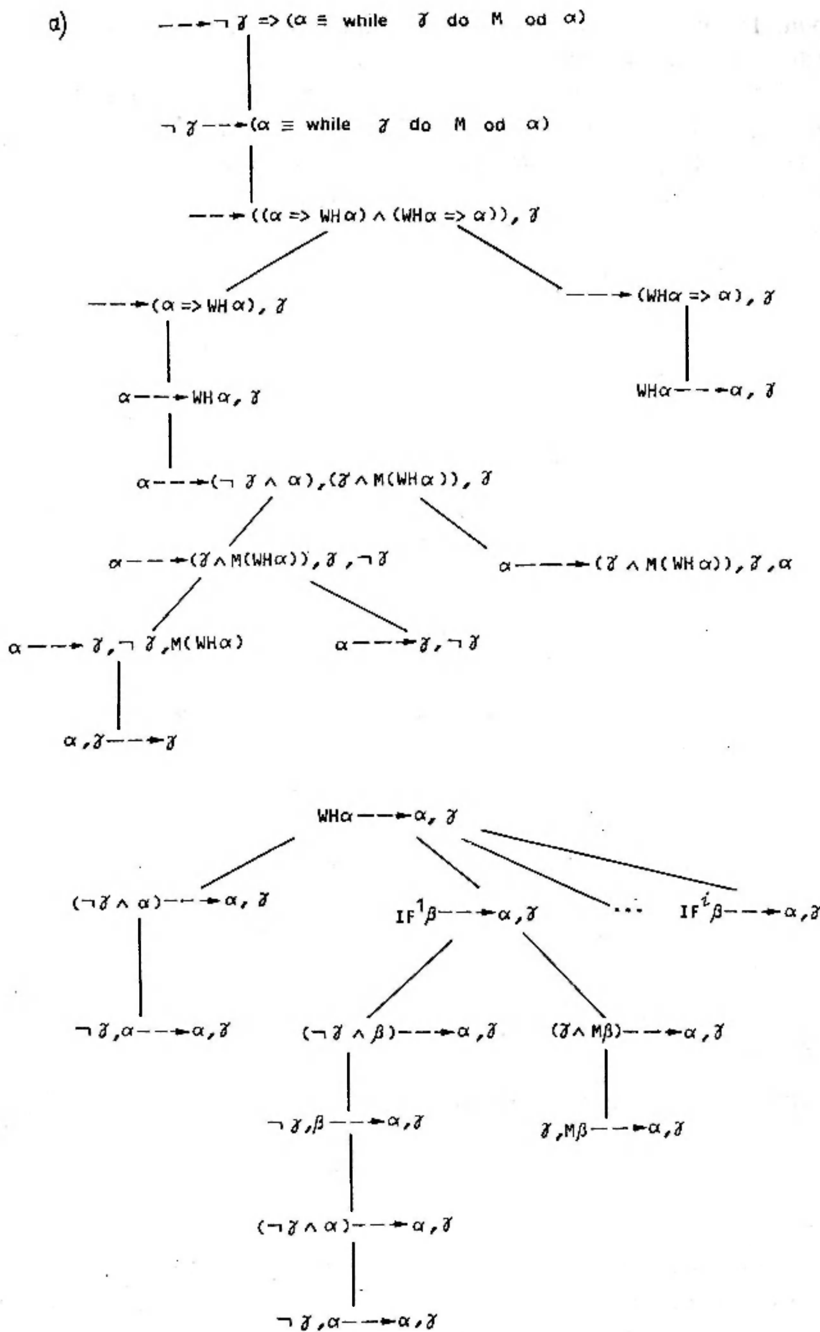
$$\neg\gamma \Rightarrow (\alpha \equiv \textbf{while } \gamma \textbf{ do M od } \alpha)$$

Dla uproszczenia przyjęliśmy następujące oznaczenia:

$$\mathrm{WH} \stackrel{\mathrm{df}}{=} \textbf{while } \gamma \textbf{ do M od}$$
$$\mathrm{IF} \stackrel{\mathrm{df}}{=} \textbf{if } \gamma \textbf{ then M fi}$$
$$\beta \stackrel{\mathrm{df}}{=} (\neg\gamma \wedge \alpha)$$

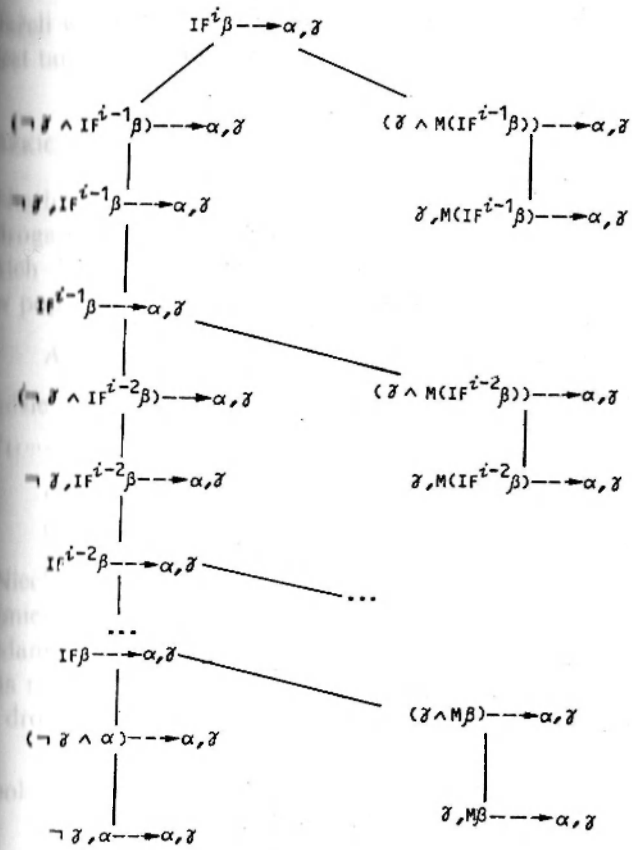
Diagram ten ma nieskończoną liczbę dróg, z których każda jest skończona. Jednak nie istnieje wspólne ograniczenie ich długości. □

Diagram formuły może być drzewem skończonym lub drzewem nieskończonym. Podobnie, jak w przypadku dowodu w systemie Hilberta (por. p. 4.2), diagram może być nieskończony z dwu przyczyn: albo istnieje rozgałęzienie nieskończone, albo istnieje droga nieskończona.




Rys. 4.6

ii)




RYS. 4.6

LEMAT 4.14

Jeżeli diagram formuły $\alpha$ jest drzewem, którego wszystkie drogi są skończone, a wszystkie etykiety liści są aksjomatami, to formuła $\alpha$ jest tautologią.

SZKIC DOWODU

Zauważmy, że jeżeli $\mathbf{\Gamma} \rightarrow \mathbf{\Omega}$ jest aksjomatem systemu GAL, to formuła odpowiadająca temu sekwentowi jest prawdziwa w każdej strukturze danych. Co więcej, jeżeli wszystkie formuły odpowiadające przesłankom pewnej reguły są tautologiami, to na mocy lematu 4.13 formuła odpowiadająca wnioskowi tej reguły też jest tautologią. Ponieważ (na mocy założenia) wszystkie drogi w rozważanym diagramie są skończone, więc za pomocą indukcji wykażemy, że wszystkie formuły odpowiadające etykietom diagramu są tautologiami. W szczególności formuła (**true** $\Rightarrow \alpha$), odpowiadająca korzeniowi jest tautologią, a co za tym idzie, $\alpha$ jest tautologią. □

LEMAT 4.15

Jeżeli $\alpha$ jest tautologią i wszystkie drogi w drzewie diagramu formuły $\alpha$ są skończone, to formuły odpowiadające liściom tego drzewa są aksjomatami systemu GAL.

DOWÓD

Niech $\alpha$ będzie tautologią, a $\langle D, d \rangle$ diagramem formuły $\alpha$, którego wszystkie drogi są skończone. Przypuśćmy, że dla pewnego wierzchołka końcowego $w$, sekwent $d(w) = \mathbf{\Gamma} \to \mathbf{\Omega}$ nie jest aksjomatem. Zatem $\mathbf{\Gamma}$ i $\mathbf{\Omega}$ są rozłączne oraz wszystkie formuły występujące w tym sekwencie są nierozkładalne. Na podstawie formuł sekwentu $\mathbf{\Gamma} \to \mathbf{\Omega}$ zbudujemy pewną strukturę danych i wskażemy w niej wartościowanie, które nie spełnia formuły $\alpha$.
Niech

$$\mathbf{A} = \langle \mathrm{T}, \{\varphi_{\mathbf{A}}\}_{\varphi \in \Phi}, \{\varrho_{\mathbf{A}}\}_{\varrho \in \Omega} \rangle$$

gdzie T jest zbiorem termów języka $\mathrm{L}(\pi)$ oraz dla dowolnych termów $\tau_1, \ldots, \tau_n \in \mathrm{T}$ przyjmujemy

$$\varphi_{\mathbf{A}}(\tau_1, \ldots, \tau_n) = \varphi(\tau_1, \ldots, \tau_n)$$
$$\varrho_{\mathbf{A}}(\tau_1, \ldots, \tau_n) = \mathbf{1} \quad \text{wtw} \quad \varrho(\tau_1, \ldots, \tau_n) \in \mathbf{\Gamma}$$

Niech $v$ będzie wartościowaniem w $\mathbf{A}$ takim, że

$$v(x) = x, \quad v(q) = \mathbf{1} \quad \text{wtw} \quad q \in \mathbf{\Gamma}$$

dla dowolnej zmiennej indywiduowej $x$ oraz dla dowolnej zmiennej zdaniowej $q$. (Powyższe definicje struktury danych i wartościowania są poprawne dzięki założeniu, że zbiory $\mathbf{\Gamma}$ i $\mathbf{\Omega}$ są rozłączne.)

Niech $\sigma_0$ będzie formułą odpowiadającą sekwentowi $\mathbf{\Gamma} \to \mathbf{\Omega}$. Z przyjętych definicji wynika natychmiast, że formuła $\sigma_0$ nie jest spełniona przez wartościowanie $v$ (wszystkie formuły występujące w poprzedniku sekwentu $\mathbf{\Gamma} \to \mathbf{\Omega}$ są prawdziwe, a wszystkie formuły występujące w następniku tego sekwentu są fałszywe przy wartościowaniu $v$),

$$\mathbf{A}, v \models \neg \sigma_0$$

Niech $w_0, \ldots, w_n$ będzie drogą w diagramie od wybranego wierzchołka końcowego $w$ do korzenia drzewa (tzn. $w = w_0$, $w_n = \emptyset$ oraz $w_i$ jest synem wierzchołka $w_{i+1}$). Niech $\sigma_0, \sigma_1, \ldots, \sigma_n$ będą formułami odpowiadającymi etykietom wierzchołków rozważanej drogi, w szczególności $\sigma_n = (\mathbf{true} \Rightarrow \alpha)$. Jeżeli dla pewnego $i < n$, $\mathbf{A}, v \models \neg \sigma_i$, to na mocy lematu 4.13 $\mathbf{A}, v \models \neg \sigma_{i+1}$. Zasada indukcji matematycznej pozwala nam więc wywnioskować, że wszystkie formuły $\sigma_0, \sigma_1, \ldots, \sigma_n$ są fałszywe przy wartościowaniu $v$. Zatem $\mathbf{A}, v \models \neg \alpha$ co przeczy założeniu, że formuła $\alpha$ jest tautologią. □

LEMAT 4.16

Jeżeli w diagramie formuły $\alpha$ istnieje droga nieskończona, to formuła $\alpha$ nie jest tautologią.

SZKIC DOWODU

Niech $\langle D, d\rangle$ będzie diagramem formuły $\alpha$. Przypuśćmy, że w $\langle D, d\rangle$ istnieje droga nieskończona. Niech $\theta N$ i $\theta P$ oznaczają, odpowiednio, zbiór wszystkich formuł występujących w następnikach i zbiór wszystkich formuł w poprzednikach sekwentów na tej drodze. Zdefiniujemy strukturę

$$\mathbf{A} = \langle \mathrm{T}, \{\varphi_{\mathbf{A}}\}_{\varphi\in\Phi}, \{\varrho_{\mathbf{A}}\}_{\varrho\in\Omega}\rangle$$

gdzie T jest zbiorem termów języka $\mathrm{L}(\pi)$ oraz dla dowolnych termów $\tau_1, \ldots, \tau_n \in \mathrm{T}$

$$\varphi_{\mathbf{A}}(\tau_1, \ldots, \tau_n) = \varphi(\tau_1, \ldots, \tau_n)$$
$$\varrho_{\mathbf{A}}(\tau_1, \ldots, \tau_n) = \mathbf{1} \quad \text{wtw} \quad \varrho(\tau_1, \ldots, \tau_n) \in \theta P$$

Niech $v$ będzie wartościowaniem w $\mathbf{A}$ takim, że $v(x) = x$ dla dowolnej zmiennej indywiduowej $x$ oraz $v(q) = \mathbf{1}$ wtw $q \in \theta P$ dla dowolnej zmiennej zdaniowej $q$. Oczywiście zbiory $\theta P$ i $\theta N$ są rozłączne, bo w przeciwnym razie na rozważanej drodze nieskończonej istniałby sekwent będący aksjomatem i droga byłaby skończona.

Przez indukcję ze względu na stopień komplikacji formuły, można pokazać, że dla dowolnej formuły $\beta$ zachodzą następujące własności:

$$\mathbf{A}, v \models \neg\beta \quad \text{dla } \beta \in \theta N, \qquad \mathbf{A}, v \models \beta \quad \text{dla } \beta \in \theta P$$

Ponieważ $\alpha \in \theta N$ zatem

$$\mathbf{A}, v \models \neg\alpha$$

a więc formuła $\alpha$ nie jest tautologią. □

Jako natychmiastowy wniosek z przedstawionych twierdzeń otrzymujemy

TWIERDZENIE 4.12

Formuła $\alpha$ jest tautologią wtedy i tylko wtedy, gdy wszystkie drogi w diagramie formuły $\alpha$ są skończone i wszystkie sekwenty końcowe są aksjomatami GAL. ■

# Algorytmiczne teorie struktur danych 5

## Od specyfikacji do implementacji 5.1

Potrzeba narzędzi, które umożliwiłyby szybkie wytwarzanie dobrego jakościowo oprogramowania, jest niewątpliwa. Wiele osób uważa, że w przyszłości będziemy posługiwać się przemysłowymi metodami produkcji oprogramowania. My podzielamy ten pogląd.

Zacznijmy od przykładu ilustrującego rozwijanie oprogramowania w sposób, który umożliwia wymianę i wielokrotne użycie modułów. Nasze uwagi ilustrują *zasadę faktoryzacji* sformułowaną po raz pierwszy przez Hoare'a [28]. Zauważył on, że — w większości przypadków — tworząc nowy system programistyczny mamy za zadanie ułożyć algorytm(y) wykonujący(e) operacje, jakich nie dostarcza komputer ani jego (zastane) oprogramowanie. Innymi słowy, nasz przyszły program ma być wykonywany w strukturze danych innej niż dostarczona nam przez sprzęt i systemowe oprogramowanie, w strukturze danych właściwej dla rozwiązywanego zadania. W 1972 r. Hoare zauważył, że w takim przypadku powinniśmy podzielić nasze zadanie na dwa podzadania:

(1) Zdefiniowanie i zrealizowanie odpowiedniej struktury danych;

(2) zaprojektowanie, analiza i uruchomienie programu abstrakcyjnego.

Zgodnie z tą radą powinniśmy stworzyć dwa moduły

```
Program            Moduł
abstrakcyjny       implementujący
```

Program abstrakcyjny charakteryzuje się tym, że może być łączony z różnymi implementacjami pojęć w nim występujących. Co więcej do jego wykonania jest niezbędne współdziałanie z modułem implementującym, który zawiera definicje użytych w programie abstrakcyjnych operacji. Program taki nie bierze też pod uwagę żadnych specyficznych własności komputera, na którym ma być wykonywany. Ogniwem, które ma wiązać program abstrakcyjny i moduł implementujący, jest moduł

Specyfikacja struktury danych

Zespół tworzący program korzysta z informacji o zadaniu i o własnościach struktury danych wymienionych w specyfikacji. Zespół tworzący moduł implementujący stosuje specyfikację jako kryterium poprawności implementacji. Czym więc jest specyfikacja? Jest to zbiór aksjomatów specyficznych (pozalogicznych) charakteryzujących strukturę danych.

Zalety zasady faktoryzacji są wielostronne. Umożliwia ona wykonywanie programu abstrakcyjnego w towarzystwie różnych modułów implementujących. Zmiana modułu implementującego nie powoduje żadnych zmian w programie. Możemy zyskać (bądź stracić) na efektywności obliczeń programu w zależności od wybranego modułu implementującego. Inną zaletą tej zasady jest możliwość wielokrotnego wykorzystania modułu implementującego dla potrzeb różnych programów.

Praca nad rozwiązaniem problemu programistycznego powinna więc mieć trzy etapy:

(1) sformułowanie specyfikacji (aksjomatyzacji) struktury danych;

(2) zaprojektowanie programu abstrakcyjnego i jego analiza (w tym etapie wykorzystuje się tylko specyfikację);

(3) implementacja struktury danych i weryfikacja jej poprawności (polega ona na sprawdzeniu czy są spełnione aksjomaty wymienione w specyfikacji).

Rozważmy bardziej szczegółowo następujący przykład. W nowym osiedlu ma powstać oddział banku. Przypuśćmy, że dyrektor banku zleca firmie produkującej oprogramowanie, wykonanie symulacji pracy oddziału. Chodzi o to, by w drodze eksperymentu symulacyjnego ustalić liczbę okienek i personelu tak, by nie tworzyły się kolejki klientów i by, z drugiej strony, nie zatrudnić zbyt wielkiej liczby personelu.

W tym przypadku całkiem naturalny jest pomysł utworzenia dwóch modułów

```
BANK
definiuje
pojęcia i operacje
typ KlientBanku
    Urzędnik
    Okienko
proc Przychodzi
     Wpłaca
       ...
```

```
SYMULACJA BANKU
używa
pojęć i operacji
var F,G: Okienko,
    A,B: KlientBanku
    C,D,E: Urzędnik
    ...
call Przychodzi
call Wpłaca
```

Dalsza analiza prowadzi do wniosku, że BANK jest szczególnym przypadkiem ogólniejszej struktury BIURO. W strukturze BIURO wystę-

pują pojęcia klient i obsługa, pojęcia klient banku, urzędnik, okienko stanowią rozwinięcia pojęć bardziej ogólnych. Podobnie ma się rzecz z operacjami: operacje klient przychodzi, czeka, wychodzi, są wspólne dla wielu różnych biur. Dla kierownika firmy produkującej oprogramowanie jest więc całkiem naturalne, by wyodrębnić i osobno zaimplementować moduł BIURO

```
BIURO
definiuje
pojęcia i operacje
typ Klient
    Obsługa
proc Wchodzi
     Wychodzi
```

Może on być stosowany do innych prac, np. do symulacji pracy firmy ubezpieczeniowej. Rozumowanie to można powtórzyć kilkakrotnie i wyodrębnić po kolei następujące moduły: SYMULACJA, KOLEJKA PRIORYTETOWA, KOLEJKA. W strukturze SYMULACJA mamy do czynienia z pojęciami: proces symulowany, zdarzenie (na zdarzenie składają się: nazwa procesu i chwila zdarzenia), oś czasu, na której są umieszczone zdarzenia zaplanowane do symulacji w przyszłości. Operacje struktury SYMULACJA to: zaplanuj, wykonaj, wznów, skasuj, bieżąca chwila, bieżący proces itp.

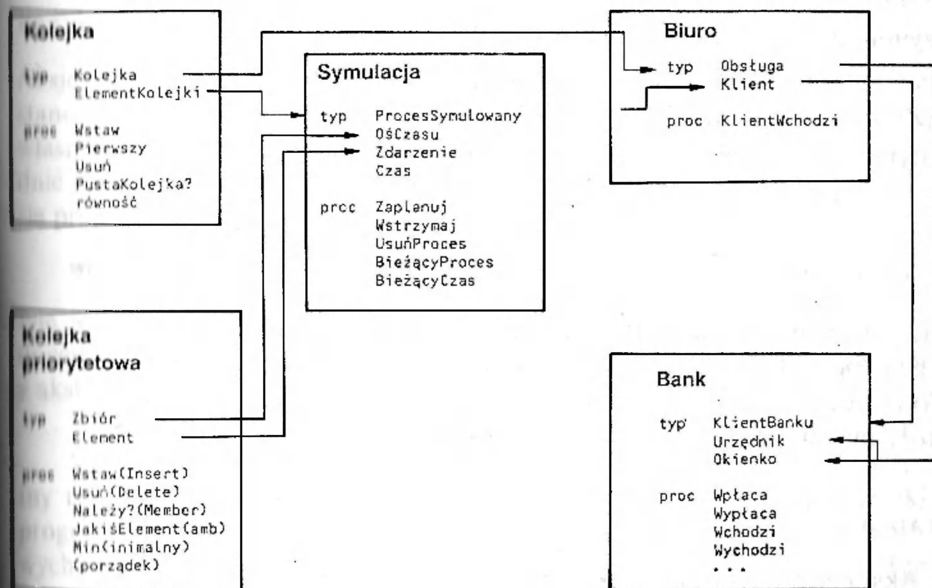
```
SYMULACJA
definicje pojęcia
typ Proces_Symulowany
    OśCzasu
    Zdarzenie
i operacje
proc Zaplanuj
     Wstrzymaj
     Usuń
     Bieżący_Proces
     Bieżący_Czas
```

Zdarzenia są elementami osi czasu, którą możemy pojmować jako kolejkę priorytetową (znaczenie tego terminu wyjaśnimy poniżej, por. p. 5.4, zob. też [1]). Procesy z kolei mogą być elementami kolejek (zwyczajnych, por. p. 5.3).

```
KOLEJKA_PRIORYTETOWA
definiuje pojęcia
typ Zbiór
    Element
i operacje
proc Wstaw (Insert)
     Usuń (Delete)
     Należy? (Member)
     Jakiś_Element(amb)
     Min(imalny)
     ≤ (porządek)
```

```
KOLEJKA
definiuje pojęcia
typ Kolejka
    Element_Kolejki
i operacje
proc Wstaw
     Pierwszy
     Usuń
     Pusta_Kolejka?
     równość
```

Tak więc doszliśmy do wniosku, że dla rozwiązania postawionego zadania symulacji oddziału banku, należy najpierw podać specyfikację, a następnie implementację dla pięciu struktur danych (rys. 5.1). Praca włożona w te specyfikacje oraz implementacje może znaleźć wielokrotne zastosowania. Struktury kolejek, kolejek priorytetowych i symulacji występują w niezliczonej ilości w najróżniejszych systemach programistycznych.




RYS. 5.1

W dalszym ciągu rozdziału przedstawimy, na konkretnych przykładach zaczerpniętych z omówionego zadania symulacji

(1) pracę nad aksjomatyczną specyfikacją struktur danych,

(2) nad implementowaniem struktur rozumianym jako interpretacja teorii jednej struktury w teorii innej struktury i

(3) zilustrujemy drogę od takich interpretacji do konstrukcji modułów programistycznych.

Metoda ta gwarantuje poprawność otrzymanego modułu względem aksjomatycznej specyfikacji.

## Specyfikacja struktur. Typy pierwotne 5.2

Punkt ten rozpoczniemy od uwag ogólnych dotyczących pojęcia specyfikacji struktur danych. Później przejdziemy do ilustracji metod specyfikowania pierwotnych typów danych występujących w ustalonym języku programowania. Dalsze punkty tego rozdziału przyniosą inne przykłady specyfikacji pewnych powszechnie stosowanych i niebanalnych struktur danych.

Przyjmijmy następujące oznaczenia:

**K** — klasa podobnych struktur danych;

$\mathfrak{M}(Z)$ — klasa wszystkich modeli zbioru formuł Z;

Ax(**K**) — pewien zbiór formuł w języku logiki algorytmicznej, o tej samej sygnaturze co struktury klasy **K**.

Definicja 5.1

Powiemy, że zbiór formuł Ax(**K**) jest specyfikacją klasy **K** wtedy i tylko wtedy, gdy $\mathfrak{M}(\mathrm{Ax}(\mathbf{K})) = \mathbf{K}$. Niech **A** będzie ustaloną strukturą danych. Zbiór Ax(**A**) będziemy nazywać specyfikacją struktury **A**, jeżeli $\mathbf{A} \models \mathrm{Ax}(\mathbf{A})$ oraz każdy model zbioru Ax(**A**) jest izomorficzny z **A**. ■

Przykład 5.1

(1) Aksjomaty algorytmiczne arytmetyki Ar (por. p. 4.4) specyfikują klasę struktur danych izomorficznych ze standardowym modelem arytmetyki liczb naturalnych. Wynika to z twierdzenia o kategoryczności algorytmicznej arytmetyki liczb naturalnych (por. twierdzenie 4.9).

(2) Aksjomaty kolejek (por. p. 5.3) stanowią specyfikację klasy standardowych kolejek, tzn. ciągów skończonych z odpowiednimi operacjami. Wynika to z twierdzenia o reprezentacji (twierdzenie 5.4). □

Lemat 5.1

Niech $Z$ będzie zbiorem formuł, a **K** klasą struktur podobnych. Jeżeli istnieje własność $\alpha$ taka, że nie można udowodnić jej ze zbioru formuł $Z$, a każda struktura klasy **K** jest modelem zbioru $Z \cup \{\alpha\}$, to $Z$ *nie jest specyfikacją* klasy **K**.

Dowód

Rzeczywiście, niech $\mathbf{A} \models \alpha$ oraz $\mathbf{A} \models Z$ dla dowolnej struktury $\mathbf{A} \in \mathbf{K}$. Jeżeli *non* $Z \vdash \alpha$, to na mocy twierdzenia o pełności logiki algorytmicznej (por. p. 4.4)

istnieje struktura danych $\mathbf{B}$ taka, że $\mathbf{B} \models Z$ oraz *non* $\mathbf{B} \models \alpha$. Wynika stąd, że $\mathbf{B} \in \mathfrak{M}(Z)$ i $\mathbf{B} \notin \mathbf{K}$, zatem $\mathfrak{M}(Z) \neq \mathbf{K}$.

Zatem jeżeli zbiór formuł $\mathrm{Ax}(\mathbf{K})$ jest specyfikacją klasy struktur $\mathbf{K}$, to dla dowolnej formuły $\alpha$ jest spełniona równoważność:

$$(\forall \mathbf{A} \in \mathbf{K}) \quad \mathbf{A} \models \alpha \equiv \mathrm{Ax}(\mathbf{K}) \vdash \alpha$$ □

PRZYKŁAD 5.2

Aksjomaty kolejek (por. p. 5.3) bez aksjomatu algorytmicznego Q6 nie stanowią specyfikacji klasy zwyczajnie rozumianych kolejek, bo istnieje własność prawdziwa w klasie kolejek skończonych, której nie można udowodnić na podstawie tych aksjomatów. Rozpatrzmy własność zatrzymywania się programu

**while** $\neg em(q)$ **do** $q := o(q)$ **od**

Nie jest ona prawdziwa w klasie wszystkich kolejek rozumianych jako dowolne ciągi (skończone lub nie). Wobec tego nie ma ona dowodu z aksjomatów Q1–Q5 i Q7. □

W każdym języku programowania występuje co najmniej jeden pierwotny typ danych, tzn. typ wprowadzony wraz z językiem. Na ogół języki programowania przewidują też możliwość wprowadzania nowych, zdefiniowych przez programistę typów danych, o czym później. Jeżeli pierwotny typ danych jest skończony, to jego opis nie nastręcza specjalnych trudności. Dla przykładu, rozważmy typ boolowski, który jest dwuelementowym zbiorem z odpowiednimi działaniami (por. p. 2.2). Wykażemy, że jest on w pełni opisany następującymi formułami AxBool w języku pierwszego rzędu z równością:

Bool0 $(\forall y)(\forall x)\,\big((y = (x \cap -x) \vee y = (x \cup -x)) \wedge \neg((x \cap -x) = (x \cup -x))\big)$

Bool1 $(\forall x, y, z)\big((x \cap (z \cap y)) = ((x \cap z) \cap y) \wedge (x \cup (y \cup z)) = ((x \cup y) \cup z)\big)$

Bool2 $(\forall x, y)\,((x \cap y = y \cap x) \wedge (x \cup y = y \cup x))$

Bool3 $(\forall x, y)\,\big((x \cap (x \cup y)) = (x \cup (x \cap y)) = x\big)$

Bool4 $(\forall x, y, z)\big((x \cap (z \cup y)) = ((x \cap z) \cup (x \cap y)) \wedge (x \cup (z \cap y)) = ((x \cup z) \cap (x \cup y))\big)$

Bool5 $(\forall x, y)\,\big((x \cap (y \cup -y)) = x \wedge (x \cup (y \cap -y)) = x\big)$.

Łatwo zauważyć, że dwuelementowa algebra Boole'a zdefiniowana w przykładzie 2.2

$$\mathbf{B}_0 = \langle \{\mathbf{1}, \mathbf{0}\}; \cap, \cup, - \rangle$$

jest modelem zbioru formuł AxBool. Co więcej, pokażemy, że wszystkie modele tego zbioru formuł są identyczne z dokładnością do izomorfizmu, tzn. wykażemy, że każdy model zbioru AxBool jest izomorficzny z dwuelementową algebrą Boole'a.

TWIERDZENIE 5.1

W klasie systemów relacyjnych, które spełniają aksjomaty AxBool wszystkie modele są izomorficzne.

DOWÓD

Rozważmy dowolny model **A** zbioru AxBool

$$\mathbf{A} = \langle A; \cap_{\mathbf{A}}, \cup_{\mathbf{A}}, -_{\mathbf{A}} \rangle$$

Na mocy aksjomatu Bool0 uniwersum modelu **A** ma dokładnie dwa elementy. Niech $A = \{prawda, fałsz\}$ i przyjmijmy

$$fałsz = -_{\mathbf{A}}\ prawda \cap_{\mathbf{A}} prawda \text{ oraz } prawda = -_{\mathbf{A}}\ fałsz \cup_{\mathbf{A}} fałsz$$

Na mocy aksjomatu Bool1 oba działania $\cap_{\mathbf{A}}$, $\cup_{\mathbf{A}}$ są łączne, a na mocy Bool2 oba działania są przemienne. Ponadto aksjomat Bool4 zapewnia rozdzielność jednego działania względem drugiego. Zgodnie z przyjętą interpretacją predykatu $=$ i na mocy aksjomatów Bool3, Bool5 operacje $\cap_{\mathbf{A}}$, $\cup_{\mathbf{A}}$, $-_{\mathbf{A}}$ są wszędzie określone.

Zauważmy proste własności modelu **A** prawdziwe dla dowolnych $a, b \in A$. Na mocy aksjomatu Bool5 i przyjętych założeń o elementach zbioru $A$

$$\begin{aligned} fałsz &= -_{\mathbf{A}}\ prawda \cap_{\mathbf{A}} prawda = -_{\mathbf{A}}\ prawda \cap_{\mathbf{A}} (-_{\mathbf{A}}\ fałsz \cup_{\mathbf{A}} fałsz) = \\ &= -_{\mathbf{A}}\ prawda \\ prawda &= -_{\mathbf{A}}\ fałsz \cup_{\mathbf{A}} fałsz = -_{\mathbf{A}}\ fałsz \cup_{\mathbf{A}} (\neg\ prawda \cap_{\mathbf{A}} prawda) = \\ &= -_{\mathbf{A}}\ fałsz \end{aligned}$$

na mocy aksjomatu Bool3

$$a \cap_{\mathbf{A}} a = a \cap_{\mathbf{A}} (a \cup_{\mathbf{A}} (a \cap_{\mathbf{A}} b)) = a$$
$$a \cup_{\mathbf{A}} a = a \cup_{\mathbf{A}} (a \cap_{\mathbf{A}} (a \cup_{\mathbf{A}} b)) = a$$

Ponieważ na mocy aksjomatu Bool5, $fałsz \cup_{\mathbf{A}} a = a$ oraz $prawda \cup_{\mathbf{A}} a = prawda$ zatem

$$a \cup_{\mathbf{A}} b = fałsz \quad \text{wtw} \quad a = fałsz \text{ i } b = fałsz$$

Analogicznie, na mocy aksjomatu Bool5, $fałsz \cap_{\mathbf{A}} a = fałsz$ oraz $prawda \cap_{\mathbf{A}} a = a$ zatem

$$a \cap_{\mathbf{A}} b = prawda \quad \text{wtw} \quad a = prawda \text{ i } b = prawda$$

Niech $h$ będzie odwzorowaniem struktury $\mathbf{A}$ w dwuelementową algebrę Boole'a $\mathbf{B}_0$ takim, że

$$h(fałsz) = \mathbf{0} \text{ i } h(prawda) = \mathbf{1}$$

Na mocy tych równości oraz definicji działań w dwuelementowej algebrze Boole'a mamy

$$h(a) \cap h(b) = \mathbf{1} \equiv h(a) = \mathbf{1} \text{ i } h(b) = \mathbf{1} \equiv a = prawda \text{ i } b = prawda \equiv$$
$$\equiv a \cap_{\mathbf{A}} b = prawda \equiv h(a \cap_{\mathbf{A}} b) = prawda$$
$$h(a) \cup h(b) = \mathbf{0} \equiv h(a) = \mathbf{0} \text{ i } h(b) = \mathbf{0} \equiv a = fałsz \text{ i } b = fałsz \equiv a \cup_{\mathbf{A}} b =$$
$$= fałsz \equiv h(a \cup_{\mathbf{A}} b) = fałsz$$

Stąd

$$h(a \cap_{\mathbf{A}} b) = h(a) \cap h(b)$$
$$h(a \cup_{\mathbf{A}} b) = h(a) \cup h(b)$$
$$h(\underset{\mathbf{A}}{-} a) = -h(a)$$

Wykazaliśmy w ten sposób, że dowolny model zbioru AxBool jest izomorficzny z dwuelementowym modelem $\mathbf{B}_0$, a tym samym pokazaliśmy, że zbiór aksjomatów AxBool ma, z dokładnością do izomorfizmu, jeden model. □

UWAGA

Teoria $T_0 = \langle L, \vdash, \text{AxBool} \rangle$ jest więc kategoryczna (por. p. 4.4). Model $\mathbf{B}_0$ jest jej modelem standardowym. Twierdzenie 5.1 można sformułować zatem w równoważnej postaci jako: każdy model teorii $T_0$ jest izomorficzny z modelem standardowym. ■

Rozważania te pozwalają uznać, że formuły AxBool stanowią specyfikację (formalną definicję) typu Boolean. Podobnie można określić systemy urządzeń zewnętrznych, jakimi dysponuje komputer: ekran, klawiaturę czy też drukarkę.

Zupełnie inaczej ma się rzecz z typem **integer**. Postawmy sobie zadanie opisania klasy skończonych zbiorów liczbowych wraz z operacjami odpowiadającymi dodawaniu, odejmowaniu i innym działaniom arytmetycznym. Ponieważ różne komputery dopuszczają różne podzbiory zbioru $Z$ liczb całkowitych, a pomimo to każdy taki zbiór chcemy traktować jako dopuszczalną interpretację typu **integer**, trzeba uznać, iż naszym celem jest właśnie opisanie takiej nieskończonej klasy struktur danych, z których każda ma skończoną ilość elementów. Zauważmy, że zadanie takie nie jest rozwiązywalne, jeżeli do opisu chcemy użyć wyłącznie formuł pierwszego rzędu. Okazało się bowiem (por. p. 2.5), że własności „być zbiorem skończonym”,